PREFAZIONE

L'obiettivo principale di questa tesi è quello di accompagnare il lettore, passo per passo, allo sviluppo di un pacchetto software.
Si cerca di dimostrare attraverso quali fasi si svolge il lavoro: la definizione del problema, le scelte da effettuare, l'analisi dei dati e delle procedure fino alla realizzazione pratica del programma.
Inoltre sono stati introdotti aspetti moderni di programmazione, quali lo sviluppo di applicazioni in ambienti con interfacce grafiche, in questo specifico caso Microsoft Windows.

CAPITOLO 1
Introduzione

1.1 Incontro con l'azienda

Come studente della Scuola diretta a fini speciali in informatica, è previsto un periodo di stage presso un'azienda, ai fini della stesura di una tesi da presentare e discutere in sede di diploma. Quando sono state proposte le aziende, io ho scelto la "**Coop Bilanciai**" di Campogalliano (MO) e, per essere sincero, la scelta è dovuta a venali motivi di comodità (essendo vicino alla mia abitazione).

La "Coop Bilanciai" è un'azienda di discrete dimensioni, circa 200 dipendenti; è un'azienda manifatturiera (dal punto di vista giuridico una *cooperativa*) che lavora su commessa. Essa produce esclusivamente bilance industriali, da bilance per mercati ortofrutticoli fino a pese per treni o macchine per trasporto su strada; bisogna comunque dire che è l'azienda leader in Italia per quanto riguarda questi prodotti.

Chiaramente il mio ambito lavorativo è il centro di calcolo (CED) della Coop Bilanciai; un ufficio, composto da due programmatori, che si occupa di tutta la gestione del Sistema Informativo aziendale.

L'incontro con l'azienda è stato molto positivo, meno positiva è la situazione in cui essa si trova, in questo momento, dal punto di vista informatico. La Coop Bilanciai è in una fase un po' particolare, infatti si sta completamente sostituendo il vecchio sistema informatico (sviluppato internamente) con un nuovo sistema, tramite un pacchetto modulare che copre tutte le aree

aziendali acquistato all'esterno. Questo, come si può immaginare, è un processo lento e difficile da attuare, in quanto provoca innumerevoli problemi.

Dopo aver conosciuto l'azienda, la sua organizzazione, il suo Sistema Informativo e i pacchetti software da essa utilizzati, mi è stata proposta la realizzazione di un progetto. Questo progetto software rientra nel nuovo Sistema Informatico; questo significa ad esempio che vengono già codificati i dati secondo le caratteristiche del nuovo sistema informativo.

1.2 Quale lavoro e' stato proposto

L'azienda, ordinariamente e straordinariamente, si trova a dover compiere una serie di manutenzioni riguardanti i macchinari utilizzati, le attrezzature dei macchinari stessi, gli impianti ecc.
Svariati possono essere i soggetti da manutenere, ad esempio:

❶ un reparto in cui c'è da spostare dei materiali;
❷ una macchina (cespite) a controllo numerico alla quale c'è da cambiare l'acqua;
❸ un'attrezzatura da riparare;
❹ l'impianto idraulico ed elettrico da riparare ecc..

Da questi esempi si può benissimo capire che la mole di lavoro che coinvolge il discorso manutenzioni, non è del tutto indifferente per l'azienda. Questi lavori vengono svolti da personale interno o da terzi fornitori, con materiale interno (magazzino) o acquistato all'esterno, quindi

con utilizzo di risorse aziendali; in ogni caso il costo della manutenzione dovrà gravare sul prezzo del prodotto finito.
Tutta la gestione relativa alle manutenzioni è stata fino ad ora tenuta manualmente, ma con la riorganizzazione di tutto il sistema informativo anche questa deve essere automatizzata. Quindi il mio compito è quello di sviluppare un pacchetto che si occupi della gestione di queste manutenzioni.

1.3 Perché questa proposta

Si è già intuito, nella precedente sezione, il bisogno di questo pacchetto; vediamo ora di puntualizzare in modo più formale cosa vogliamo ottenere da questo programma.
Sostanzialmente sono **tre** gli obiettivi da raggiungere con la realizzazione e l'uso del pacchetto:

1). Organizzare la gestione delle manutenzioni

L'obiettivo principale è quello di organizzare il lavoro e renderlo il più efficiente possibile. Per le manutenzioni periodiche, ad esempio, chi si occupa delle manutenzioni deve conoscere in ogni momento:

- quando intervenire su un soggetto manutenibile;
- se il lavoro è fatto internamente o esternamente all'azienda;
- come intervenire, con quali risorse (personale e materiali);
- la disponibilità di risorse.

2). Determinare i costi

Chi si occupa delle manutenzioni deve poter quantificare i costi relativi a un determinato soggetto in qualsiasi momento. Questo è vitale per l'azienda, in quanto il costo deve comparire nella contabilità industriale, essere ripartito opportunamente ed accumulato al prezzo del prodotto finito.

3). Partecipare alla programmazione aziendale

Un'efficace gestione delle manutenzioni, partecipa alla conoscenza della realtà aziendale, e quindi ne migliora la qualità.
Ad esempio, dall'analisi statistica di dati storici sulle manutenzioni, si può capire quando un cespite è ormai obsoleto, e quindi richiede troppe manutenzioni, e valutare così se è il caso di acquistarne uno nuovo.

CAPITOLO 2
Scelta degli strumenti hardware e degli ambienti di lavoro

2.1 Scelta del calcolatore

La "Coop Bilanciai" possiede un **AS400** con collegati circa 50 terminali, quasi tutti PC, quindi è stato scelto di sviluppare il pacchetto su un personal computer con sistema operativo **Dos**, ossia quello adottato dall'azienda e tutto sommato più diffuso come standard internazionale.
Sarebbe stato possibile sviluppare il pacchetto in ambiente AS400, ma la gestione delle manutenzioni è una parte isolata ed autonoma ed è gestita da una sola persona, quindi si è ritenuto più giusto non intralciare il sistema informatico, già di per se molto complesso, e sviluppare il pacchetto su pc. Non ci sono problemi di capacita' di elaborazione in quanto la mole di dati che il pacchetto deve elaborare permette discrete prestazioni sia su **386** che **286**, macchine in dotazione all'azienda.

2.2 Scelta dell'ambiente operativo

La scelta più ardua è stata sicuramente quella dell'ambiente operativo, sicuramente un ambiente operante su un pc; scartato unix, non utilizzato dalla azienda, se non su macchine di tipo **RISC**, sono rimasti **Dos** e **Windows**. La strada Dos sarebbe stata molto più semplice, visto le mie discrete conoscenze del sistema e la possibilità di ottenere vari aiuti da

persone più esperte, sia nell'ambito universitario sia presso l'azienda dove ho svolto lo stage; invece è stato preferito Windows !

Perché Windows? Sicuramente non è possibile prevedere il futuro, ma una cosa sembra certa, ed è che Windows diventerà uno standard di fatto. Ormai grosse software houses (Microsoft, Borland International Inc. ecc..) sviluppano i loro pacchetti esclusivamente sotto Windows, e tutte le aziende stanno spostando le loro applicazioni verso il fortunato ambiente operativo della Microsoft. Non fa eccezione la "Coop Bilanciai" che già adesso si collega (in emulazione terminale con una scheda ethernet) da Windows direttamente con l'AS400.

2.2.1 Cos'è Windows

Prima di tutto cerchiamo di capire cos'è Windows, quali sono le sue caratteristiche e quali vantaggi offre, al fine di motivare la scelta che è stata fatta.

Windows non è un Sistema Operativo, anche se svolge molti compiti propri di quest'ultimo. Viene spesso usato il termine "Ambiente Operativo" che mette in risalto la gradevole e semplice interfaccia di Windows. Windows è, comunque, molto più di un ambiente con una interfaccia user-friendly; le sue maggiori qualità sono legate all'indipendenza del codice delle applicazioni dall'hardware sottostante e agli innumerevoli strumenti predefiniti che offre (soprattutto per la grafica).

Un PC è costituito da un'insieme di parti, che sono fondamentalmente dei circuiti: *cpu, memoria, tastiera, schermo, dischi* ecc.. La comunicazione tra software e questi circuiti avviene in vario modo: *interrupt* effettuati da un elemento dell'hardware che generano una chiamata di funzione ad un indirizzo in dipendenza del tipo di interrupt, *operazioni di mappatura* a degli indirizzi in memoria (come avviene per lo schermo) oppure utilizzo delle porte di I/O.

Per rendere il software relativamente indipendente dall'hardware ed offrire una programmazione più semplice rispetto al controllo diretto dell'hardware, i PC integrano un primo strato di software, il **BIOS**. Al di sopra del BIOS si colloca il sistema operativo, per esempio il Dos. Quest'ultimo è un sistema che adopera le primitive del BIOS per accedere all'hardware ed offre servigi più sofisticati, in particolare per la gestione dei dischi e dei file.

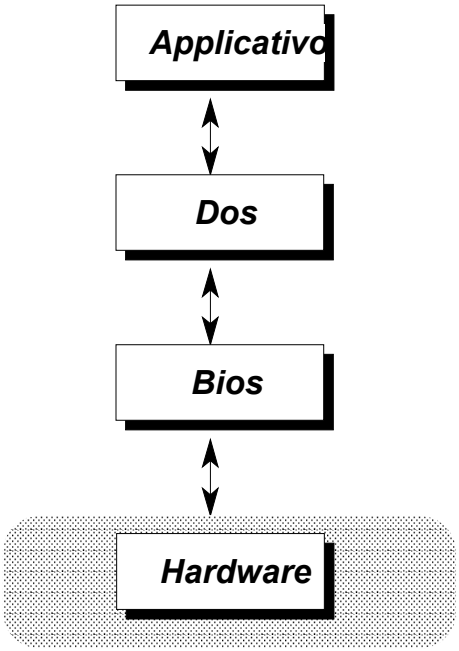

Figura 2.1 Accesso all'hardware attraverso DOS e BIOS

Un'applicazione sviluppata per un PC sotto Dos accede normalmente all'hardware attraverso il Sistema Operativo stesso (figura 2.1).
Qui iniziano i problemi, in quanto il Dos è carente nella gestione della tastiera e del video, di conseguenza molti programmatori preferiscono scavalcarlo e accedere all'hardware chiamando direttamente le primitive del BIOS, se non addirittura lo stesso hardware (figura 2.2).

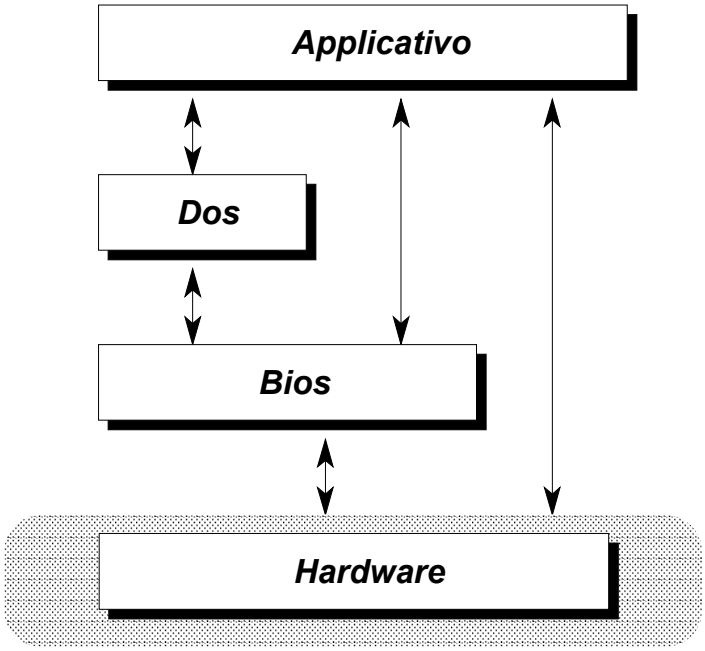

Figura 2.2 Accesso diretto all'hardware

È non è tutto, ad esempio l'integrazione di un nuovo elemento hardware, non noto al BIOS, è difficoltosa: bisogna aggiornare il BIOS oppure l'applicazione deve accedere direttamente al nuovo elemento.
Windows è stato creato per evitare al programmatore di preoccuparsi del tipo di hardware sul quale gira il proprio applicativo. Tutte le periferiche abituali di un microcomputer sono virtualizzate: *tastiera, mouse, video, stampante, tavoletta grafica, mezzi di comunicazione,* offrendo un'interfaccia

standard, perfettamente definita e completa, totalmente indipendente dall'hardware installato.

2.2.2 Confronto Dos-Windows

Nel paragrafo precedente si sono già messe a fuoco differenze tra Dos e Windows, e in fondo è inevitabile a questo punto non confrontarli. Ecco cosa offre in più Windows rispetto allo standard Dos:

① una interfaccia grafica caratterizzata da finestre, menu, controlli ecc..

② code di input;

③ device grafiche indipendenti;

④ multitasking.

✪ Interfaccia grafica

Uno degli obiettivi principali di Windows è dare accesso agli utenti a tutte le applicazioni nello stesso tempo. Questo è stato realizzato tramite la condivisione dello schermo tra le applicazioni. Windows assegna ad ogni applicazione una parte dello schermo e quindi l'utente ha sottocchio tutte le applicazioni.

Un'applicazione condivide il display usando una finestra, ossia un rettangolo che rappresenta una porzione dello schermo, in realtà una finestra è una combinazione di device, come *menu', scrollbar* ecc.. Quindi a differenza del Dos, dove lo schermo è tutto a disposizione, il programmatore, all'inizio del suo programma, deve sempre crearsi la propria finestra di lavoro.

✪ Code di Input

Una delle principali differenze tra Dos e Windows è come viene ricevuto l'input. In Dos si fa una chiamata a una funzione che restituisce ad esempio un carattere letto dalla tastiera. Windows riceve tutto l'input dalla tastiera, dal mouse, dal timer e lo mette in una *coda messaggi*. Quando l'applicazione deve leggere dell'input, legge il prossimo messaggio nella coda.

Un messaggio è una struttura più complessa dell'input del Dos, ma permette di trattare nello stesso modo tutto l'input indipendentemente da dove esso arriva.

Questo meccanismo permette di virtualizzare la tastiera, Windows contiene le *device drivers* di un certo numero di tipi diversi di tastiera; queste device drivers convertono i caratteri in messaggi standard che Windows riconosce.

✪ Device grafiche indipendenti

In Windows si ha accesso ha un ricco insieme di operazioni su device grafiche, questo significa che l'applicazione può facilmente disegnare cerchi, ellissi nella propria regione. Inoltre Windows virtualizza lo schermo, o meglio tutto l'output, quindi i comandi sono unici per scrivere o disegnare su una CGA come su una VGA, e si usano le stesse funzioni anche per stampare o per usare un plotter.

Anche qui Windows ha bisogno di una device driver per convertire il comando in codice per la periferica, quindi risulta molto semplice (a differenza di come visto con il BIOS) adottare una nuova periferica purché Windows o il costruttore ci fornisca la device driver ad essa associata.

✪ Multitasking

Un discorso a parte merita il *multitasking*, in considerazione del fatto che tutti i testi da me consultati parlano di Windows come un tale sistema. In realtà in Windows il termine Multitasking è inteso come condivisione di risorse; a differenza del Dos, un'applicazione non occupa tutte le risorse, ma le condivide con tutte le altre applicazioni. Esiste multiprogrammazione in quanto più processi possono stare contemporaneamente in memoria, ma i processi non sono certo spezzati in task ed eseguiti in modo concorrente.
In sistemi di multitasking reale mentre un processo effettua operazioni di I/O, la CPU passa ad elaborare un altro task. Io stesso ho provato, in ambiente Windows, ad eseguire programmi che utilizzano molto I/O; quando si cambia applicazione essi non continuano ad elaborare in background come prevede il multitasking reale.

2.2.3 Punto di vista del programmatore

Vediamo ora di capire cosa cambia per il programmatore abituato a sviluppare programmi sotto Dos quando entra nel mondo Windows. Windows purtroppo presenta un duplice aspetto; se da una parte per l'utente finale risulta molto semplice da utilizzare, potente e di immediato apprendimento, dall'altra risulta (almeno in un primo impatto) confuso, di difficile accesso per il programmatore.

❍ Processo di costituzione di un applicativo

Mettiamo in evidenza la differenza procedurale di costituzione di un applicativo Dos (figura 2.3) e di un applicativo Windows (figura 2.4).

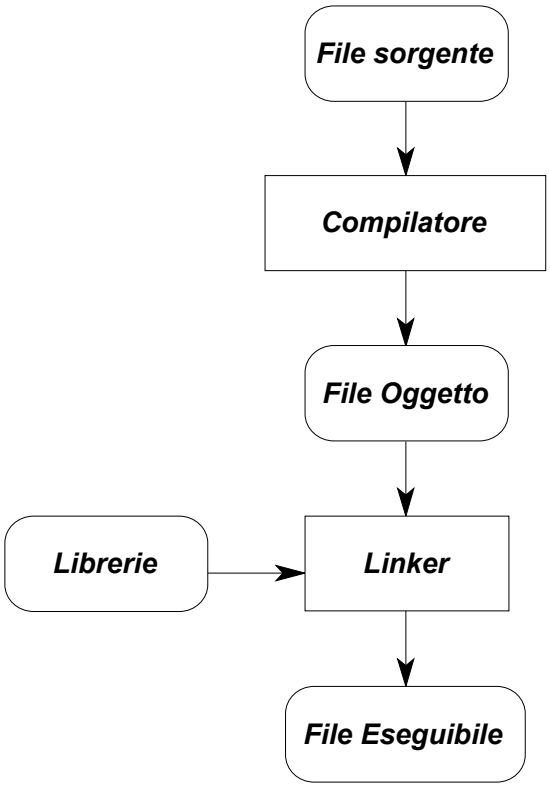

Figura 2.3 Applicativo sotto DOS

Come si può vedere dalle figure il processo in Windows si complica un po', nasce una attività parallela rappresentata dalla compilazione delle risorse. Il file sorgente di base è il *file delle risorse* (di estensione **RC**), scritto in un linguaggio descrittivo particolare. Esso contiene la descrizione delle risorse utilizzate dall'applicativo, può trattarsi di: *cursori, immagini bitmap, icone, menu* ecc..

Le risorse devono essere compilate con un apposito compilatore (**Resource Compiler**) ottenendo così il file oggetto di estensione .**RES**; quest'ultimo deve essere collegato, tramite il linker, all'eseguibile dal medesimo Resource Compiler.

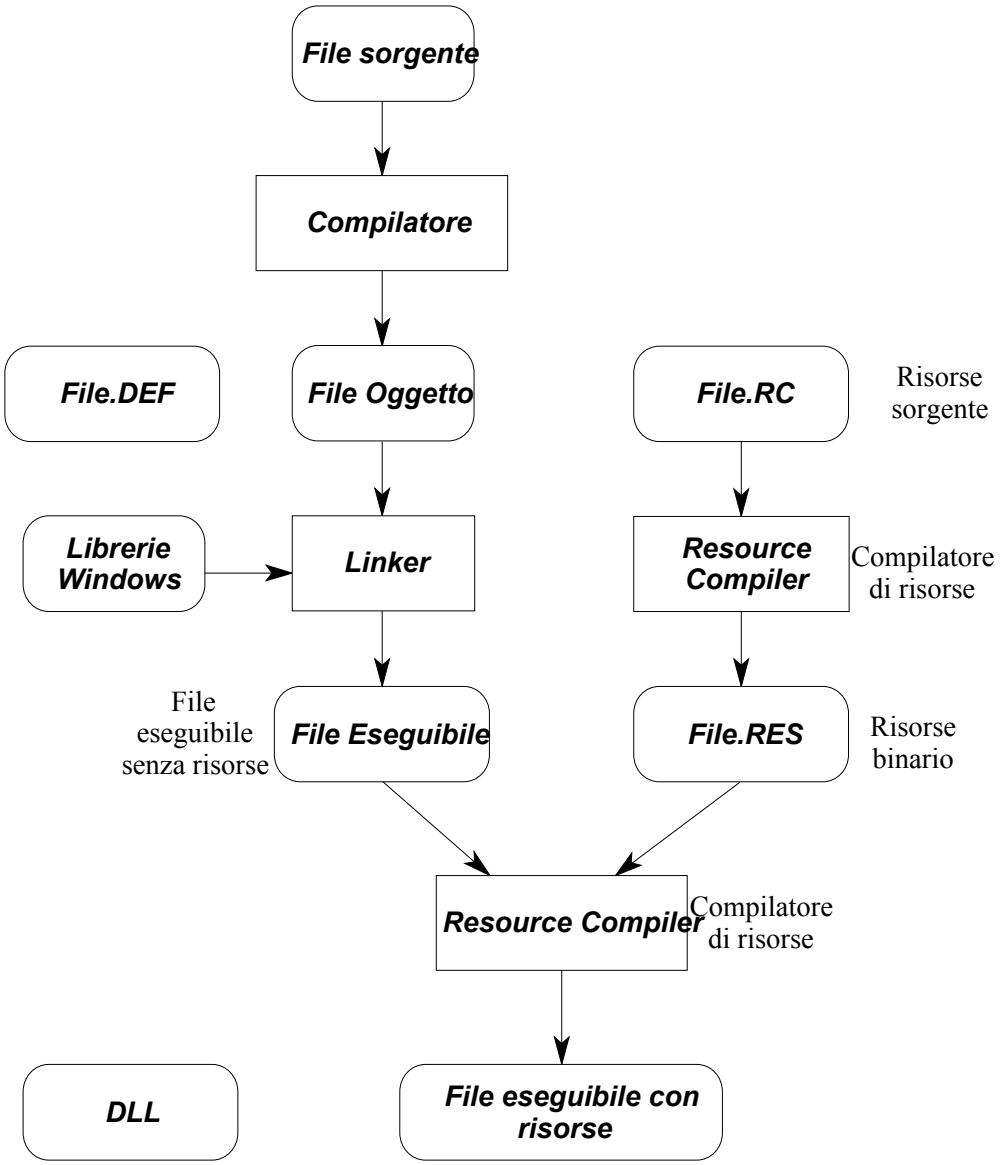

Figura 2.4 Applicativo sotto WINDOWS

Altre due novità sono i files di estensione .**DEF** che devono contenere informazioni riguardo la memoria da assegnare all'applicativo in Windows e le *librerie dinamiche*, ossia librerie di funzioni che vengono collegate in fase di esecuzione.

❍ Struttura del programma principale

Abbiamo parlato dei messaggi e di come viene processato l'input in ambiente Windows, questo comporta per il programmatore il cambiamento della struttura dei propri programmi. Generalmente la struttura di un programma sotto Dos è quella presentata in figura 2.5.

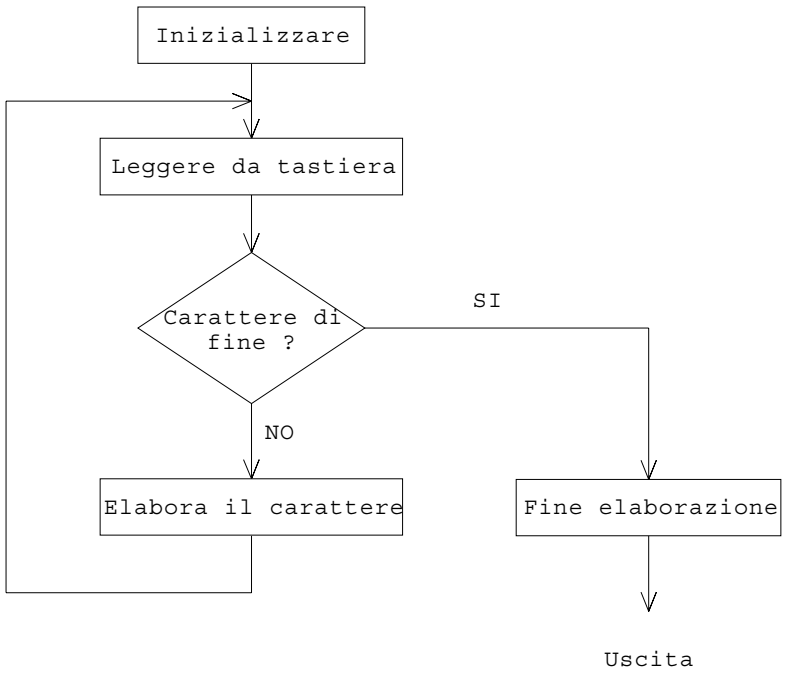

Figura 2.5 Struttura di un programma DOS

Non differisce molto il corpo principale di una applicazione Windows se non per il tipo di input (figura 2.6).

Quindi non si leggono più caratteri, ma messaggi che possono provenire da innumerevoli fonti.

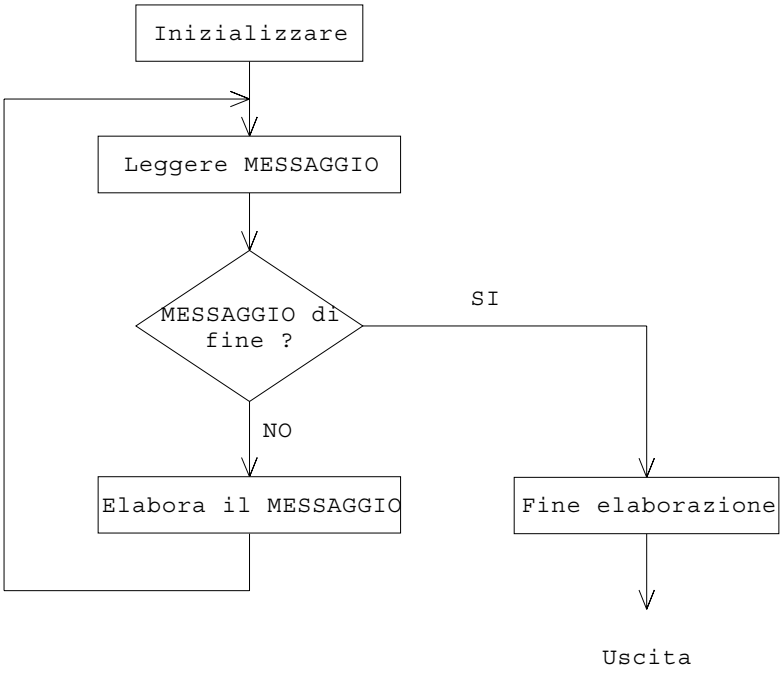

Figura 2.6 Struttura di un programma WINDOWS

❍ Message Loop

Vediamo di approfondire meglio come i messaggi vengono elaborati da Windows e di conseguenza come devono essere trattati dal programmatore. Windows colleziona tutto l'input nella *system queue* (coda unica nel sistema) e trasferisce i messaggi nelle varie *application queues* (una coda per ogni applicazione) (figura 2.7).

Ogni applicazione deve essere in grado di accettare i messaggi, ma a differenza del Dos, essi non vengono elaborati direttamente, chiamando ad esempio una funzione, ma vengono distribuiti a ogni "Window function".

Qualunque applicazione in Windows è formata da:

- una funzione principale (come il main del c) denominata WinMain;
- un insieme di funzioni, dette window functions, ognuna delle quali è collegata a una finestra del mio applicativo.

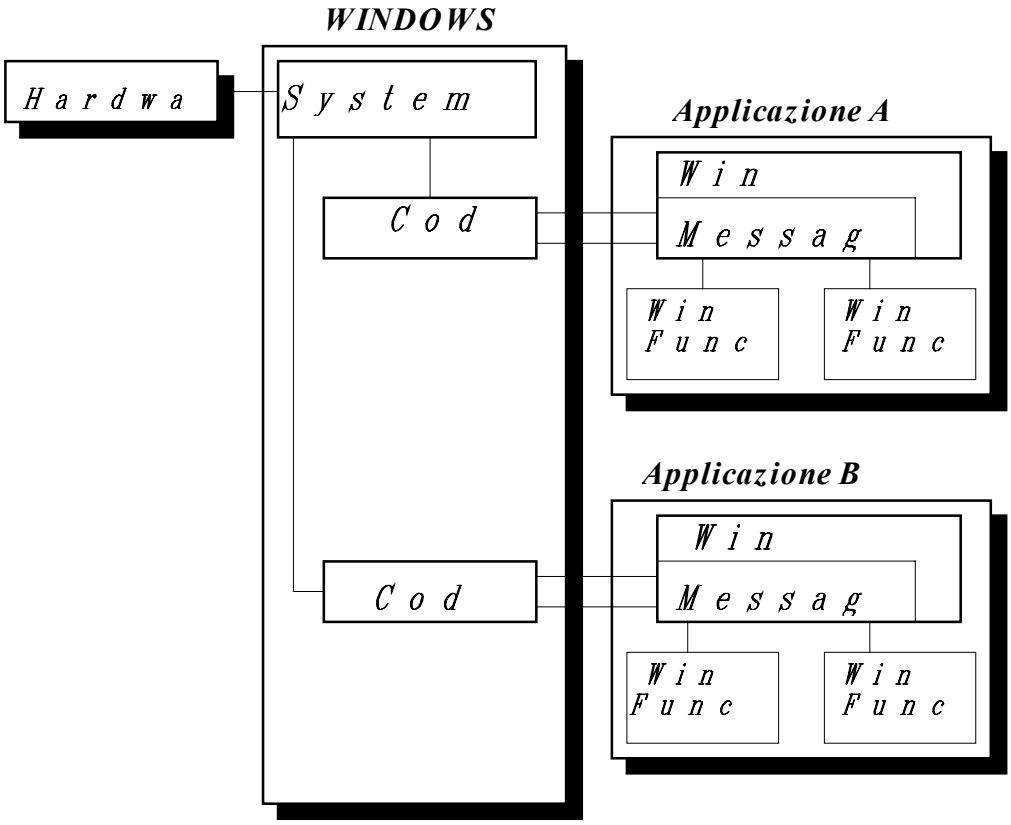

Figura 2.7 Trattamento dei messaggi in WINDOWS

La WinMain deve quindi leggere tutti i messaggi e distribuirli alle varie window functions che si prenderanno carico di eseguire certe operazioni per ognuno dei possibili messaggi.

Per fare questo ogni applicazione ha un "message loop", ossia un ciclo di lettura e distribuzione messaggi.

Il programma, quindi, interagisce continuamente con Windows, non ci sarà mai una chiamata diretta a una funzione associata a una finestra, ogni collegamento tra window functions del mio programma avviene attraverso un messaggio.

2.3 Scelta del linguaggio di programmazione

La scelta del linguaggio di programmazione da utilizzare, per implementare il pacchetto, risulta in genere difficile. In questo caso, i problemi da risolvere sono soprattutto legati all'ambiente in cui si è deciso di lavorare; in ambiente windows, infatti, non sono molte le possibilità, almeno nel momento in cui scrivo questa tesi di diploma.

Cosa usare per realizzare un pacchetto di tipo gestionale in ambiente windows? questa domanda ha trovato risposta solamente osservando l'offerta del mercato.

Attualmente non esistono discreti **DBMS** in ambiente windows; i linguaggi che offrono la possibilità di sviluppo sotto windows sono: *il 'C' ed il pascal*. Sinceramente avrei preferito il pascal, in quanto lo conoscevo già da tempo, ma l'azienda ha scelto il 'C', questo principalmente per tre motivi:

① il 'C' è un linguaggio che predilige l'efficienza, e questo è un'aspetto molto gradito dall'azienda;

② l'azienda svolge tutte le mansioni di tipo gestionale in ambiente AS400, relegando ai PC solo funzioni statistiche (lotus 123), di office automation (word processor), ma soprattutto di automazione industriale (programmazione di memorie EEPROM per le bilance elettroniche); per questo l'unico linguaggio conosciuto, all'interno dell'azienda, nel mondo personal computers è il 'C';

③ come vedremo dopo, è stato trovato uno strumento che facilita, sotto 'C', la gestione degli archivi; questo ha avuto un peso importante nella decisone finale.

Alla fine, la scelta è stata obbligata (ma comunque gradita), in quanto l'azienda ha adottato come linguaggio di programmazione il 'C'; quindi, in previsione di manutenzioni future al pacchetto, è stato preferito un linguaggio conosciuto nell'ambito aziendale.
Questo mi ha causato non pochi problemi in quanto ho dovuto apprendere un nuovo linguaggio di programmazione, per giunta neanche molto semplice.

2.4 Scelta dell'ambiente di programmazione, Borland c++

Due sono gli ambienti 'C' in cui è possibile sviluppare sotto Windows: *Borland C++ e Microsoft C*. È stato scelto il primo in quanto presente in azienda, ma soprattutto perché ambiente di lavoro da me già conosciuto; avendo già utilizzato l'ambiente integrato del Turbo Pascal della stessa Borland. La scelta di questo ambiente è stata anche influenzata dall'annuncio della futura commercializzazione del "Borland C++" per Windows.

2.5 Scelta Database, Paradox Engine

Potrebbe sembrare inopportuno utilizzare un linguaggio come il 'C' per una applicazione fondamentalmente gestionale (sarebbe stato più semplice utilizzare *clipper, dbIII* o qualche DataBase sotto Windows).
Innanzitutto il mercato attualmente (al momento della scelta) non offre una grossa scelta di database sotto Windows; per cui la soluzione trovata è stata quella di adottare "*Paradox Engine*". Paradox Engine non è un Data Base,

bensì un'insieme di librerie pascal, 'C' e Windows contenenti funzioni che permettono di accedere alle tabelle di Paradox (famoso e diffuso DataBase). Queste funzioni permettono di accedere in modo navigazionale (o procedurale) alle tabelle, questo significa che le tabelle vengono manipolate tramite puntatori ai loro records, e non come insieme di dati come avviene in un linguaggio di tipo SQL.

CAPITOLO 3
Analisi del pacchetto

3.1 Definizione del problema

3.1.1 Problematiche principali

Abbiamo già introdotto nel capitolo 1 il problema al quale si è cercato, con questo pacchetto, di dare un'efficiente soluzione; vediamo ora di capire meglio le varie problematiche, inquadrandole soprattutto sotto il profilo informatico.

All'interno dell'azienda attualmente una persona sola si occupa delle manutenzioni, questa persona è un punto di riferimento per tutti i dipendenti. Questo significa, ad esempio, che se si rompe il telefono di un ufficio, l'abituale utilizzatore dell'apparecchio telefonico fa presente all'addetto delle manutenzioni ciò che è accaduto. L'addetto delle manutenzioni si prende l'incarico di telefonare alla Sip e richiedere la riparazione dell'apparecchio. La stessa cosa accade se si rompe un calcolatore, l'impianto idraulico o elettrico, ma anche se si guasta una macchina a controllo numerico o una semplice attrezzatura.

Il pacchetto deve mettere in condizione l'addetto alle manutenzioni di intervenire tempestivamente; per fare questo occorre che egli sappia in qualsiasi momento: a chi bisogna rivolgersi per l'intervento, se a un fornitore di servizi esterno oppure a personale interno all'azienda; quali

ricambi utilizzare, se materiale di magazzino oppure occorre richiedere la fornitura di nuovi pezzi.

Fino ad ora ho parlato di quelle manutenzioni cosiddette *urgenti* o *straordinarie*, ma esiste anche tutta una serie di interventi ordinari o programmati. Un esempio può essere quello del ricambio dell'acqua refrigerante in una fresatrice, quest'acqua serve per raffreddare gli utensili metallici; l'acqua viene continuamente riciclata e fatta scorrere nella macchina, ma periodicamente deve essere sostituita, altrimenti porta a sgradevoli conseguenze.

In questi casi non è chi lavora sulla macchina che deve informare l'addetto alle manutenzioni, ma è quest'ultimo che, essendo a conoscenza dei precedenti interventi, deve ricordarsi quando operare sulla macchina e informare di conseguenza chi di dovere per compiere la manutenzione.

Il pacchetto deve permettere al responsabile delle manutenzioni di memorizzare in modo preventivo, all'inizio dell'anno, tutte le manutenzioni programmate, e successivamente di trasformare la manutenzione prevista in manutenzione consuntiva, completando tutti i dati relativi alla manutenzione.

3.1.2 Archivi gestiti

Può sembrare inopportuno iniziare ad analizzare il pacchetto individuando gli archivi che vengono gestiti. In realtà la scelta è appositamente voluta per due motivi:

1. durante i colloqui con il mio responsabile in azienda e con l'addetto alle manutenzioni, si sono subito messi in evidenza le informazioni di cui

avevano bisogno e i raggruppamenti di questi dati; in sostanza si sono intuite immediatamente le anagrafiche da gestire.

2. il secondo motivo è di ordine mentale, infatti la mia analisi, come avremo modo di capire prossimamente, è di tipo bottom-up. Il mio obiettivo è di arrivare a comprendere le operazioni che il pacchetto deve fare sui dati, successivamente si cerca da queste operazioni di estrapolare tutta la struttura del pacchetto stesso (classico approccio bottom-up).

Chiaramente per determinare le operazioni sui dati, dobbiamo introdurre una spiegazione di quali dati il pacchetto gestisce, opportunamente raggruppati nei propri archivi. Vediamo ora quali sono gli archivi da gestire, senza spingerci troppo nel dettaglio in quanto verranno spiegati meglio in sede di documentazione:

◆ **Anagrafica RDL o richieste di lavoro**

Questo archivio serve per tenere una traccia storica delle richieste di lavoro effettuate, sia quelle programmate che quelle straordinarie. Questo è l'archivio più importante in quanto fa riferimento a quasi tutti gli altri archivi.

Senza entrare troppo nel dettaglio, questo archivio contiene: un numero progressivo di richiesta di lavoro, la struttura su cui si compie la manutenzione, il tipo di manutenzione (ispettiva, programmata o urgente, ma anche interna o esterna), il fornitore che compie la manutenzione e il costo, opportunamente ripartito.

◆ **Anagrafica strutture**

Questo archivio contiene tutti i soggetti da manutenere, quindi le macchine in officina, l'impianto idraulico, l'impianto elettrico, i calcolatori, le stampanti ecc...

Questo archivio contiene tutte le informazioni relative alla struttura, ossia: il codice della struttura, la data di acquisto o produzione, le ore di funzionamento giornaliero, il reparto ecc... Da notare che vengono calcolate automaticamente le ore di funzionamento per l'intero anno grazie all'utilizzo di un calendario.

◆ **Ricambi o anagrafica parti**

Questo archivio contiene tutti i pezzi di ricambio che servono per mantenere funzionanti le strutture. Le informazioni contenute in questa anagrafica sono: il codice del ricambio, la sua ubicazione nel magazzino e i fornitori che hanno rifornito recentemente l'azienda di quei materiali. Da notare che vengono memorizzati solo gli ultimi tre fornitori che hanno avuto a che fare con l'azienda; un eventuale quarto fornitore porta all'eliminazione del fornitore più vecchio.

◆ **Anagrafica fornitori**

Questo archivio contiene tutti i fornitori dell'azienda che sono coinvolti dal discorso delle manutenzioni. Questo include fornitori di ricambi, fornitori di strutture e fornitori di contratti di manutenzione o di garanzia. Questo archivio contiene: il fornitore e i suoi dati anagrafici, compreso il nome della persona di riferimento all'interno dell'azienda fornitrice.

◆ **Calendari**

Questo è un archivio che comprende vari calendari, con una specifica informazione per ogni giorno dell'anno. Esiste un calendario principale (chiamato "standard") a cui fanno riferimento tutte le strutture cosiddette "normali" e una serie di calendari particolari, ognuno dei quali è associato a una specifica struttura.

◆ **Distinta base**

Viene gestita anche una piccola distinta base tra le strutture e i ricambi. In questo modo l'addetto alle manutenzioni può sapere facilmente le relazioni tra le strutture e i ricambi, quindi quali ricambi servono per una data struttura e quali strutture hanno bisogno di un dato ricambio. Comunque cos'è precisamente una distinta base sarà approfondito più avanti.

3.1.3 Operazioni sugli archivi

Vediamo brevemente quali sono le operazioni che bisogna implementare sui suddetti archivi; nel prossimo paragrafo si metteranno in evidenza le problematiche relative all'implementazione e al costo di queste operazioni.

■ **Anagrafica RDL o richieste di lavoro**

immissione di una richiesta di lavoro
aggiornamento di una richiesta di lavoro
cancellazione di una richiesta di lavoro
visualizzazione di tutto lo scadenzario RDL
interrogazione per codice struttura
interrogazione per reparto
interrogazione per intervallo di data (data reale di manutenzione)

tutte queste interrogazioni solo per manutenzioni ispettive, programmate o urgenti oppure solo per quelle interne o esterne.

stessi output sulla stampante

■ **Anagrafica strutture**

immissione di una struttura
aggiornamento di una struttura
cancellazione di una struttura
visualizzazione a video dell'intero archivio
interrogazione per intervallo di data
stampa

è inclusa anche l'operazione di associazione di un calendario e la sua lettura.

■ **Ricambi o anagrafica parti**

immissione di un ricambio
aggiornamento di un ricambio
cancellazione di un ricambio
visualizzazione a video dell'intero archivio
stampa

■ **Anagrafica fornitori**

immissione di un fornitore
aggiornamento di un fornitore
cancellazione di un fornitore
visualizzazione a video dell'intero archivio
stampa

■ **Calendario**

creazione calendario
lettura calendario

scrittura di un calendario
copia di un calendario
stampa

■ **Distinta base**

immissione legame di distinta base
cancellazione legame
esplosione ad un livello
esplosione riepilogativa
implosione ad un livello
stampa

sono incluse le operazioni di ricerca sulle strutture e sui ricambi.

3.1.4 Fasi dell'analisi

Affronteremo ora il nucleo dell'analisi, basata sul passaggio dei dati e delle operazioni, definite, per cosi' dire, alla rinfusa, come nel capitolo precedente; all'organizzazione ordinata e sistematica di questi dati per ottenere il pacchetto in forma definitiva. Questa analisi si suddivide in **3** fasi, con i seguenti obiettivi:

☞ **fase 1** : determinazione del modello concettuale;

☞ **fase 2** : determinazione del modello del carico;

☞ **fase 3** : determinazione del modello logico;

3.1.5 Analisi Bottom-up e modello concettuale (fase 1)

Come ho già avuto occasione e modo di spiegare nei capitoli precedenti, dopo aver avuto un lungo colloquio con l'addetto alle manutenzioni all'interno dell'azienda; sono subito emerse le operazioni necessarie sugli archivi. Per questo motivo si è scelta una metodologia di analisi di tipo *bottom-up* invece di una metodologia *top-down* che avrebbe richiesto una visione generale delle problematiche.

Una prima parte dell'analisi è rivolta alla stesura del modello concettuale, così come prevedono i progetti di basi di dati. Il modello concettuale (o modello ER) ci permetterà di avere una visione generale dei dati e delle relazioni tra i dati stessi. Inoltre permetterà di scegliere opportunamente l'implementazione delle relazioni, ottimizzando i tempi e i costi di ricerca, eliminando ogni tipo di ridondanza e permetterà la generalizzazione dei dati, ossia il loro raggruppamento in gerarchie ben precise.

❑ Terminologia e simbologia dei diagrammi entità-associazioni

Spieghiamo brevemente cos'è un modello concettuale e qual'è la simbologia che abitualmente si usa in questa sede.

Il modello concettuale si pone come obiettivo la rappresentazione della realtà, indipendentemente dalla possibile implementazione della stessa realtà su di un calcolatore o su un supporto cartaceo. Quindi un qualsiasi progettista che affronta la fase di modellazione dei dati, che è appunto questa fase, deve, prima di tutto, attribuire una struttura ai dati e cercare le relazioni che intercorrono tra i dati stessi.

In questa fase si parla di *entità, attributi, chiavi, associazioni tra entità e generalizzazioni (o gerarchie ISA).*

Una "entità" si può definire come qualsiasi cosa, concreta o astratta, distinguibile dalle altre. È compito del progettista concettuale classificare tutte le entità, della realtà in esame, in base alle similitudini che esse presentano.

Nella nostra realtà, entità possono essere ad esempio i fornitori , e viene spontaneo raggrupparli insieme. Questo processo mentale, volto ad individuare le somiglianze di diverse entità, astraendo dalle loro specifiche differenze, è noto come 'classificazione'. Gli insiemi di entità si dicono tipi di entità, cosi' ad esempio il fornitore Rossi è una entità di tipo 'fornitori'.

Di ogni tipo di entità il progettista deve individuare le proprietà (dette attributi) interessanti per chi osserva la realtà, ad esempio, sempre per i fornitori, possibili attributi sono: ragione sociale, indirizzo, telefono ecc...

Un insieme di attributi, che consente di distinguere un'entità dall'altra, viene detto 'chiave'. Nel caso del tipo di entità fornitori, la chiave potrebbe essere *{nome, cognome, data_di_nascita, luogo_di_nascita}*; infatti questi dati possono bastare per individuare univocamente un'entità.

Un discorso più lungo meritano le associazioni tra le entità;

> un'associazione **A** su n insiemi di entità $\mathbf{S_1,S_2,S_3,...,S_n}$ è una relazione matematica sugli stessi insiemi, vale a dire un sottoinsieme del prodotto cartesiano **S1 x S2 x S3 x ... x S4**.

Questo significa, ad esempio, che un fornitore può essere associato a *n* ricambi, come un ricambio può essere associato a *n* fornitori; se cosi fosse, si

otterrebbe un insieme di relazioni pari al prodotto cartesiano tra i fornitori e i ricambi; in realtà non esistono tutte queste relazioni, in quanto, ad esempio, un ricambio può essere fornito solamente da un fornitore, cosi una associazione è un sottoinsieme del prodotto cartesiano fra più insiemi.
Si potrebbe pensare alle associazioni come entità, con attributi delle varie entità a cui esse fanno riferimento; le associazioni e le entità presentano però una differenza concettuale che deve essere di guida al progettista: una associazione esiste in dipendenza di alcune entità, mentre le entità possono avere un'esistenza indipendente.

Ad una associazione **A** fra **X** e **Y** si attribuisce in genere un verso, diretto da X a Y (*associazione diretta*). Si conviene allora di chiamare inversa di A l'associazione A^{-1} fra Y e X ottenuta da A, semplicemente considerando le coppie nell'ordine inverso.

Un'associazione è di tipo **1:1** (*uno a uno*) se la diretta o l'inversa sono univoche. Un tipico esempio di associazione 1:1 è quella che intercorre fra presidi e scuole.
Un'associazione è di tipo **1:N** o **N:1** (*uno a molti*) se è multipla la diretta o l'inversa (ma non entrambe). Una associazione 1:N è quella che intercorre fra madri e figli.
Un'associazione è di tipo **N:M** (*molti a molti*) se sono multiple sia la diretta che l'inversa. Un'associazione N:M è quella che intercorre in genere fra libri e lettori.

Nel mio modello ho usato una simbologia leggermente diversa, infatti le associazioni hanno due coppie di valori, una per un verso e una per l'altro. Una coppia di valori indica, rispettivamente, il minimo e il massimo numero di entità a cui ci si può riferire.

Un ultimo elemento che riguarda la fase di modellazione dei dati è la 'generalizzazione'. Vi sono alcune situazioni reali in cui è possibile riconoscere e/o stabilire che un certo insieme **X** di entità, costituisce un sottoinsieme di una classe di entità **Y** più ampia che li comprende. Le entità di tipo X sono cioè dotate di tutte gli attributi posseduti dalle entità di tipo Y, ma ne possono avere degli altri che le contraddistinguono.
Ad esempio, con riferimento ad una realtà scolastica, i docenti sono una specializzazione dei dipendenti: un docente fa parte del personale e come tale interessa i settori amministrativi addetti a paghe, carriera ecc... Quando si vuole attestare una situazione di questo genere, si stabilisce una associazione particolare, indicata comunemente con il termine **ISA**; strutture di questo tipo sono note come **gerarchie ISA**.

I tipi di entità e le associazioni, individuate dal progettista, possono essere convenientemente rappresentate in forma grafica, mediante i cosiddetti diagrammi entità-associazioni. Per quanto ci riguarda, faremo riferimento alle seguenti convenzioni (vedi figure 3.1 e 3.2):

1) I tipi di entità sono rappresentati con il loro nome racchiuso dentro un rettangolo.

2) Gli attributi delle entità sono rappresentati come barre, terminanti con un cerchio, con accanto il nome dell'attributo. Queste barre saranno collegate al riquadro che indica l'entità a cui si riferiscono.

Errore. L'origine riferimento non è stata trovata.

Figura 3.1 Rappresentazioni di Entità (a) e Associazioni (b)

3) Gli attributi che rappresentano le chiavi sono normali attributi con la barretta in neretto.

4) Un'associazione di nome A tra X e Y viene rappresentata mediante il nome A racchiuso dentro un rombo, collegato con un arco ad X e con un altro arco a Y.

5) Il tipo di associazione viene rappresentato mettendo le diciture 1 e N sugli archi che collegano la associazione alle entità.

6) Le gerarchie ISA si rappresentano come in figura 3.2.

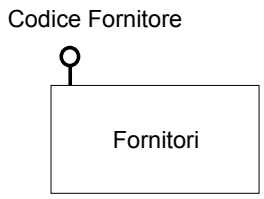

Figura 3.3

Modello ER dei Fornitori

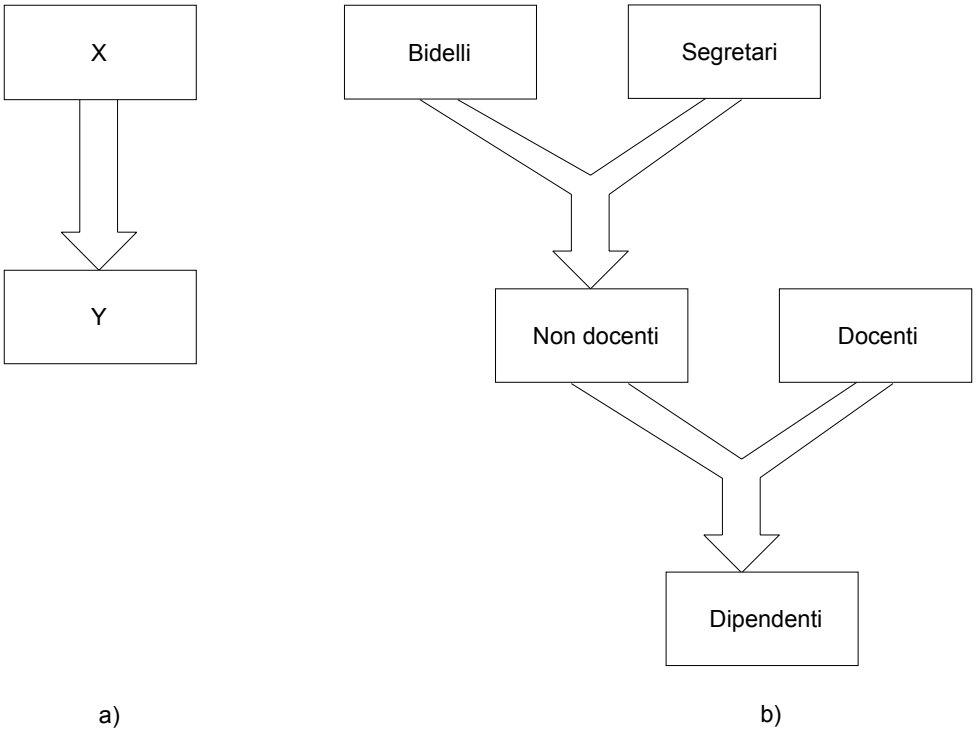

Figura 3.2 Rappresentazione di X ISA Y (a) e di una gerarchia ISA (b)

Iniziamo ora l'analisi bottom-up partendo dai modelli ER delle varie operazioni, ci renderemo conto così di quali entità e associazioni ogni operazione coinvolge.

Nei seguenti diagrammi entità-associazioni non saranno inclusi tutti gli attributi, per motivi di spazio e di ordine visivo, quindi mi sono limitato alle chiavi, ma si possono vedere gli elenchi degli attributi nella documentazione successiva.

❑ Operazioni sui fornitori

L'anagrafica fornitori è sicuramente l'archivio più isolato fra tutti quelli del pacchetto, questo significa che non interferisce direttamente con gli altri, o meglio, ogni sua consultazione non provoca accessi ad altre anagrafiche.

Al contrario, quasi tutti gli altri archivi fanno riferimento ai fornitori, e quindi, questi ultimi, sono collegati ad una serie di associazioni in modo, per cosi' dire, 'passivo'. Per le varie operazioni di immissione, aggiornamento, cancellazione, visualizzazione o stampa, il semplice modello ER è quello della figura 3.3.

❑ Operazioni sui ricambi

Anche questa anagrafica, come i fornitori, non coinvolge molti dati memorizzati in altri archivi, deve pero' contenere un riferimento ai fornitori che possono fornire l'azienda di un specifico ricambio. Per questo, esiste un'associazione tra ricambi e fornitori, la sua cardinalità si può notare essere da uno a tre; infatti all'azienda non importa tenere una lista illimitata di fornitori, vengono tenuti solo gli ultimi tre fornitori che hanno fornito all'azienda quello specifico ricambio.

Il modello ER per le operazioni di immissione, aggiornamento, cancellazione, visualizzazione o stampa è il seguente:

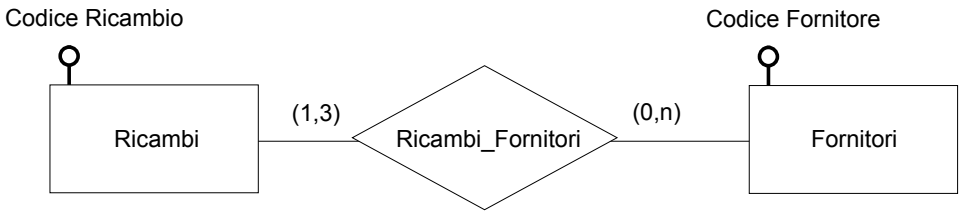

Figura 3.4 Modello concettuale dei Ricambi

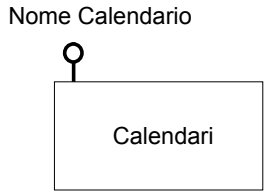

Figura 3.5 Modello ER dei Calendari

❑ Operazioni sui calendari

I calendari sono una raccolta di informazioni per ogni giorno dell'anno, relativo ad una data struttura. Questo significa che dentro questa anagrafica ci saranno memorizzati i giorni lavorativi, i giorni festivi ecc.. Per questo, l'unica associazione che riguarda i calendari è quella con le strutture ad essi relativi.

Le operazioni di aggiornamento e di copia di altri calendari non riguardano le strutture e quindi hanno il semplice modello della figura 3.5; per le operazioni di immissione e cancellazione, invece, il modello ER è il seguente:

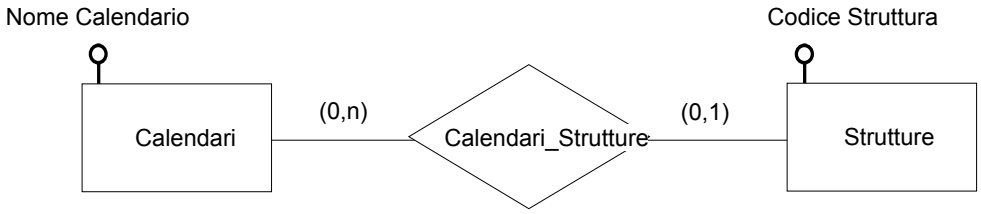

Figura 3.6 Modello concettuale dei Calendari per le operazioni di immissione e cancellazione

❑ Operazioni sulle strutture

Le strutture sono molto complesse da gestire, una struttura ha molti riferimenti ad altri dati:

Errore. L'origine riferimento non è stata trovata.Errore. Il segnalibro non è definito.prima di tutto bisogna conoscere il fornitore, ammesso che

esista, in quanto la struttura può anche essere prodotta internamente, ecco perché la cardinalità della associazione strutture-fornitori in figura 3.7 è (0,1);

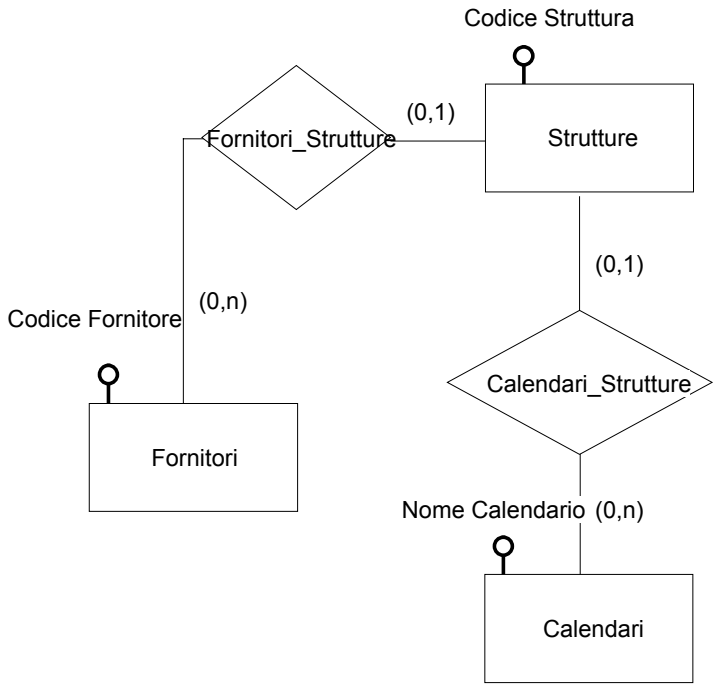

Figura 3.7 Modello concettuale delle strutture

Errore. L'origine riferimento non è stata trovata.Errore. Il segnalibro non è definito.per ogni struttura bisogna conoscere quante sono le ore lavorative annue, per fare questo bisogna che l'utente inserisca quante ore la struttura lavora nei giorni feriali, quante ore nei sabati lavorativi e che il programma calcoli il tutto automaticamente (facendo riferimento ad un calendario);

infine mi è stata richiesta l'implementazione di una serie di note tecniche sulla struttura, quindi l'inclusione di un testo libero di lunghezza variabile;

Il modello ER per le operazioni di immissione, aggiornamento, cancellazione, visualizzazione o stampa è quello della figura 3.7.

❑ Operazioni sulla distinta base

La distinta base è un'anagrafica che mette in relazione le strutture e i ricambi, quindi viene naturale esprimere questa relazione attraverso un'associazione, il modello ER è quindi il seguente:

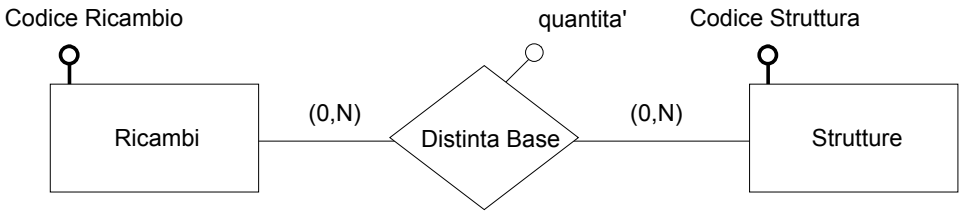

Figura 3.8 Modello concettuale della Distinta Base

❑ Operazioni sulle richieste di lavoro

Le richieste di lavoro, al contrario dei fornitori, fanno riferimento ad una grossa mole di dati esterni. Il modello ER è il seguente (figura 3.9).

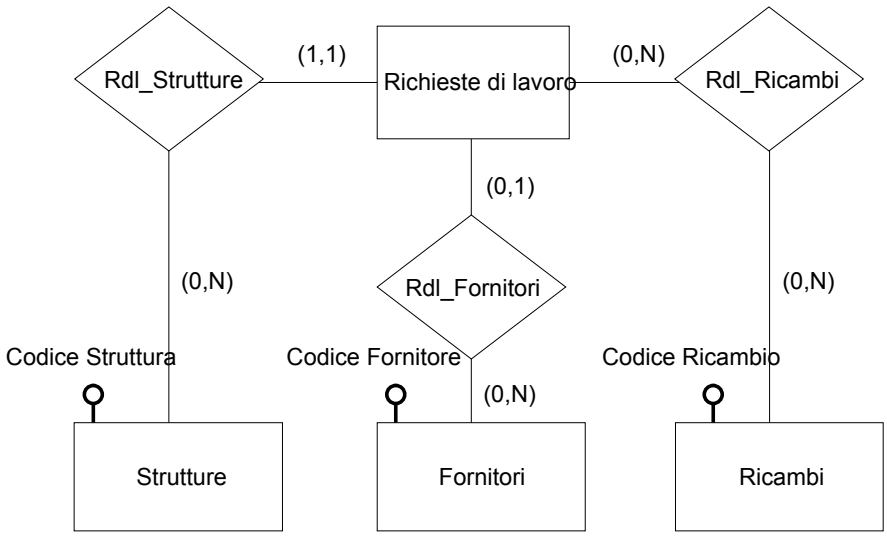

Figura 3.9 Modello concettuale delle Richieste di Lavoro

- abbiamo il solito riferimento al fornitore che compie la manutenzione, che può anche non esistere come già fatto osservare in precedenza, in quanto l'intervento può essere eseguito internamente;
- una lista di ricambi che servono per compiere la manutenzione, non è previsto un numero massimo di ricambi quindi la cardinalità è (0,n);
- la struttura sulla quale si compie l'intervento; anche qui è richiesta l'implementazione di un testo libero, di lunghezza variabile, sotto il nome di 'note tecniche sull'intervento;

❑ Diagramma entità-associazioni

Ora il passo più importante ! riunire tutte le varie entità e associazioni in un unico modello. Il modello della figura 3.10 rispecchia fedelmente la realtà ed è indipendente dall'implementazione e dalle specifiche che saranno utilizzate successivamente, quindi è la base di partenza per lo sviluppo del pacchetto.

Il seguente modello è già in forma *leggibile, corretta e minimale*; prima della sua stesura pero' è stato fatto un grosso lavoro di ristrutturazione e di organizzazione del modello stesso. La prima forma del modello era piuttosto caotica, soprattutto perché i molteplici collegamenti, tra entità e associazioni, erano stati disposti casualmente. Successivamente il modello è stato ristrutturato, cercando di renderlo ordinato, ma soprattutto leggibile e interpretabile da chiunque.

Dopo la suddetta operazione, sono state eliminate alcune ridondanze, ossia dei dati ripetuti inutilmente, senza alcun svantaggio in termini di costi. In questo modo si è ottenuto un modello ridotto ai minimi termini (anche se a vederlo non sembrerebbe), ma pur sempre specchio fedele della realtà.

3.1.6 Modello di carico (fase 2)

Adesso inizia la seconda fase, che ha come obiettivo la determinazione di un modello di carico.

Nella prossima fase, il compito principale del progettista è quello di trasformare il modello concettuale in uno schema adeguato al linguaggio che verrà utilizzato; in sostanza uno schema che rispecchi l'implementazione che si darà della realtà.

Per fare questo bisogna prima valutare ogni possibile implementazione, compatibilmente con quello che offre il sistema operativo, l'ambiente di lavoro e il linguaggio di programmazione.

Il progettista deve quindi individuare degli indici di prestazione, cioè dei parametri, con i quali si valuta la bontà di un'implementazione rispetto ad un'altra; esempi di indici di prestazione sono: il tempo di risposta e l'occupazione di memoria.

Come può il progettista valutare quali sono le implementazioni ottimali? Io ho compilato un modello dei dati, ossia uno schema contenente:

- la frequenza d'uso dei dati o di alcune operazioni, in genere giornaliera;
- la popolazione, ossia per ogni entità e associazione il numero di istanze, ad esempio il numero di fornitori.

Una volta redatto il modello dei dati, posso scegliere implementazioni diverse, privilegiando il tempo di risposta o l'occupazione di memoria, a seconda delle frequenze d'uso.

Possiamo vedere un esempio, che ci aiuta a capire meglio questa importante valutazione; un'operazione, che viene richiamata 50 volte al giorno, come

può essere quella della ricerca di una richiesta di lavoro, non può avere un tempo di risposta di 5 minuti, altrimenti l'addetto alle manutenzioni sprecherebbe più di 4 ore (50x5=250 minuti) al giorno, del suo lavoro, inutilmente. In questo caso è più conveniente sprecare molta memoria e diminuire al massimo il tempo di risposta; in questo modo, l'addetto alle manutenzioni potrà usufruire di questo servizio quando vuole, senza preoccuparsi di attendere molto.

Il progettista, in questo caso, deve pensare a sviluppare un algoritmo più complicato, ma con una complessità computazionale minore. Al contrario, una operazione che viene eseguita una volta all'anno, può anche essere implementata con un algoritmo semplice, che non fa spendere molto in fase di progettazione, anche se non è molto efficiente.

Nella redazione del mio modello dei dati ho valutato soltanto due indici di prestazione: *la frequenza d'uso dei dati* per ogni operazione e *la quantità di istanze coinvolte*.

Di seguito presentiamo la tabella delle istanze coinvolte:

Richieste_di_lavoro	210 (al mese)
Fornitori	70
Ricambi	4000
Strutture	400
Calendari	161 (160+1)
Rdl_Strutture	1,9 (400:210)
Rdl_Ricambi	0,19 (1,9:10)
Rdl_Fornitori	57 (30x1,9)
Fornitori_Strutture	30
Ricambi_Fornitori	300 (30x10)
Strutture_Calendari	0,4 (161:400)
Ricambi_Strutture	10

Di seguito presentiamo la tabella delle frequenze d'uso (previste) delle operazioni più significative:

1	Input Richieste di lavoro	5 al giorno
2	Aggiornamento Rdl	2 al giorno
3	Ricerca rdl per cod. struttura	7 al giorno
4	Ricerca rdl per reparto	1 alla settimana
5	Ricerca rdl per int. di data	2 al giorno
6	Aggiornamento Struttura	1 ogni due mesi
7	Aggiornamento Ricambio	2 al mese
8	Aggiornamento Fornitore	1 al mese
9	Consultazione Calendario	1 alla settimana
10	Implosione	2 al mese
11	Esplosione scalare	7 al giorno
12	Esplosione riepilogativa	1 al giorno

Ora presentiamo, per ogni operazione precedente, quanti accessi agli archivi e alle associazioni essa compie:

operazione->	1	2	3	4	5	6	7	8	9	10	11	12
entità assoc.												
Rdl	1825	730	2555	52	730							
Fornitori								12				
Ricambi							24				25550	36500
Strutture						6				48		
Calendari									624			
Rdl_Str.	1825	730	2555	52	730							
Rdl_Ric.	7300	2920	10220	208	2920							
Rdl_Forn.	1825	730	2555	52	730							
Forn_Strutt.						6						
Ric_Forn.							72					
Strutt_Cal.						6						
Ric_Strutt.										48	25550	36500
Frequenze	5	2	7	1	2	6	2	1	1	2	7	1
Riferito												
all'anno	1825	730	2555	52	730	6	24	12	52	24	2555	365

❍ **Osservazioni**

Innanzitutto si può notare che il modulo più utilizzato è quello delle richieste di lavoro, ma questo si era già capito da tempo; la ricerca delle richieste di lavoro per codice struttura è l'operazione fondamentale, in quanto il responsabile delle manutenzioni richiederà la situazione storica della struttura ogni volta che bisogna intervenire (come da noi previsto, circa 7 volte al giorno).

Le altre operazioni di maggiore utilizzo sono quelle relative alla distinta base, in particolare l'esplosione scalare, che serve per conoscere di quanti e quali ricambi una determinata struttura necessita. Per queste operazioni si avrà un occhio particolare durante la fase di implementazione.

Lo schema precedente, in realtà, è frutto di una serie di ottimizzazioni raggiunte per passaggi successivi. Osservando il primo modello dei dati, si è cercato di ottimizzare l'utilizzo di entità e associazioni, ottenendo cosi' successivi modelli, più semplici e ridotti ai minimi termini.

3.1.7 Progetto logico (fase 3)

La terza fase, che ci apprestiamo ad analizzare, riguarda l'effettiva traduzione del modello concettuale in un modello logico, cosi' come è stato definito in precedenza. Ora che siamo a conoscenza di quanto ogni operazione incide sulla complessità computazionale del programma, possiamo scegliere le implementazioni più efficaci e soprattutto più efficienti.

Per creare il modello logico mi sono rifatto ai modelli relazionali di E.F. Codd, in genere questi modelli si usano nei data base management system (ad esempio *Informix*, *DbIII* o *Oracle*). Io ho deciso di adottare questi schemi, anche se il mio programma sarà fatto in 'C', per due motivi:

- rappresentano in modo semplice e intuitivo le relazioni tra i dati;
- usando PXEngine, il mio programma usa delle tabelle di Paradox, quindi, indirettamente, io utilizzo un data base management system, Paradox appunto.

❍ Terminologia, simbologia e regole dei modelli relazionali

Introduciamo brevemente, come è stato fatto con i diagrammi entità-associazioni, i concetti e la simbologia classica nei modelli relazionali. Il modello relazionale si basa sul fondamentale concetto matematico di relazione:

Una relazione **R**, su una sequenza di insiemi $\mathbf{D_1,D_2,...,D_n}$ (non necessariamente distinti), è un sottoinsieme finito del prodotto cartesiano $\mathbf{D_1 \times D_2 \times ... \times D_n}$: R incluso in senso largo in $\mathbf{D_1 x D_2 x ... x D_n}$.

Il numero **n** (n>=1) è detto grado della relazione. Gli insiemi $D_1,D_2,...,D_n$ sono detti domini della relazione. Ad ogni dominio è associato un nome, detto attributo. Gli elementi di R sono delle ennuple (d1,d2,...,dn), viene però spesso usato il termine 'tupla'. Il numero **m** (m>=0) di tuple presenti, in un dato istante, in una relazione R, viene detto cardinalità della relazione.

Da un punto di vista intensionale, una relazione è caratterizzata da un nome e dalla lista dei suoi attributi (schema di relazione). Nel seguito adotteremo la seguente sintassi:

<nome_relazione> ($<attr_1>$:$tipo_1$,$<attr_2>$:$tipo_2$, ... ,$<attr_n>$:$tipo_n$)

dove $tipo_1$,$tipo_2$,...,$tipo_n$ sono i tipi degli attributi.

Da notare che gli attributi chiave saranno sottolineati, mentre le chiavi esterne, di cui parleremo dopo, saranno in neretto.

A questo punto qualche lettore potrebbe trovarsi un po' spaesato, facciamo quindi un esempio:

una relazione, come può essere 'persone', può avere come attributi: nome, età e sesso. La relazione persone nel modello logico sarà perciò rappresentata cosi:

persone(nome:alfanumerico[lunghezza],eta':numerico,sesso:carattere)

Molto più interessante è capire come si rappresentano le associazioni tra i dati nel modello relazionale. Infatti ogni tipo di entità, del modello concettuale, diventa una relazione nel modello relazionale, ma cosa succede a una associazione? Vediamo come viene trasformato ogni singolo elemento del modello concettuale nel modello relazionale:

- Un tipo di entità **E** con attributi elementari $\mathbf{A_1,A_2,...,A_n}$ trova una immediata rappresentazione mediante una relazione:

R(A1,A2,...,An)

- Un tipo di entità **E** con attributi composti e/o multipli (ad esempio dei gruppi ripetitivi) non può essere trasformata in una relazione, perché essa

non risulterebbe in forma normale; in questo caso si può procedere in due modi:

❶ per gli attributi composti e per quelli multipli con un numero massimo di elementi (come una tabella), si effettua una scomposizione; cosi' se ho un attributo indirizzo composto da via e numero civico, non inserisco l'attributo indirizzo, ma i due attributi: via e numero civico;

❷ per gli attributi multipli con un numero variabile di elementi, la scomposizione non può essere effettuata, gli elementi potrebbero essere virtualmente infiniti. In questo caso si spezza la relazione originale in due relazioni ad essa equivalenti (cioè aventi, complessivamente, lo stesso contenuto di informazione), vedi figura 3.11. Nella relazione non normalizzata FARMACI di figura 3.11a, l'attributo *Indicazioni* è multiplo: il numero di malattie che un farmaco può curare è variabile a seconda del farmaco (e comunque non limitato).

Errore. L'origine riferimento non è stata trovata.Errore. L'origine riferimento non è stata trovata.

Figura 3.11 Relazione con attributo multiplo (a) e sua normalizzazione (b)

Un'associazione 1:N, che nel modello concettuale è rappresentata come (0,1)-(0,N) o (0,N)-(0,1), non si trasforma in una relazione. Infatti la creazione di una terza relazione, oltre alle due già coinvolte comporta un grosso spreco; il tutto si può risolvere facendo in modo che una relazione abbia un riferimento all'altra. La soluzione classica prevede di introdurre due relazioni cosi' concepite:

1) una relazione X con i suoi attributi

2) una relazione Y con i propri attributi più gli attributi chiave della prima relazione X.

Gli attributi chiave di X presenti in Y costituiscono una cosiddetta chiave esterna, cioè una chiave appartenente ad un'altra relazione (appunto esterna).

Un'associazione N:M, che nel modello concettuale è rappresentata come (0,N)-(0,N), diventa una relazione nel modello relazionale. Infatti in questo caso non è possibile utilizzare il precedente meccanismo della chiave esterna, in quanto servirebbero per ogni tupla n chiavi esterne.

Allora si crea una relazione avente gli attributi chiave della relazione X e gli attributi chiave della relazione Y (minimo due attributi), vedi figura 3.12. L'associazione molti a molti, esistente fra ARTICOLI e FORNITORI, in quanto un fornitore può fornire più articoli ed inoltre un articolo può essere fornito da più fornitori; può essere decomposta nello schema relazionale seguente:

Errore. L'origine riferimento non è stata trovata.Errore. L'origine riferimento non è stata trovata.

Figura 3.12 Associazione molti a molti nel modello relazionale

Modello relazionale

Adesso che conosciamo le regole dei modelli relazionali, cerchiamo di costruire il modello logico (o modello relazionale) del pacchetto.

Rich.di lavoro	(	Numero_progressivo:numerico, **Codice_struttura**:alfanumerico[16], Tipo_manutenzione:carattere, Data_prevista_esecuzione:data, Data_reale_di_esecuzione:data, Ore_previste:numerico, **Puntatore_ricambi_previsti**:numerico, Stato_richiesta_di_lavoro:carattere, Flag_interno_esterno:carattere, **Codice_fornitore**:alfanumerico[6], Costo_totale_fatturato_da_fornitore:numerico, Costo_fatturato_da_fornitore_manodopera:numerico, Costo_fatturato_da_fornitore_per_ricambi:numerico, Ore_consuntivate_manutenzione_interna:numerico, Ore_fermo_macchina:numerico, Reparto:alfanumerico[2], **Puntatore_ricambi_consuntivi**:numerico, **Puntatore_note**:numerico
	)	

Strutture	(	Codice_interno_manutenzione:alfanumerico[16], Codice_cespite_ced:alfanumerico[16], Centro_di_costo_ced:alfanumerico[5], Centro_di_costo_interno_manutenzione:alfanumerico[5], Costo_orario:numerico, Descrizione:alfanumerico[30], Flag_acquistato_prodotto:carattere, Data_acquisto_produzione:data, **Codice_fornitore**:alfanumerico[6], Ore_giorni_lavorativi:numerico, Ore_sabati_lavorativi:numerico, Ore_anno_funzionamento_previsto:numerico, Potenza_installata_kw:numerico, **Puntatore_note**:numerico, Ubicazione:alfanumerico[6], Reparto:alfanumerico[2], **Codice_forn_contratto_manutenzione**:alfanumerico[6], **Nome_calendario**:alfanumerico[20],
	)	

Ricambi	(	
		Codice_ricambio:alfanumerico[16], Descrizione_ricambio:alfanumerico[30], Note_tecniche:alfanumerico[30], Ubicazione:alfanumerico[6], **Codice_fornitore_ultimo**:alfanumerico[6], **Codice_ricambio_fornitore_ultimo**:alfanumerico[16],

		Costo_ultimo:numerico, Data_ultima:data, **Codice_fornitore_1**:alfanumerico[6], **Codice_ricambio_fornitore_1**:alfanumerico[16], Costo_1:numerico, Data_1:data, **Codice_fornitore_2**:alfanumerico[6], **Codice_ricambio_fornitore_2**:alfanumerico[16], Costo_2:numerico,
	)	

Fornitori	(	Codice_fornitore:alfanumerico[6], Ragione_sociale:alfanumerico[35], Indirizzo:alfanumerico[35], Tel:alfanumerico[12], Fax:alfanumerico[12], Persona_di_riferimento:alfanumerico[20],
	)	

Distinta Base	(	**Codice_struttura**:alfanumerico[16], **Codice_ricambio**:alfanumerico[16], Quantità:numerico,
	)	

Calendari	(	Nome_calendario:alfanumerico[20], Anno:numerico, Mese:numerico, Numero_giorni:numerico, Giorni:alfanumerico[31],
	)	

Note	(	Indice:numerico, Riga:alfanumerico[78],
	)	

Ric	(	Indice:numerico, **Codice_ricambio**:alfanumerico[16],
	)	

Figura 3.13 Modello relazionale dell'intero programma

❍ Osservazioni

Analizziamo attentamente il modello relazionale della figura 3.13, cerchiamo di esaminarlo e capire quali trasformazioni sono state fatte, osservando soprattutto le nostre vecchie associazioni cosa sono diventate.

Come si può vedere ogni entità del modello ER è diventata una relazione, cosi' è accaduto infatti per le richieste di lavoro, i fornitori, le strutture, i ricambi e i calendari.

Si sono aggiunti a queste entità degli attributi, che rappresentano delle chiavi esterne, vediamo in quali casi (associazioni 1:N) :

- esiste una associazione tra le strutture e i calendari, siccome ogni struttura può essere associata a un solo calendario, mentre un calendario può essere associato a più strutture; la soluzione più efficiente è quella di introdurre un riferimento (chiave esterna) nella relazione 'strutture' con il nome del calendario ad esso associato, questa infatti è la chiave che permette di individuare univocamente un calendario;
- la stessa cosa è avvenuta per il fornitore associato alla struttura, in realtà sono due le chiavi esterne: 'codice_fornitore' che rappresenta il fornitore della struttura e il 'codice_fornitore_contratto_di_manutenzione' che rappresenta il fornitore di contratti di assistenza;
- altre chiavi esterne sono presenti nella relazione 'ricambi', mi riferisco al codice del fornitore, anche qui sono presenti tre chiavi esterne, una per ogni fornitore del ricambio in tempi o costi diversi;
- due chiavi esterne sono presenti nella relazione 'richieste di lavoro': il codice della struttura e il codice del fornitore.

In altri casi (attributi multipli senza un numero massimo di elementi) non è stato possibile utilizzare chiavi esterne, è il caso delle due liste di ricambi (ricambi preventivi e consuntivi) presenti nelle richieste di lavoro.

In questo caso bisognerebbe memorizzare n chiavi esterne, il che, con gli strumenti software a mia disposizione, non è possibile; quindi è stata creata una nuova relazione ('*ric*'), essa contiene un numero intero che mi serve per individuare insiemi di tuple e il codice di un ricambio.
In questo modo all'interno della relazione 'richieste di lavoro' conservo solo un riferimento numerico alla relazione 'ric', quando voglio vedere la lista di ricambi cerco tutte le tuple in 'ric' con indice uguale a quello memorizzato nelle richieste di lavoro.
È stata utilizzata la stessa relazione ('ric') per conservare tutte e due le liste di ricambi presenti in 'richieste di lavoro'. Lo stesso meccanismo è stato utilizzato per implementare il testo libero delle note, sia per le strutture che per le richieste di lavoro. Infatti le note sono memorizzate per righe (di una lunghezza massima fissata a priori), nelle relazioni 'strutture' e 'richieste di lavoro' è stato posto solamente un puntatore, ossia un numero intero, indice della relazione 'note'. Quando si vogliono visualizzare le note, si leggono le righe con indice uguale a quello memorizzato nel puntatore.

Un discorso a parte merita l'associazione 'distinta base' che rappresenta una associazione N:M. In questo caso l'associazione è stata trasformata in una relazione, la nuova relazione contiene due chiavi esterne, una per riferirsi alle strutture e un'altra per riferirsi ai ricambi, in questo modo si può implementare l'associazione molti a molti, propria della distinta base.
Inoltre, ogni coppia di questa relazione, che rappresenta un legame di distinta base, ha bisogno di una informazione aggiuntiva, ossia la quantità, cosi' questo attributo è stato aggiunto alla relazione 'distinta base'.

3.1.8 Scelta degli strumenti e dei modi di accesso agli archivi

Questo è il momento di scegliere gli strumenti e i metodi di accesso agli archivi, cosi' come definiti in precedenza. Questa fase, in genere, risulta piuttosto delicata, la scelta va ponderata in funzione delle frequenze di accesso ai dati.

Nel mio caso la scelta non ha riguardato tanto lo strumento di accesso agli archivi, infatti ho utilizzato gli indici che mette a disposizione paradox engine; quanto piuttosto il numero di indici e la modalità di utilizzo degli stessi.

Io, comunque, ho limitato la scelta degli indici solo sulle operazioni di ricerca effettivamente implementate. Vediamo ora quali indici ho deciso di creare e di mantenere aggiornati, durante l'utilizzo del pacchetto; questo per ogni relazione del modello relazionale.

- L'anagrafica fornitori ha solo un indice di chiave primaria sul codice fornitore.
- L'anagrafica ricambi ha solo un indice di chiave primaria sul codice ricambio.
- L'anagrafica strutture ha l'indice di chiave primaria sul codice struttura, ma anche un indice di chiave secondaria sulla data di acquisto o produzione.
- L'anagrafica calendari ha solo un indice di chiave secondaria sul nome del calendario.
- L'anagrafica richieste di lavoro ha un indice di chiave primaria sul numero progressivo di richiesta, inoltre, essa possiede tre indici di

chiave secondaria: uno sul codice della struttura a cui si riferisce la richiesta di lavoro, uno sul reparto e uno sulla data reale di svolgimento del lavoro.

- L'archivio che gestisce la distinta base ha tre indici: uno di chiave primaria sulla coppia di campi codice padre e codice figlio, uno di chiave secondaria solo sul codice padre e uno solo sul codice figlio. Infine gli archivi note e ric hanno un indice di chiave secondaria sul campo "indice", o meglio sull'identificatore.

3.2 Modularizzazione del problema

Giunti a questo punto, nell'analisi del pacchetto, iniziamo a parlare delle procedure che devono gestire i dati; fino ad ora, infatti, abbiamo approfondito soprattutto l'aspetto che riguarda la definizione dei dati stessi.
Come si usa spesso per programmi di tipo gestionale, il pacchetto è stato suddiviso in moduli, nel nostro specifico caso sono **sei** moduli, che probabilmente, almeno in parte, si sono già intuiti, poiché rispecchiano la suddivisione, definita in precedenza, degli archivi.
Sono stati introdotti due livelli di modularizzazione (figura 3.14), un primo livello suddivide l'intero pacchetto in *aree operative* (o aree di lavoro), quindi una ripartizione che privilegia l'oggetto su cui si opera; un secondo livello di modularizzazione invece è introdotto all'interno di ogni modulo, questa suddivisione è, al contrario della precedente, di tipo funzionale, quindi si ripartiscono le istruzioni al calcolatore in base alla funzione che esse svolgono nel contesto del programma nel quale si trovano.

Errore. L'origine riferimento non è stata trovata.Errore. L'origine riferimento non è stata trovata.

Figura 3.14 Modularizzazione a più livelli del pacchetto

Elenchiamo i moduli di primo livello del pacchetto, cosi' come mostra la figura 3.14:

Fornitori
Ricambi
Strutture
Calendari
Distinta Base
Richieste di lavoro

Vedremo successivamente più in dettaglio, quando parleremo degli algoritmi di ogni modulo, i moduli di secondo livello; in questa sede, invece intendo introdurre prima le caratteristiche comuni a tutti i moduli e successivamente analizzare più in dettaglio i moduli di primo livello; iniziando da quelli più semplici.

3.2.1 Caratteristiche comuni a tutti i moduli

Tutti i moduli del pacchetto hanno una struttura molto simile, tipica dei programmi sotto windows; questo vale soprattutto per l'interfaccia utente, che segue, in linea di massima, lo standard **CUA**. Il "*Common User Access*" (CUA) è uno studio, portato avanti dall'IBM, che definisce i componenti di una interfaccia grafica a finestre e le modalità di uso degli elementi stessi; esso attualmente è uno standard riconosciuto a livello internazionale.
I miei moduli utilizzano un'interfaccia grafica che segue gli standard CUA, in questo modo l'utente non si trova spaesato di fronte al pacchetto, in quanto vede strumenti comuni a tutte le più diffuse "*Graphics User Interface*" (**GUI**).
Si può eseguire un modulo effettuando un'operazione di click, con il mouse, sull'apposita icona; appare così una finestra, di dimensione pari allo schermo attualmente in uso, essa contiene un'insieme di campi in cui é possibile inserire o aggiornare vari valori (vedi figura 3.15).
All'inizio tutti i campi sono disabilitati, quindi l'utente è costretto ad utilizzare il menù. Il menù contiene una serie di opzioni, classiche nei pacchetti di tipo gestionale, ad esempio:

Immissione
Aggiornamento
Cancellazione
Visualizzazione
Stampa
Informazioni
Aiuto

Le prime due opzioni abilitano la maschera, e contemporaneamente disabilitano il menù; l'utente può quindi inserire i dati di un nuovo record o aggiornare un record precedentemente ricercato, attraverso la chiave di accesso all'archivio. La visualizzazione apre una nuova finestra, in cui è possibile scorrere, avanti o indietro, tra i record dell'archivio, in ordine, chiaramente, di chiave primaria. Le ultime due danno rispettivamente informazioni sul pacchetto e una videata di aiuto rapido.

Tutti i moduli contengono queste opzioni di menù (tranne i calendari e la distinta base che sono un po' particolari, e saranno discussi in dettaglio successivamente) e sono strutturati attraverso un doppio stato: quello di utilizzo del menù e quello di utilizzo della maschera.

Vediamo ora tutti i moduli, senza entrare troppo nel merito dell'utilizzo del pacchetto, aspetto rimandato al manuale utente.

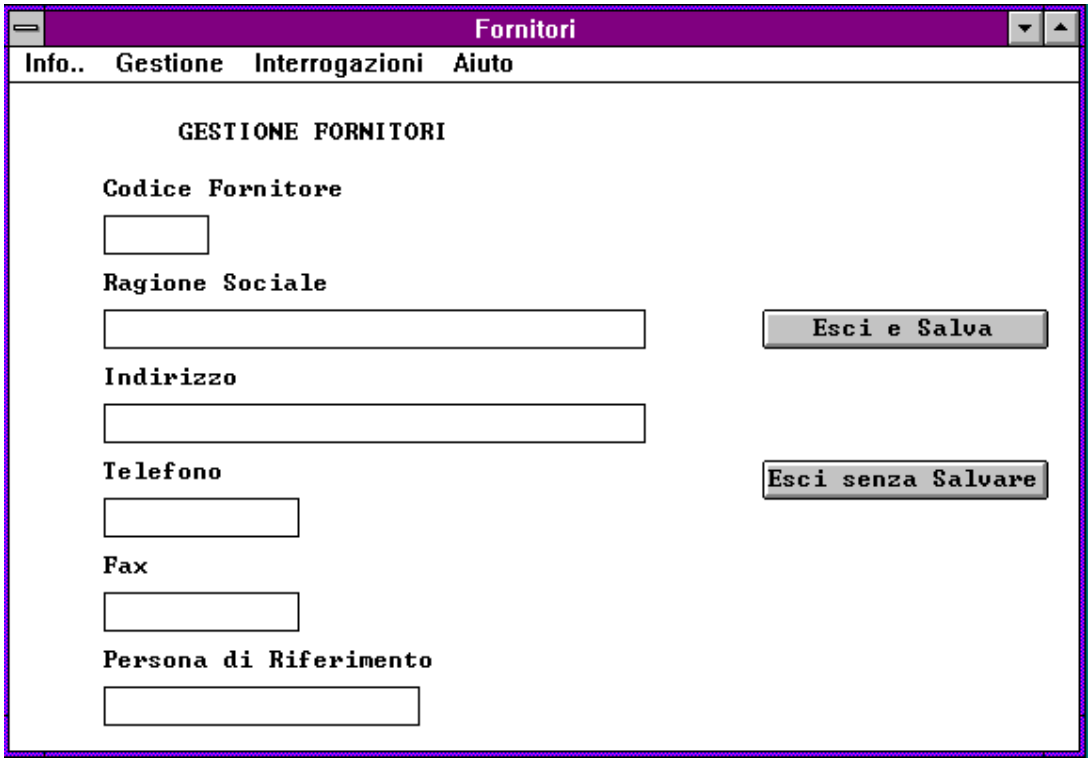

Figura 3.15 Videata che appare al momento dell'esecuzione del modulo FORNITORI

3.2.2 Modulo dei fornitori

Il modulo dei fornitori gestisce tutte le operazioni sulla anagrafica fornitori, questo significa che esso permette l'immissione, l'aggiornamento e la cancellazione di un fornitore; inoltre implementa le operazioni di visualizzazione a video e di stampa dell'intera anagrafica.

Non c'è molto da dire su questo modulo, in quanto assai semplice e rispecchia in pieno le caratteristiche comuni sopra citate. Esso è l'unico a gestire l'anagrafica Fornitori, quindi il responsabile delle manutenzioni, dopo aver inserito i fornitori di sua pertinenza, non utilizzerà molto spesso questo modulo. Solamente quando un nuovo fornitore entrerà in contatto con l'azienda, il responsabile delle manutenzioni inserirà un fornitore, oppure, cancellerà il medesimo, se questo ha esaurito il proprio rapporto con

l'azienda stessa. Inoltre, é indispensabile avere la possibilità di visualizzare o stampare tutta l'anagrafica, mentre non c'è assolutamente bisogno di interrogazioni di altro tipo.

L'anagrafica fornitori contiene queste informazioni:

Codice fornitore, codice che identifica in modo univoco un fornitore, non ha nulla a che fare con la contabilità industriale;

Ragione sociale;

Indirizzo;

Numero di telefono;

Numero di Fax;

Persona di riferimento all'interno dell'azienda fornitrice.

3.2.3 Modulo dei ricambi

Il modulo dei ricambi, come quello dei fornitori, non presenta particolari operazioni, se non quella dei tre fornitori. Infatti, come già accennato, l'azienda intende mantenere solamente gli ultimi tre fornitori che hanno fornito ricambi all'azienda; così in questo modulo, quando l'utente modifica i dati relativi a un fornitore, automaticamente si perdono le informazioni del più vecchio fornitore e gli altri vengono spostati di un posto per permettere l'entrata dl nuovo fornitore. Chiaramente i campi che visualizzano le informazioni dei due più vecchi fornitori, nella maschera a video, sono rappresentati da campi di solo output, mentre il fornitore ultimo può essere modificato, ed è proprio questa modifica che permette lo spostamento dei fornitori.

Per il resto, anche questo modulo presenta le operazioni di inserimento di un nuovo ricambio, di aggiornamento di un ricambio esistente e di cancellazione di un ricambio; inoltre è possibile visualizzare o stampare l'intera anagrafica.

L'anagrafica ricambi contiene le seguenti informazioni:

Codice ricambio, codice che identifica univocamente un ricambio, esso rispecchia la codifica standard dell'azienda, è infatti formattato nel modo seguente:

xx.xxxxx.xx.xxxx

dove i singoli campi di sottocodici sono: la classe (2 caratteri), la famiglia (5 caratteri), il raggruppamento (2 caratteri) e il progressivo (4 caratteri);

Descrizione ricambio;

Ubicazione del ricambio;

Ora ci sono tre gruppi di campi che si ripetono, sono le informazioni che riguardano i fornitori di ricambi, essi sono memorizzati dal più recente al più vecchio:

Codice fornitore ultimo;

Codice ricambio fornitore ultimo, il codice con cui il fornitore codifica, nel proprio sistema informativo, il ricambio stesso;

Costo ultimo, ossia il costo del ricambio presso quel fornitore;

Data ultima;

Codice fornitore 1;

Codice ricambio fornitore 1;

Costo 1;

Data 1;

Codice fornitore 2;

Codice ricambio fornitore 2;

Costo 2;

Data 2.

3.2.4 Modulo delle strutture

Questo modulo, insieme a quello delle richieste di lavoro, é uno dei più grandi come quantità di istruzioni al calcolatore ed uno dei più complessi da implementare.

Due novità sono state introdotte in questo modulo per quanto riguarda l'aspetto squisitamente estetico: l'implementazione di un flag di scelta attraverso due *radiobutton* (si veda il capitolo relativo alla traduzione dell'algoritmo in codice per maggiori dettagli sui radiobutton) e l'implementazione di un mini *editor* per le note.

Infatti, nella maschera delle strutture, appare una cornice con dentro due bottoni (radiobutton), effettuando un'operazione di click su uno di questi, si determina una scelta che, automaticamente, esclude l'altra possibilità. Nella figura 3.16 si possono vedere i due possibili stati dei bottoni, nell'archivio viene poi memorizzato un carattere per identificare la scelta, in questo specifico caso i caratteri sono **'a'** o **'p'**, che stanno appunto per struttura acquistata o prodotta.

Figura 3.16 Possibili stati del gruppo radio button

nella maschera del modulo strutture

Le note invece si possono vedere nella maschera, in quanto presenti tramite una finestra sul testo, in cui é possibile muoversi (mediante una scroll-bar) in senso verticale (vedi figura 3.17).

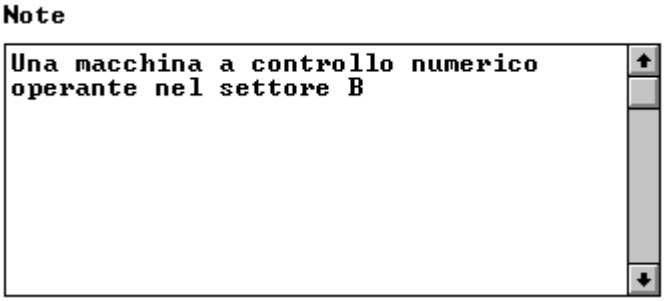

Figura 3.17 Esempio di editor per le note nella maschera del modulo Strutture

Infine, esiste un solo campo di output, rappresentato dalle ore lavorative della struttura, nell'arco di un intero anno; queste infatti risultano dal seguente calcolo:

ore totali di funzionamento previsto =
ore lavorative per giorno X giorni lavorativi previsti +
ore sabati lavorativi X sabati previsti

Le ore lavorative nei giorni feriali e nei sabati sono inserite dall'utente, mentre il numero dei giorni feriali e dei sabati di lavoro sono letti automaticamente dal calendario associato alla struttura.
In questo modulo è implementata la ricerca delle strutture per intervallo di data (di produzione o di acquisto), quindi scegliendo questa opzione, dal menù, è possibile inserire una data iniziale e una data finale ed ottenere un elenco delle strutture con data compresa fra le due specificate.

L'anagrafica strutture contiene le seguenti informazioni:

Codice interno manutenzione (oppure codice struttura), codice strutturato come specifica la codifica standard aziendale, il seguente é il formato:

xx.xxxxx.xx.xxxx

dove i singoli campi di sottocodici sono: la classe (2 caratteri), la famiglia (5 caratteri), il raggruppamento (2 caratteri) e il progressivo (4 caratteri).

Questo codice é interno al pacchetto stesso, a differenza del prossimo codice che equivale a quello della codifica aziendale.

Codice cespite, strutturato come il codice interno manutenzione, ma equivalente al codice della codifica aziendale, quindi svolge la funzione di strumento per associare la struttura alla contabilità industriale.

Centro di costo, codice che indica il centro di costo a cui imputare il costo stesso, corrisponde esattamente a quello della contabilità industriale.

Centro di costo interno manutenzione, ha le stesse funzionalità del codice "centro di costo", solo che serve esclusivamente per uso interno al pacchetto,

Costo orario, costo orario della manutenzione sulla struttura (indipendentemente dal lavoro che si compie),

Descrizione,

Flag acquistato/prodotto, indica se la struttura é stata acquistata o prodotta internamente all'azienda,

Data di acquisto o produzione,

Codice fornitore, codice del fornitore che ha fornito la struttura,

Ore lavorative al giorno,

Ore lavorative nei sabati,

Ore totali di funzionamento previsto, calcolate come visto in precedenza,

Potenza installa, in kilowattore,

Note tecniche, un puntatore a una serie di record della tabella 'note',

Ubicazione della struttura,

Reparto della struttura,

Codice fornitore del contratto di manutenzione, se questo effettivamente esiste, esso è uguale al precedente se chi svolge praticamente la manutenzione, lo fa sotto contratto di assistenza e quindi è lo stesso che ha fornito la struttura,

Nome del calendario su cui bisogna calcolare i giorni lavorativi.

3.2.5 Modulo dei calendari

Questo modulo ha una struttura leggermente diversa da quelli visti fino ad ora, infatti le operazioni implementate sui calendari differiscono leggermente da quelle classiche delle anagrafiche. La maschera dei calendari presenta **31** caselle dove l'utente può inserire un carattere, ci si può spostare tra le caselle con i tasti direzionali (su, giù, destra e sinistra). Inoltre ci sono due campi di output, l'anno del calendario attualmente in uso e il mese visualizzato in quel momento sullo schermo; ci sono due bottoni che permettono di spostarsi al mese precedente o al mese successivo in forma circolare (quindi il successivo del mese 12 è gennaio). Le operazioni che si possono selezionare dal menù sono:

- Apertura di un calendario, viene caricato in memoria l'intero calendario, così facendo si può modificare il calendario come si desidera;

- Nuovo calendario, crea un nuovo calendario inizializzandolo a dovere (con tutti i giorni definiti come lavorativi);
- Visualizza lista dei calendari
- Stampa calendario.

Nella maschera sono anche presenti dei bottoni per il salvataggio dell'intero calendario e per copiare tutto da un calendario già esistente. Infine è presente una legenda con i caratteri che è possibile inserire nelle caselle, essi sono:

'l' per i giorni feriali;

's' per i sabati;

'd' per le domeniche;

'f' per i giorni festivi;

'c' per gli scioperi;

'r' per le ferie collettive;

L'anagrafica calendari contiene le seguenti informazioni:

- Nome del calendario,
- Anno al quale il calendario si riferisce,
- Mese al quale il record si riferisce,
- Numero di giorni di quel mese,
- Insieme di 31 caratteri per definire il mese del calendario.

Da notare che l'anagrafica calendari non ha una chiave primaria, infatti ogni calendario è rappresentato in questa anagrafica da 12 record, uno per ogni mese dell'anno al quale il calendario si riferisce, quindi l'unico indice associato al calendario è quello del nome del calendario, che rappresenta comunque solamente una chiave secondaria.

3.2.6 Modulo della distinta base

- **Cos'e' una distinta base**

La distinta base è l'entità informativa fondamentale per le attività di un'azienda manifatturiera. A questa entità è collegato la maggior parte del sistema informativo. Essa è un archivio che indica, per ogni articolo prodotto dall'azienda, tutti i suoi componenti (*anagrafica parti*) unitamente alle quantità utilizzate e ad altre informazioni necessarie al ciclo produttivo.
Naturalmente, queste informazioni aggiuntive cambiano da un'azienda all'altra e dipendono dallo specifico uso che l'azienda intende fare.
L'anagrafica parti è vista come un contenitore in cui si trovano "*alla rinfusa*" tutti gli oggetti (particolari e composti che formano un prodotto o parti di esso). La distinta base rappresenta, invece, la struttura di un prodotto ed essenzialmente contiene le informazioni su di esso. L'azienda, infatti, può usare questo archivio nelle diverse fasi del ciclo di vita di un articolo.

La distinta base viene utilizzata nella fase di progettazione del prodotto. In questa fase devono essere raccolte e selezionate idee per nuovi prodotti e successivamente redigere i progetti e i disegni dei prodotti di prossima produzione. Affinché il processo di progettazione produca buoni disegni, non sono necessarie solo buone idee, ma occorre che in ogni momento, i dati relativi a questo processo siano ben definiti, per poter essere controllati dai vari organi dell'azienda. Affinché l'articolo sia curato in ogni particolare, occorre suddividere le sue parti in base alla loro funzionalità. In questo ambito la distinta base di progettazione può essere molto utile, in

quanto risponde proprio alle esigenze di definizione delle diverse funzionalità del prodotto.

La distinta base viene utilizzata nella fase di produzione, in questa sede, infatti, essa fa da guida ai processi produttivi e ha la funzione di standardizzazione delle varie fasi del processo produttivo stesso. In questo caso le informazioni relative ad ogni pezzo, che la distinta contiene, non si limitano al solo impiego, ma possono riguardare dimensioni e peso delle varie parti, tempi di lavorazione, giorni di fermo e tante altre informazioni.

La distinta base risulta molto utile per risolvere il problema degli approvvigionamenti dei materiali e dei componenti, necessari a gestire i cicli di produzione. I materiali tenuti in magazzino rappresentano per l'azienda un immobilizzo, che occorre minimizzare. D'altra parte bisogna assicurare, ai reparti di produzione, i materiali necessari a non arrestare la continuità del ciclo produttivo, pena la salita dei costi di gestione. Inoltre, a volte può risultare vantaggioso acquistare quantità sufficienti ad ottenere sconti ed abbuoni dai fornitori. Qualunque sia il metodo utilizzato per la gestione degli approvvigionamenti, tutte le volte che un'azienda riceve un ordine da un cliente deve fare una serie di controlli per vedere se è in grado di soddisfare tale richiesta. Se il pezzo richiesto è un pezzo elementare non ci sono grossi problemi, ma se non lo è, l'azienda deve sapere quali sono le parti che lo compongono e in che quantità, per poter in questo modo controllare le giacenze di magazzino ed effettuare gli eventuali ordini ai fornitori; ed è proprio per questo che la distinta base ha un ruolo piuttosto rilevante.

Le due tipiche elaborazioni che vengono eseguite sulla distinta, al fine di avere le informazioni necessarie sugli articoli, sono l'**esplosione** (o sequenza d'assieme) e l'**implosione** (o sequenza d'impiego). L'esplosione è la trasformazione della struttura del prodotto in un elenco dettagliato di tutti i suoi componenti, siano essi materie prime, parti elementari o assiemi. L'implosione è la ricerca di tutti i possibili utilizzi dei componenti negli assiemi di livello superiore. A seconda del livello a cui si spinge la consultazione della struttura, possiamo distinguere poi i seguenti tipi di analisi:

ad un livello;

scalare;

riepilogata.

Si ottengono le seguenti combinazioni, che rappresentano le fondamentali operazioni sulla distinta base:

Esplosione ad un livello, il risultato di questa elaborazione è un prospetto, in cui ogni assieme risulta composto nei suoi componenti diretti, con l'indicazione dei rispettivi coefficienti d'impiego.

Esplosione scalare, è una tecnica di elaborazione delle informazioni contenute nella struttura di un prodotto, che consente di conoscere la composizione completa di un assieme, distinguendo tutti i suoi componenti, livello per livello, fino all'ultimo elemento.

Esplosione riepilogata, permette di scomporre un assieme in tutti i suoi componenti sottostanti, e di totalizzare le quantità richieste per ogni singolo componente, sommando i diversi consumi unitari quando un componente si presenta più volte all'interno della stessa struttura.

Implosione ad un livello, permette di individuare gli impieghi diretti di un componente.

Implosione scalare, consente di individuare tutti gli impieghi diretti ed indiretti di un componente, fino ai prodotti finiti con l'evidenzazione dei livelli.

Implosione riepilogata, consente di determinare tutti gli impieghi diretti ed indiretti di un componente con l'accumulo delle quantità, a fronte di ciascun assieme.

CODICE	DESCRIZIONE	QUANTITA'
A	....	....
B	....	....
C	....	....
D	....	....
K	....	....
....	....	....

PADRE	FIGLIO	QUANTITA'	TIPO LEGAME	DATA INIZIALE	DATA FINALE
A	B	1	pr	....	....
A	C	4	pr	....	....
P	B	3	fin	....	....
P	K	2	fin	....	....
K	D	5	fin	....	....
K	E	2	fin	....	....
K	F	4	fin	....	....
D	G	2	fin	....	....
D	H	6	fin	....	....
....	....	....	....	....	....

Figura 3.18 Rappresentazione relazionale di una distinta base

Tralasciamo le varie forme di rappresentazione di una distinta base, analizziamo invece come essa viene solitamente implementata in un sistema relazionale, che è fondamentalmente ciò che ci interessa più da vicino (figura 3.18).

Come si può vedere dalla figura 3.18, la distinta base è rappresentata da due archivi, il primo è l'*anagrafica parti*, contenente tutti i componenti presenti in azienda, dalla semplice vite fino ai prodotti finiti; il secondo rappresenta la *struttura*, quindi descrive le relazioni tra le diverse parti. Viene definito un indice di chiave primaria sul codice delle parti, nel secondo archivio la relazione principale è padre-figlio che sarà la chiave. Posso definire due indici, di chiave secondaria, sul secondo archivio, uno su padre e uno su figlio. Tra le altre informazioni che di solito si trovano accanto al legame sono:

il tipo legame, cioè, ad esempio preliminare (ancora in prova) o finale (definitivo);

la data iniziale di validità del legame;

la data finale di validità del legame ecc...

Possiamo mostrare, per maggior chiarezza, una rappresentazione grafica delle distinte base della figura 3.18, i legami in questo caso sono archi che collegano gli elementi per formare un albero (vedi figura 3.19). Possiamo capire facilmente come, ad esempio il prodotto **K** è formato dai componenti **D** **E** e **F**, D a sua volta è composto da **G** e **H**; infine **P** ha come suo componente **K** che è già definito in un preciso modo.

Errore. L'origine riferimento non è stata trovata.

Figura 3.19 Rappresentazione grafica di una distinta base

- **Cenni sul modulo**

La distinta base del mio pacchetto non gioca, chiaramente, un ruolo determinante per la produzione aziendale, come la distinta di progettazione, di produzione o di approvvigionamento, ma risulta tuttavia molto utile per il responsabile delle manutenzioni.

Quando si guasta una struttura, l'addetto alle manutenzioni, consultando la distinta base di quella struttura, può immediatamente venire a conoscenza dei pezzi di ricambio che servono per svolgere il lavoro di manutenzione; inoltre, grazie all'implementazione delle informazioni in linea, può immediatamente sapere a quale fornitore rivolgersi per un eventuale ordine di pezzi di ricambio.

L'anagrafica parti del mio pacchetto non è unica, ma comprende gli archivi delle strutture e dei ricambi, mentre il ruolo di contenitore dei legami è affidato a un archivio chiamato, appunto, 'distinta base'.

Nel mio caso, la distinta base mette in relazione le strutture con i ricambi, quindi un codice di una struttura può solo comparire come elemento padre, non vale la stessa cosa per i ricambi, in quanto un ricambio può essere composto da altri pezzi di ricambio. La distinta base, nel mio specifico caso non avrà più di **tre** o **quattro** livelli.

Anche questo modulo si discosta parecchio dagli altri, infatti la maschera comprende semplicemente: un campo per immettere un codice di una struttura, un campo per immettere un codice di un ricambio, una list-box (lista di elementi) ed un bottone per uscire. Dal menù possono essere selezionate le seguenti voci:

Immettere un nuovo legame di distinta base, in questo caso si richiede l'immissione di un codice struttura, di un codice ricambio e di una quantità;

Cancellare un legame, in questo caso si richiede l'immissione del codice padre, successivamente si sceglie il legame da cancellare dalla list-box che mostra l'elenco dei figli;

Esplosione ad un livello, dopo aver immesso il codice padre, viene visualizzata la lista dei figli di primo livello, selezionando una voce dalla lista si possono ottenere tutte le informazioni in dettaglio;

Esplosione riepilogativa, funziona come la precedente per quanto riguarda l'interfaccia utente, ma esplode fino agli ultimi livelli, elencando così nella listbox solamente i ricambi che non hanno più figli;

Implosione ad un livello, viene richiesta l'immissione del codice figlio, la lista viene così riempita con i codici padri corrispondenti, anche in questo caso selezionando una voce si possono ottenere tutte le informazioni in dettaglio.

L'anagrafica distinta base, come certamente si è capito, è composta dai seguenti campi:

Codice della struttura o del ricambio padre,

Codice del ricambio figlio,

Quantità, ossia il numero di oggetti figli che servono per l'oggetto padre.

3.2.7 Modulo delle richieste di lavoro

Questo è il modulo di gran lunga più importante e complesso, non introduce grosse novità rispetto a implementazioni già viste in moduli precedenti,

però le raggruppa tutte. La maschera comprende l'utilizzo di un testo libero (note tecniche) e di tre gruppi di radiobutton per scelte che si autoescludono (vedi modulo delle strutture). Il menù permette l'interrogazione dell'archivio per codice struttura, per reparto e per intervallo di data (oltre alle consuete visualizza e stampa e alla gestione dei record).

L'unica novità introdotta da questo modulo è l'implementazione di due liste di elementi, questo per gestire i ricambi previsti e i ricambi consuntivi, che, come accennato nei precedenti paragrafi non prevedono un numero fisso di elementi. Queste sono due list-box in cui è possibile modificare gli elementi selezionandoli; inoltre esiste un elemento, chiamato '<*nuovo ric.*>', che abilita l'inserimento di un nuovo ricambio.

Infine parliamo del meccanismo del numero progressivo, che rappresenta anche la chiave della tabella 'rdl'; questo è gestito completamente in automatico dal programma. Quando si immette una nuova richiesta di lavoro, appare in un campo di output il prossimo numero progressivo, quindi quello della nuova richiesta di lavoro. Questo valore viene conservato nel primo record del file, quindi la richiesta di lavoro numero 0 non può esistere, in quanto è adattata a record di comodo.

L'anagrafica richieste di lavoro contiene le seguenti informazioni:

numero progressivo di richiesta di lavoro,

codice della struttura a cui la manutenzione si riferisce,

tipo manutenzione, questo campo è un flag che indica se la manutenzione è programmata, ispettiva o urgente,

data prevista, la data in cui si prevede di effettuare la manutenzione,

data reale, la data in cui si effettua realmente la manutenzione,

ore previste, le ore totali in cui si prevede di effettuare la manutenzione,

ricambi previsti, puntatore ad una lista dei ricambi che si prevede di usare per il lavoro,

flag stato rdl, indica se la manutenzione è evasa, ancora in corso oppure in attesa di esecuzione,

flag interno/esterno, indica se la manutenzione viene svolta internamente o presso un'azienda esterna, fornitrice del servizio,

codice fornitore, colui che eventualmente compie la manutenzione,

costo totale, costo totale del lavoro fatturato dal fornitore, questo viene suddiviso nei costi dei campi seguenti,

costo fatturato dal fornitore per la manodopera,

costo fatturato dal fornitore per i suoi ricambi,

ore consuntivate, le ore totali che ci sono realmente volute per svolgere il lavoro,

ore fermo macchina, ore in cui la macchina è stata ferma,

reparto in cui è situata la struttura,

ricambi consuntivi, puntatore ad una lista dei ricambi che realmente sono serviti per il lavoro,

note, puntatore ad un testo libero.

3.3 Algoritmo per ogni modulo

Ora realizziamo gli algoritmi per ogni modulo. Risulta molto difficile organizzare, in questo momento dell'analisi, degli algoritmi completi per ogni singolo modulo; questo perché, già in fase di progettazione del software è stato introdotto il modo diverso di programmare sotto windows. In pratica, gli algoritmi sono stati pensati per una programmazione ad eventi, tipica delle *graphics user interface*, che avremo modo di approfondire

più avanti. Per queste ragioni, invece di un algoritmo completo per ogni modulo, presento una serie di algoritmi relativi alle operazioni implementate in ogni singolo modulo. Questi algoritmi rappresenteranno solamente frammenti del codice che è stato generato.
Per descrivere gli algoritmi ho utilizzato uno pseudo linguaggio, o linguaggio algoritmico, che comunque rende l'idea dell'algoritmo a qualsiasi persona che s'intende di informatica, o meglio di programmazione. Ricordo che ogni istruzione in neretto e sottolineato indica che quella istruzione viene spiegata più tardi con un altro algoritmo.

3.3.1 Algoritmo dei fornitori

Le operazioni implementate sui fornitori sono: immissione fornitore, aggiornamento fornitore, cancellazione fornitore, visualizzazione e stampa dell'intero archivio. Presentiamo ora gli algoritmi per ognuna di queste operazioni.

Da notare che non viene aggiornato nessun indice, non viene controllato se il record immesso con quella chiave è già presente; tutto questo è gestito e implementato da paradox engine. La stessa cosa vale per i prossimi algoritmi, da questo momento viene ignorato ogni controllo effettuato da paradox engine, risulta sottinteso che i controlli non effettuati direttamente da me sono implementati da paradox engine stesso.

```
Immissione_fornitore
{
 prepara maschera;
 immissione campi;
```

```
 prendi i dati a video e mettili nel buffer;
 appendi un nuovo record;
 pulisci i campi a video;
 cancella la maschera;
}

Aggiornamento_fornitore
{
 chiedi il codice fornitore
 se è stato confermato l'aggiornamento:
 {
  cerca il record con codice fornitore uguale a quello
   inserito;
  se trovato il record
  {
   carica il record nel buffer;
   prendi i dati dal buffer e visualizza a video;
   prepara la maschera;
  }
 }
 altrimenti messaggio di errore "record non trovato";
 }
 aggiorna i campi;
 prendi i dati a video e mettili nel buffer;
 aggiorna un nuovo record;
 pulisci i campi a video;
 cancella la maschera;
}
 Cancellazione_fornitore
 {
  chiedi il codice fornitore;
  cerca il record;
  se trovato
  {
   carica i dati nel buffer;
   prendi i dati dal buffer e visualizzali a video;
   chiedi conferma per la cancellazione;
   se cancellazione confermata
    cancella il record;
   pulisci i campi della maschera;
  }
  altrimenti messaggio di errore "record non trovato";
```

```
}


Visualizzazione_intero_archivio
{
 posizionati sul primo record;
 mentre è abilitata la possibilità di una prossima pagina
 {
  mentre ci sono record nel file
  {
   carica i dati nel buffer;
   stabilisci il formato di visualizzazione;
   visualizza i dati del record letto;
   se non è possibile visualizzarli
   {
    abilita la possibilità di una prossima pagina;
    esci dal ciclo;
   }
   altrimenti disabilita la possibilità di una prossima pagina;
   posizionati sul prossimo record;
  }
 }
}


Stampa_intero_archivio
{
 crea un device context associato alla stampante impostata;
 se è possibile creare il device context
 {
  manda un messaggio di inizio documento alla stampante;
  posizionati sul primo record;
  contarighe=0;
  mentre ci sono record nel file
  {
   carica i dati nel buffer;
   stabilisci il formato di stampa;
   se contarighe=20
   {
    contarighe=0;
    invia alla stampante messaggio di nuova pagina;
```

```
    }
    stampa i dati del record letto;
    posizionati sul prossimo record;
    incrementa contarighe;
   }
   invia messaggio di nuova pagina alla stampante;
   invia messaggio di fine documento;
   rilascia device context;
  }
  altrimenti messaggio di errore "Errore nella stampante";
 }
```

3.3.2 Algoritmo dei ricambi

Le operazioni implementate sui ricambi sono: immissione ricambio, aggiornamento ricambio, cancellazione ricambio, visualizzazione e stampa dell'intero archivio. Presentiamo ora gli algoritmi per ognuna di queste operazioni.

```
Immissione_ricambio
{
 prepara la maschera;
 immetti i dati;
 prendi i dati a video e mettili nella struttura in memoria;
 controlla i dati inseriti;

 se ci sono campi errati
 {
  mostra l'errore;
  finisci;
 }
 trasferisci i dati dalla struttura al buffer;
 appendi un nuovo record;
 pulisci i campi a video;
 pulisci la struttura in memoria;
 cancella la maschera;
}
```

```
Aggiornamento_ricambio
{
 chiedi il codice ricambio;
 cerca il record con codice ricambio dato;
 se trovato il record
 {
  carica i dati nel buffer;
  prendi i dati dal buffer e visualizzali a video;
  prepara la maschera
  aggiorna i dati;
  se è stato modificato il fornitore ultimo
  {
   metti i dati del fornitore 1 nei campi della struttura in
   memoria riservati al fornitore 2;
   metti i dati del fornitore ultimo nei campi della struttura in
   memoria riservati al fornitore 1;
  }
  prendi i dati a video e mettili nel record;
  controlla i dati inseriti;
  se ci sono campi errati
  {
   mostra l'errore;
   finisci;
  }
  trasferisci i dati dalla struttura al buffer;
  aggiorna il record corrente;
  pulisci i campi a video;
  pulisci la struttura in memoria;
  cancella la maschera;
 }
 altrimenti messaggio "record non trovato";
}
Cancellazione_ricambio
{
 chiedi il codice ricambio;
 cerca il record;
 se trovato
 {
  carica i dati nel buffer;
  prendi i dati dal buffer e visualizzali a video;
  chiedi conferma per la cancellazione;
  se cancellazione confermata
   cancello il record;
```

```
 pulisci i campi della maschera;
}
altrimenti messaggio di errore "record non trovato";
}
```

La visualizzazione e la stampa dell'intero archivio sono identiche al modulo precedente, solo che la ricerca è fatta sull'indice costruito sul codice ricambio della tabella dei ricambi.

3.3.3 Algoritmo dei calendari

A differenza dei due moduli precedenti, molto simili tra di loro, in questo modulo le operazioni realizzate sono leggermente diverse: apertura di un calendario, creazione di un nuovo calendario, copia di un calendario, stampa di un calendario. Presentiamo ora gli algoritmi per ognuna di queste operazioni.

```
Apertura_calendario
{
 chiedi il nome del calendario
 se il calendario esiste
 {
  carica calendario in memoria;
  anno corrente = anno del calendario caricato;
  mese corrente = 1;
  prepara la maschera;
  visualizza a video il mese corrente;
  aggiornamento_calendario;
  se i caratteri sono corretti
  {
   metti nel buffer il calendario in memoria;
   per ogni mese
   {
    cerca il record;
    aggiorna il record;
```

```
    }
    pulisci lo schermo;
    cancella la maschera;
   }
   altrimenti
   {
    messaggio "carattere errato";
    evidenzia carattere errato;
   }
  }
  altrimenti messaggio "calendario non esistente";
 }
```

L'aggiornamento del calendario è un po' particolare, in quanto, come già visto quando abbiamo parlato della modularizzazione del programma, solo un mese alla volta è visualizzabile sullo schermo; vediamo l'algoritmo che gestisce il movimento tra i mesi di un calendario in memoria.

```
Aggiornamento_calendario
{
 mese corrente = 1;
 carica il mese corrente dalla memoria e visualizza;
 aggiorna il mese corrente;
 se richiesto mese successivo
 {
  se i caratteri immessi sono giusti
  {
   prendi i dati a video e mettili in memoria;
   incrementa il mese corrente in forma ciclica;
   carica il mese corrente dalla memoria e visualizza;
  }
  altrimenti
   manda messaggio di errore evidenziando la casella errata;
 }
 se richiesto il mese precedente
 {
  se i caratteri immessi sono giusti
  {
   prendi i dati a video e mettili in memoria;
   decrementa il mese corrente in forma ciclica;
```

```
   carica il mese corrente dalla memoria e visualizza;
  }
  altrimenti
   manda messaggio di errore evidenziando la casella errata;
 }
}

Creazione_nuovo_calendario
{
 chiedi il nome del calendario e l'anno;
 se il calendario non esiste
 {
  salva calendario di default;
  carica il calendario in memoria;
  mese corrente = 1;
  prepara la maschera;
  visualizza a video il mese corrente;
  aggiornamento_calendario;
  se i caratteri sono corretti
  {
   metti nel buffer il calendario in memoria;
   per ogni mese
      cerca ed aggiorna il record;
   pulisci lo schermo;
   cancella la maschera;
  }
  altrimenti
  {
   messaggio "carattere errato";
   evidenzia carattere errato;
  }
 }
 altrimenti messaggio "calendario già esistente";
}
```

Quando viene creato un nuovo calendario, prima viene salvato nell'archivio il calendario di default (con tutti i giorni lavorativi), successivamente lo stesso calendario viene caricato ed aggiornato. Il salvataggio del calendario è simile alla seconda parte dell'algoritmo di aggiornamento, solo che il record in memoria viene caricato con caratteri 'l'.

```
Copia_calendario
{
 chiedi il nome calendario;
 se esiste il calendario
 {
  carica in memoria il calendario da copiare;
  mese_corrente=1;
  visualizza a video il mese corrente;
  aggiornamento_calendario;
 }
 altrimenti messaggio "calendario corrente";
}
```

```
Stampa_calendario
{
 chiedi il nome del calendario;
 se il calendario esiste
 {
  crea un device context associato alla stampante impostata;
  se è possibile creare il device context
  {
   manda un messaggio di inizio documento alla stampante;
   ottengo l'anno corrente;
   ottengo il nome del calendario da stampare;
   stampo l'intestazione;
   per mese corrente che va da 1 a 12
   {
    stampa numero del mese;
    per i che va da 0 al numero dei giorni di mese corrente
     stampa carattere i di mese corrente
   }
   invia messaggio di nuova pagina alla stampante;
   invia messaggio di fine documento;
   rilascia device context;
  }
  altrimenti messaggio di errore "Errore nella stampante";
 }
 altrimenti messaggio "calendario non esistente";
}
```

3.3.4 Algoritmo delle strutture

Le operazioni implementate in questo modulo sono simili ai moduli dei fornitori e dei ricambi, vengono aggiunte le operazioni che riguardano le note e una ricerca sull'archivio per intervallo di data. Le operazioni sono: immissione struttura, aggiornamento struttura, cancellazione struttura, visualizzazione strutture per intervallo di data di produzione o di acquisto ed infine visualizzazione e stampa dell'intero archivio.

```
Immissione_struttura
{
 inizializza i radiobutton;
 pulisci i campi a video;
 pulisci il record in memoria;
 prepara la maschera;
 setta a 0 il flag di modifica delle note;
 immetti dati nella maschera;
 prendi i dati dallo schermo e mettili nel record in memoria;
 controlla i dati inseriti;
 se ci sono campi errati
   mostra l'errore e finisci;
 chiedi il nome del calendario da associare alla struttura;
 calcola_ore_totali_funzionamento;
 verifica se le note sono state modificate;
 se le note sono state modificate salva_note;
 se le note inserite sono nulle
  metti nel puntatore alle note il puntatore nullo;
 trasferisci i dati del record in memoria al buffer;
 appendi un nuovo record;
 pulisci i campi a video;
 cancella la maschera;
}
Calcola_ore_totali_funzionamento
{
 apri la tabella calendario;
 crea il buffer associato alla tabella calendario;
 cerca il primo record con nome calendario dato;
 se non esiste il calendario
```

```
 {
  messaggio "calendario non esistente";
  chiudo il buffer;
  chiudo il calendario;
  finisci;
 }
 mentre ci sono record
 {
  carica i dati nel buffer;
  somma1=somma1+caratteri 'l' del mese caricato;
  somma2=somma2+caratteri 's' del mese caricato;
  cerca il prossimo record;
 }
 somma1=somma1*ore_g_lav+somma2*ore_g_sab;
 chiudi il buffer;
 chiudi la tabella;
}


Salva_note
{
 verifica se esiste la tabella note;
 se non esiste
 {
  crea la tabella note;
  crea la chiave primaria sull'indice delle note;
 }
 apri la tabella note;
 crea il buffer associato alle note;
 cont=numero di linee delle note;
 se cont>40
 {
  cont=40;
  messaggio "note troppo lunghe, perse le ultime righe";
 }
 se la tabella non è vuota
 {
  cerco il primo record;

  mentre ci sono record e i<cont
  {
   prelevo la riga i delle note;
   aggiorno la riga;
```

```
  cerco il prossimo record;
 }
 se i=cont
 {
  mentre ci sono record
  {
   cancella il record;
   cerco il prossimo record;
  }
 }
 prossimo=i;
 per i che va da prossimo a cont-1
 {
  ottieni la riga i;
  aggiungi il record;
 }
}
chiudi il buffer;
chiudi la tabella;
}
```

```
Aggiornamento_struttura
{
 chiedi il codice ricambio;
 cerca il record con codice struttura uguale a quello inserito;
 se trovato il record
 {
  carica il record nel buffer;
  prendi i dati dal buffer e visualizza a video;
  prepara la maschera;
  se il puntatore alle note non è nullo
   carica_e_visualizza_note;
  metti a 0 il flag per le note;
 }
 altrimenti messaggio di errore "record non trovato";
 aggiorna le informazioni sullo schermo;
 prendi i dati dallo schermo e mettili nel record in memoria;
 controlla i dati inseriti;
 se ci sono campi errati
 {
  mostra l'errore;
```

```
 finisci;
}
chiedi il nome del calendario da associare alla struttura;
calcola_ore_totali_funzionamento;
verifica se le note sono state modificate;
se le note sono state modificate salva_note
se le note inserite sono nulle
 metti nel puntatore alle note il puntatore nullo;
trasferisci i dati del record in memoria al buffer;
aggiorna il record corrente;
pulisci i campi a video;
cancella la maschera;
}


Carica_e_visualizza_note
{
apri la tabella note;
se è possibile aprire la tabella
{
 apri un buffer associato alle note;
 se la tabella non è vuota
 {
  cerca il primo record;
  se esistono le note
  {
   azzera testo globale;
   nrighe=0;
   mentre ci sono record
   {
    incrementa nrighe;
    carica il record;
    collega la riga al testo globale;
    inserisci caratteri di fine riga;
   }
   trasferisci il testo globale nelle note;
  }
 }
 chiudi buffer;
 chiudi tabella;
}
altrimenti
{
```

```
  crea la tabella note;
  crea la chiave secondaria sull'indice delle note;
  messaggio "archivio note non esistente";
 }
}


Cancellazione_struttura
{
 chiedi il codice struttura;
 cerca il record;
 se trovato
 {
  carica i dati nel buffer;
  prendi i dati dal buffer e mettili nel record in memoria;
  visualizza a video i dati;
  chiedi conferma per la cancellazione;
  se cancellazione confermata
  {
   ottengo il puntatore alle note;
   cancello il record;
   apro la tabella note;
   creo un buffer per le note;
   se la tabella non è vuota
   {
    ricerca il record con indice = puntatore alle note;
    mentre ci sono record cancella il record;
   }
   elimina il buffer per le note;
   chiudi la tabella note;
  }
  pulisci i campi della maschera;
 }
 altrimenti messaggio di errore "record non trovato";
}


Visualizzazione_strutture_per_intervallo_di_data
{
 chiedi le date (data iniziale e finale);
 converti data iniziale in formato numerico;
 converti data finale in formato numerico;

 per data iniziale che va dal valore corrente alla data finale
```

```
{
 ricerca il primo record con data = data iniziale
 mentre è abilitata la possibilità di una prossima pagina
 {
  mentre ci sono record
  {
   carica i dati nel buffer;
   stabilisci il formato di visualizzazione;
   visualizza il record letto;
   se non è possibile visualizzarlo
   {
    abilita la possibilità di una prossima pagina;
    esci dal ciclo;
   }
   altrimenti disabilita la possibilità di una prossima pagina;
   cerca il prossimo record;
  }
 }
}
}
```

La visualizzazione e la stampa dell'intera anagrafica delle strutture presentano uno schema algoritmico simile ai precedenti moduli riguardanti i fornitori o i ricambi, quindi evitiamo di ripresentare gli algoritmi, tenendo presente che le operazioni sono eseguite sulla tabella delle strutture.

3.3.5 Algoritmo delle richieste di lavoro

Questo è il modulo più complesso, ma le operazioni implementate sono le stesse dei moduli precedenti. Sono aggiunte le operazioni per gestire le note e le due liste di ricambi, oltre a diverse ricerche sull'archivio. Le operazioni sono: immissione richiesta di lavoro, aggiornamento richiesta di lavoro, cancellazione richiesta di lavoro, ricerca richieste di lavoro per codice

struttura, per reparto, per intervallo di data reale di manutenzione ed infine le solite operazioni di visualizzazione e stampa dell'intero archivio.

```
Immissione_richieste_di_lavoro
{
 inizializza i radiobutton;
 pulisci il record in memoria;
 pulisci i campi a video;
 cerca il record con progressivo uguale a 0;
 leggi il progressivo conservato nel campo numero 6;
 visualizza il progressivo a video;
 prepara la maschera;
 setta a 0 il flag delle modifiche delle note;
 setta a 0 il flag delle modifiche dei ricambi previsti;
 setta a 0 il flag delle modifiche dei ricambi consuntivi;
 immetti i dati nella maschera;
 leggi flag di modifica delle note;
 prendi i dati dallo schermo e mettili nel record in memoria;
 controlla i dati inseriti;
 se ci sono campi errati
 {
  mostra l'errore;
  finisci;
 }
 trasferisci i dati nel buffer;
 appendi un nuovo record;
 se è possibile salvare
 {
  cerca record con chiave uguale a 0;
  leggi il progressivo nel campo numero 6;
  incrementa progressivo;
  aggiorna record;
 }
 verifica se le note sono state modificate;
 se le note sono state modificate salva_note;
 verifica se la list box 1 è stata modificata;
 se è stata modificata salva_listbox;
 verifica se la list box 2 è stata modificata;
 se è stata modificata salva_listbox;
 pulisci i campi a video;
 cancella la maschera;
```

```
}
```

La funzione di salvataggio delle note è uguale a quella mostrata nel modulo delle strutture.

```
Salva_listbox
{
 verifica se esiste la tabella ric;
 se non esiste
 {
  crea la tabella ric;
  crea la chiave primaria sull'indice dei ricambi;
 }
 apri la tabella ric;
 crea il buffer associato ai ricambi;
 cont=numero di elementi nella list box;
 se cont>80
 {
  cont=80;
  messaggio "troppi ricambi, persi ultimi elementi";
 }
 se la tabella non è vuota
 {
  i=1;
  cerco il primo record;
  mentre ci sono record e i<cont
  {
   prelevo l'elemento i della list box;
   aggiorno il record;
   cerco il prossimo record;
   incrementa i;
  }
  se i=cont
  {
   mentre ci sono record
   {
    cancella il record;
    cerco il prossimo record;
   }
  }
  prossimo=i;
  per i che va da prossimo a cont-1
```

```
  {
   ottieni l'elemento i dalla lista sullo schermo;
   aggiungi il record;
  }
 }
 chiudi il buffer;
 chiudi la tabella;
}
```

```
Aggiornamento_richiesta_di_lavoro
{
 pulisci il record in memoria;
 chiedi il progressivo;
 cerca il record con progressivo uguale a quello inserito;
 se trovato il record
 {
  carica il record nel buffer;
  prendi i dati dal buffer e visualizza a video;
  prepara la maschera;
  se il puntatore ai ricambi previsti non è nullo
   carica_e_visualizza lista;
  se il puntatore ai ricambi consuntivi non è nullo
   carica_e_visualizza lista;
  se il puntatore alle note non è nullo
   carica_e_visualizza note;
  metti a 0 il flag che controlla se le note sono modificate;
  metti a 0 il flag per la modifica dei ricambi previsti;
  metti a 0 il flag per la modifica dei ricambi consuntivi;
 }
 altrimenti messaggio di errore "record non trovato";
}
```

La funzione che carica e visualizza le note è uguale a quella del modulo delle strutture.

```
Carica_e_visualizza_lista
{
 apri la tabella ric;
```

```
se è possibile aprire la tabella
{
 apri un buffer associato alla tabella ric;
 se la tabella non è vuota
 {
  cerca il primo record;
  se esistono i ricambi
  {
   nrighe=0;
   mentre ci sono record
   {
    incrementa nrighe;
    carica il record;
    aggiungi il record alla lista;
   }
  }
 }
 chiudi buffer;
 chiudi tabella;
}
altrimenti
{
 crea la tabella ric;
 crea la chiave secondaria sull'indice dei ricambi;
 messaggio "archivio ricambi non esistente";
}
}

Cancellazione_richiesta_di_lavoro
{
 chiedi il progressivo;
 cerca il record con progressivo dato;
 se trovato
 {
  carica i dati nel buffer;
  prendi i dati dal buffer e mettili nel record in memoria;
  visualizza a video i dati;
  chiedi conferma per la cancellazione;
  se cancellazione confermata
  {
   ottengo il puntatore alle note;
   ottengo i puntatori ai ricambi;
   cancello il record;
```

```
  cancella_ricambi (previsti)
  cancella_ricambi (consuntivi)
  apro la tabella note;
  creo un buffer per le note;
  se la tabella non è vuota
  {
   ricerca il record con indice = puntatore alle note;
   mentre ci sono record cancella il record;
  }
  elimina il buffer per le note;
  chiudi la tabella note;
 }
 pulisci i campi della maschera;
}
altrimenti messaggio di errore "record non trovato";
}

Cancella_ricambi
{
 apri la tabella ric;
 crea il buffer associato alla tabella ric;
 se la tabella non è vuota
 {
  ricerca il record con indice = puntatore ai ricambi;
  mentre ci sono record cancella il record;
 }
 elimina il buffer;
 chiudi la tabella ric;
}

Ricerca_richieste_di_lavoro_per_codice_struttura
{
 chiedi il codice struttura;
 cerca il primo record con codice struttura dato;
 mentre è abilitata la possibilità di una prossima pagina
 {
  mentre ci sono record
  {
   carica i dati nel buffer;
   se il record ha le caratteristiche richieste
   {
    visualizza il record letto nella lista;
```

```
      se non è possibile aggiungere
      {
       abilita la possibilità di una prossima pagina;
       esci dal ciclo;
      }
      altrimenti disabilita la possibilità di una prossima pagina;
     }
     cerca il prossimo record;
    }
   }
  }

  Ricerca_richieste_di_lavoro_per_reparto
  {
   chiedi il reparto;
   cerca il primo record con reparto dato;
   mentre è abilitata la possibilità di una prossima pagina
   {
    mentre ci sono record
    {
     carica i dati nel buffer;
     se il record ha le caratteristiche richieste
     {
      visualizza il record letto nella lista;
      se non è possibile aggiungere
      {
       abilita la possibilità di una prossima pagina;
       esci dal ciclo;
      }
      altrimenti disabilita la possibilità di una prossima pagina;
     }
     cerca il prossimo record;
    }
   }
  }
```

La visualizzazione e la stampa dell'intero archivio sono uguali alle precedenti e quindi vengono ignorate.

3.3.6 Algoritmo della distinta base

Questo è un altro modulo con operazioni particolari, infatti sono implementate le tipiche operazioni sulla distinta base. Esse sono: creazione legame, cancellazione legame, implosione ed esplosione scalare ad un livello. Le operazioni di esplosione riepilogativa e di stampa di una distinta base non sono state implementate, in quanto non di urgente bisogno da parte della azienda.

```
Creazione_legame_distinta_base
{
 chiedi il codice padre;
 restituisci_la_descrizione;
 chiedi il codice figlio;
 restituisci_la_descrizione;
 chiedi conferma per la creazione;
 se confermato
 {
  chiedi quantità;
  aggiungi record;
 }
 chiedi conferma per creazione di un altro legame;
 se confermato
 {
  pulisci campo di immissione del codice figlio;
  pulisci campo di descrizione del codice figlio;
 }
 altrimenti
  pulisci lo schermo;
}

Restituisci_descrizione
{
 apri tabella (strutture o ricambi);
 crea un buffer;
 cerca record con codice dato;
 se trovato
  ottieni descrizione;
```

```
 chiudi buffer;
 chiudi tabella;
 ritorna un flag per indicare l'esito dell'operazione;
}


Cancellazione_legame_di_distinta_base
{
 chiedi codice padre;
 se non esiste la struttura
 {
  se non esiste il codice padre come ricambio
   messaggio "codice non esistente";
 }
 restituisci_descrizione;
 se ci sono legami riempi_lista;
 seleziona codice figlio;
 chiedi conferma per la cancellazione;
 se confermato
 {
  cerca record da cancellare;
  se trovato cancella record
  altrimenti messaggio "legame non esistente";
 }
 elimina l'elemento dalla lista;
}

Riempi_lista
{
 apri tabella (ricambi o strutture);
 apri buffer;
 cerca record con chiave data;
 se non trovato
 {
  messaggio "legami non presenti";
  chiudi buffer;
  chiudi tabella;
  finisci;
 }
 mentre ci sono record
 {
  ottengo il codice da visualizzare;
```

```
 cerco il record nella tabella strutture o ricambi con il codice
  dato;
 se trovato
 {
  carica record;
  stabilisci formato;
  aggiungi elemento alla lista;
 }
 altrimenti messaggio "legame ignorato";
 cerca prossimo record;
}
chiudi buffer;
chiudi tabella;
}
```

```
Esplosione_scalare_ad_un_livello
{
 chiedi codice padre;
 se non esiste la struttura
 {
  se non esiste il codice padre come ricambio
   messaggio "codice non esistente";
 }
 restituisci_descrizione;
 se ci sono legami riempi_lista;
 seleziona elemento dalla lista;
 visualizza informazioni sul ricambio selezionato;
}
```

```
Implosione_scalare_ad_un_livello
{
 chiedi codice figlio;
 se non esiste il ricambio
  messaggio "codice non esistente";
 restituisci_descrizione;
 se ci sono legami riempi_lista;
 seleziona elemento dalla lista;
 visualizza informazioni sulla struttura o ricambio selezionato;
}
```

3.4 Traduzione dell'algoritmo in programma

Questo particolare momento, dello sviluppo di un software, generalmente, non è mai molto interessante, e non crea neanche grosse difficoltà. Nel mio specifico caso non è stato proprio cosi', infatti, ho dovuto apprendere un nuovo linguaggio di programmazione e, soprattutto, un nuovo stile di programmazione, la cosiddetta 'programmazione ad eventi'. Lo sviluppo di questo software si è articolato in due parti: l'implementazione della gestione degli archivi e l'implementazione dell'interfaccia grafica con l'utente.
La prima parte non è stata particolarmente difficile, il tutto si è ridotto ad imparare ad utilizzare le funzioni di paradox engine; la seconda parte, invece, mi ha costretto a cambiare il modo di organizzare i programmi; infatti, in ambiente windows, ma più in generale in qualsiasi ambiente GUI (graphics user interface), bisogna saper interagire con chi ti ospita, in questo caso 'windows'. Ora approfondiamo brevemente l'uso di paradox engine e della programmazione ad eventi.

3.4.1 Utilizzare paradox engine

Paradox Engine si può definire come una soluzione software che apre l'architettura di paradox al **pascal**, al **'c'** e, dalla versione 2, al mondo **windows**. Infatti, esso è composto da un'insieme di librerie, o meglio due librerie pascal, due librerie 'c' e una dynamic link library (dll) per windows; queste librerie contengono circa **70** funzioni che implementano le classiche operazioni che si fanno con un Data Base Management System. Questo pacchetto è nato accanto al famoso paradox, per integrare il linguaggio **PAL**,

ossia il linguaggio incluso in paradox stesso, allo scopo di allargarne le applicazioni.

In altre parole, chi vuole usare le tabelle di paradox e utilizzare strumenti hardware particolari, ad esempio un lettore di codici a barre, può implementare queste operazioni con un linguaggio più vicino all'hardware, qual'è il 'c' e accedere alle tabelle attraverso paradox engine. Tutto ciò che si fa con paradox engine è chiaramente fattibile con paradox, esso quindi rappresenta solamente un mezzo per accedere alle tabelle di paradox senza conoscerne minimamente il formato o l'implementazione delle informazioni.

Il grosso vantaggio di queste librerie, almeno nel momento in cui sto scrivendo questa tesi, è la compatibilità con le applicazioni in ambiente windows, questo indubbiamente porta il sottoscritto e l'azienda ad utilizzare sistemi di basi di dati in un ambiente, windows appunto, dove non è presente niente del genere, anzi l'unica casa, costruttrice di software, ad aver annunciato un DBMS sotto Windows è proprio la Borland con la nuova versione di Paradox.

Detto questo, cerchiamo ora di approfondire l'utilizzo di queste librerie, che sono state ampiamente sfruttate nella stesura del codice, citando soprattutto le funzioni da me più utilizzate.

Paradox Engine mette a disposizione dei programmatori circa 70 funzioni, raggruppate in insiemi ben definiti:

funzioni che riguardano la definizione delle tabelle;

funzioni che gestiscono le operazioni classiche sui record;

funzioni che riguardano i campi;

funzioni che gestiscono gli indici associati alle tabelle;

funzioni che gestiscono le operazioni in rete;

funzioni che gestiscono la sicurezza e la segretezza delle tabelle;

Le funzioni utilizzate maggiormente da me sono state quelle che gestiscono le operazioni sui record e quelle che implementano le operazioni di ricerca, quindi non ho avuto modo di conoscere meglio alcuni aspetti interessanti di queste librerie, come poteva essere la gestione della concorrenza in ambiente di rete, o la possibilità di associare delle password alle tabelle stesse. Vediamo nel dettaglio alcune funzioni seguendo i gruppi definiti precedentemente. Per quanto riguarda le tabelle possiamo ricordare:

PXtblCreate per creare una tabella, specificando il numero dei campi e il loro rispettivo tipo, molto gradita è stata l'implementazione di un tipo data;

PXTblOpen per aprire la tabella;

PXTblClose per chiudere una tabella;

PXTblExist per accertarne l'esistenza;

Per quanto riguarda i record, ricordiamo che essi sono numerati in ordine di chiave primaria ed esiste sempre un puntatore al record corrente, possiamo ricordare:

PXRecAppend per aggiungere un record;

PXRecDelete per cancellare il record corrente;

PXRecUpdate per aggiornare il record corrente.

PXTblNext per incrementare di uno il puntatore al record;

Un meccanismo un po' particolare è utilizzato per accedere ai dati delle tabelle tenendo nascosto il vero formato dei campi. È implementato un **buffer**, che corrisponde a un record della tabella, è possibile creare un

elevato numero di buffer, ed esso rappresenta l'unico mezzo per effettuare Input/Output con le tabelle. Sono implementate alcune funzioni che permettono di inserire i dati nel buffer, utilizzando appositi numeri di riferimento dei campi, ed altre per leggere i dati dal buffer:

PXRecBufOpen per creare un buffer;

PXRecBufClose per distruggere un buffer;

PXPutAlpha per scrivere un dato alfanumerico nel buffer;

PXPutShort per scrivere un dato numerico intero nel buffer;

PXPutDate per scrivere un dato di tipo data nel buffer.

PXGetAlpha per leggere un dato alfanumerico dal buffer;

PXGetShort per leggere un dato numerico intero dal buffer;

PXGetDate per leggere un dato di tipo data dal buffer.

Una, invece, è la funzione per leggere i dati contenuti nel record corrente, **PXRecGet**. Infine ci sono le funzioni che permettono la gestione degli indici:

PXKeyAdd per aggiungere un indice, primario o secondario, alla tabella;

PXSrchFld per ricercare un record con un campo precedentemente fissato;

PxSrchKey per ricercare un record con una chiave fissata a priori.

3.4.2 La programmazione ad eventi

Abbiamo già accennato agli aspetti nuovi introdotti da windows, precisamente nel capitolo in cui abbiamo discusso le scelte degli strumenti di lavoro. Ora vediamo cosa offre windows ai programmatori e cosa i programmatori devono offrire a windows, cercando di riprendere il discorso introduttivo del capitolo 2.

■ **Struttura dei programmi**

Abbiamo detto che ogni finestra che appare sullo schermo, ha una corrispondente funzione (window function) con il duplice compito di comunicare al programma quello che l'utente sta facendo sulla finestra e quello di gestire la finestra come richiede il programma. Queste comunicazioni non avvengono direttamente, come ad esempio le call a funzioni dell'ambiente DOS, ma tramite dei messaggi. Per questo fatto, tutte le window function hanno la stessa struttura. Più precisamente sono costituite da un'istruzione di condizionamento multiplo (switch del 'c' ad esempio); questa istruzione prende in esame il messaggio arrivato alla funzione e decide poi cosa fare in base al tipo di messaggio stesso. Dopo aver verificato il tipo di messaggio, si salta a un determinato punto del programma; da questo momento in poi, si compiono tutte le operazioni che il messaggio richiede; quindi il programmatore può scrivere un segmento di codice, esattamente come in ambiente DOS; questo vale in modo particolare in 'c' dove le istruzioni sono indipendenti dal sistema operativo in cui si lavora.

Un programma 'c' presenta sempre una window function per ogni finestra utilizzata, in più c'è la funzione winmain che ha il compito di creare almeno la finestra principale e distribuire i messaggi alle varie window function. Oltre alle finestre, vengono gestite le dialog box nello stesso modo, cioè le finestre che chiedono piccole informazioni di input, cambiano solo i tipi di messaggi gestiti. Ora voglio mostrare due frammenti di sorgente, relativi al modulo dei fornitori: nel primo frammento si vede la struttura della window

function della finestra principale; nel secondo si vede la dialog function che gestisce la dialog box che chiede l'input del codice fornitore.

```
long FAR PASCAL principale(HWND hwnd,unsigned messaggio,
 WORD wparam,LONG lparam)
{
 HDC hdc,printdc;
 PAINTSTRUCT ps;
 FARPROC lpdlgabout,lpdlgcodice,lpdlgvisual;
 int i,esiste,pxerr,contarighe,posy;
 recforn rec;
 char formato[50],riga[ltotale];

 switch(messaggio)
 {

  case WM_CREATE:
/* inizializzazioni paradox engine */
   er(pxerr=PXWinInit(nome_applicazione,PXSINGLECLIENT));
   if (pxerr==PXSUCCESS)
   {
/* esiste testa l'esistenza della tabella */
    er(PXTblExist(nome_tabella,&esiste));
    if (!esiste) crea_tabella(nome_tabella);
/* apro tabella e buffer */
    er(PXTblOpen(nome_tabella,&tblhandle,0,1));
    er(PXRecBufOpen(tblhandle,&rechandle));
/* inizializzazioni grafiche */
    hdc=GetDC(hwnd);
/* funzione che restituisce la altezza */
    scegli_font(hdc,OEM_FIXED_FONT);
    xcar=dim_larg(hdc);
    ycar=dim_altez(hdc);
    ReleaseDC(hwnd,hdc);
/* fisso lo stato e il focus iniziale e disabilito la maschera */
    stato=madre;
    SetFocus(hwnd);
    abilita_tutto(FALSE);
   }
   else
    SendMessage(hwnd,WM_CLOSE,0,0L);
```

```
  break;

 case WM_COMMAND:
 {

  switch(wparam)
  {

   case IDM_ABOUT:
/* richiamo la dialog box di informazione */
    lpdlgabout=MakeProcInstance(aboutproc,hinst);
    DialogBox(hinst,"about",hwnd,lpdlgabout);
    FreeProcInstance(lpdlgabout);
    break;

   case IDM_IMMISSIONE:
/* abilito i campi e disabilito i menu */
    abilita_menu(hwnd,IDM_IMMISSIONE,IDM_STAMPA,
     MF_GRAYED|MF_DISABLED);
    stato=imm;
    abilita_tutto(TRUE);
    SetFocus(h[0]);
    break;

   case IDM_AGGIORNAMENTO:
    stato=agg;
/*chiedo codice con la dialogbox codice e carico vecchio record */
    lpdlgcodice=MakeProcInstance(codiceproc,hinst);
    DialogBox(hinst,"codice",hwnd,lpdlgcodice);
    FreeProcInstance(lpdlgcodice);
/* controllo che sia stato premuto ok */
    if (stato)
    {
     er(PXPutAlpha(rechandle,1,codice));
     er(pxerr=PXSrchFld(tblhandle,rechandle,1,SEARCHFIRST));
     if ((pxerr!=PXERR_RECNOTFOUND)&&
        (PXRecGet(tblhandle,rechandle)!=PXERR_TABLEEMPTY))
     {
      abilita_menu(hwnd,IDM_IMMISSIONE,IDM_STAMPA,
       MF_GRAYED|MF_DISABLED);
/* carico i dati */
      er(PXRecGet(tblhandle,rechandle));
      prendi_buffer(rechandle);
```

```
      abilita_tutto(TRUE);
      SetFocus(h[0]);
     }
    }
    break;

   case IDM_CANCELLAZIONE:
    stato=canc;
/*chiedo codice con la dialogbox codice e cancello vecchio record */
    lpdlgcodice=MakeProcInstance(codiceproc,hinst);
    DialogBox(hinst,"codice",hwnd,lpdlgcodice);
    FreeProcInstance(lpdlgcodice);
/* controllo che sia stato premuto ok */
    if (stato)
    {
     er(PXPutAlpha(rechandle,1,codice));
     er(pxerr=PXSrchFld(tblhandle,rechandle,1,SEARCHFIRST));
     if ((pxerr!=PXERR_RECNOTFOUND)&&
        (PXRecGet(tblhandle,rechandle)!=PXERR_TABLEEMPTY))
     {
/* visualizzo e chiedo_conferma */
      er(PXRecGet(tblhandle,rechandle));
      prendi_buffer(rechandle);
      if(MessageBox(hwnd,"Vuoi cancellare il fornitore?",nome_pacchetto,
       MB_ICONQUESTION|MB_OKCANCEL|MB_DEFBUTTON2)
       ==IDOK)
      {
/* cancello il record */
       er(PXRecDelete(tblhandle));
      }
     }
     azzera_edit(0,nfin-2);
    }
    break;

   case IDM_USCITA:
/* selezionato uscita dal menu */
    SendMessage(hwnd,WM_CLOSE,0,0L);
   break;

   case IDM_VISUALIZZA:
/* apro una dialogbox con dentro una listbox e visualizza tutti i record
   del file, inoltre ci sono due bottoni : stampa e uscita */
```

```
    lpdlgvisual=MakeProcInstance(visualproc,hinst);
    DialogBox(hinst,"visual",hwnd,lpdlgvisual);
    FreeProcInstance(lpdlgvisual);
   break;

   case IDM_STAMPA:
     stampa_archivio();
   break;

   case IDM_HELP:
    MessageBox(hwnd,"Non implementato",nome_pacchetto,MB_OK);
   break;

   case ID_SALVA:
/* abilito i menu, carico il buffer */
    abilita_menu(hwnd,IDM_IMMISSIONE,IDM_STAMPA,
      MF_ENABLED);
    metti_buffer(rechandle);
/* salvo il record appendendo o modificando a seconda dello stato */
    if (stato==imm) er(PXRecAppend(tblhandle,rechandle));
    else er(PXRecUpdate(tblhandle,rechandle));
/* pulisco il video, disabilito i campi */
    azzera_edit(0,nfin-2);
    stato=madre;
    SetFocus(hwnd);
    abilita_tutto(FALSE);
   break;

   case ID_NOSALVA:
/* azzero e disabilito tutti i campi, inoltre abilito i menu */
    abilita_menu(hwnd,IDM_IMMISSIONE,IDM_STAMPA,
      MF_ENABLED);
    azzera_edit(0,nfin-2);
    stato=madre;
    SetFocus(hwnd);
    abilita_tutto(FALSE);
    break;

/* trattamento di default */
   default: return(DefWindowProc(hwnd,messaggio,wparam,lparam));
  }
  break;
 }
```

```
 case WM_CLOSE:
/* chiedo conferma per l'uscita */
  if(MessageBox(hwnd,"Vuoi veramente uscire ?",nome_pacchetto,
   MB_ICONQUESTION|MB_OKCANCEL|MB_DEFBUTTON2)
   ==IDOK)
  DestroyWindow(hwnd);
 break;

 case WM_PAINT:
  hdc=BeginPaint(hwnd,&ps);
/* disegno le scritte */
  scegli_font(hdc,OEM_FIXED_FONT);
  for(i=0;i<ntesti;i++)
   TextOut(hdc,testi[i].x,testi[i].y,testi[i].stringa,
    strlen(testi[i].stringa));
  EndPaint(hwnd,&ps);
  break;

 case WM_DESTROY:
/* istruzioni per terminare paradox engine e windows */
 er(PXRecBufClose(rechandle));
 er(PXTblClose(tblhandle));
 er(PXExit());
  PostQuitMessage(0);
  break;

/* trattamento di default */
 default: return(DefWindowProc(hwnd,messaggio,wparam,lparam));
 }
}


BOOL FAR PASCAL codiceproc(HWND hwnd,WORD
  messaggio,WORD wparam, LONG lparam)
{

 HFONT hfont;

 switch(messaggio)
 {
 case WM_INITDIALOG:
  hfont=GetStockObject(OEM_FIXED_FONT);
```

```
  SendDlgItemMessage(hwnd,IDD_COD,WM_SETFONT,hfont,TRUE);
  MoveWindow(GetDlgItem(hwnd,IDD_COD),XC,YC,lcod_forn*xcar+
   larghezza_edit,1*ycar*rapporto_riga,TRUE);
  return TRUE;
 case WM_COMMAND:
  switch(wparam)
  {
   case ID_OK:
    GetDlgItemText(hwnd,IDD_COD,codice,lcod_forn+1);
    aggiusta_cod_forn(codice);
    EndDialog(hwnd,0);
    return TRUE;
   case IDD_ANN:
    stato=madre;
    EndDialog(hwnd,0);
    return TRUE;
  }
 }
 return FALSE;
}
```

Si può notare che i messaggi trattati dalla funzione principale sono:

WM_CREATE, viene spedito da windows al momento della creazione della finestra;

WM_COMMAND, viene spedito quando l'utente sceglie un'opzione dal menu' o preme un bottone della maschera, in base al valore contenuto nel parametro 'wparam' del messaggio, si può capire cosa l'utente di preciso ha scelto. I valori di wparam vengono scelti dal programmatore, quando viene creato il menu', ossia nella definizione delle risorse. In questo caso, ho associato un identificatore per ogni voce del menu', come si può vedere dal listato.

WM_CLOSE, spedito da windows quando si richiede la chiusura della finestra, ad esempio tramite il menu' di sistema;

WM_PAINT, spedito da windows per ridipingere lo schermo;

WM_DESTROY, spedito da windows per la distruzione della finestra.

Da notare il fatto che i messaggi, che non vengono intercettati dalla window function, sono ridati a windows tramite la funzione '**DefWindowProc**'; in questo modo windows riserva al messaggio un trattamento di default. Il trattamento di una dialog box non differisce molto da una finestra normale, cambiano solo i tipi i messaggi spediti; in questo caso abbiamo **WM_INITDIALOG** al posto di WM_CREATE, ma per il resto WM_COMMAND a le stesse funzioni, cioè rappresenta informazioni che arrivano dalle finestre figlie della dialog box. In questa specifica dialog function, viene elaborata solamente la pressione dei due bottoni, 'ok' e 'annulla'.

3.4.3 Le risorse

Abbiamo già detto che menù, icone, cursori e dialog box sono raggruppate nel file delle risorse (RC). È importante capire il concetto che le risorse sono oggetti che vengono visualizzati sul video, l'utente compie delle azioni su questi oggetti e windows manda dei messaggi alle window function associate alle risorse. Infatti, ogni risorsa comprende la definizione degli identificatori che durante l'uso saranno, ad esempio, nel parametro 'wparam' del messaggio WM_COMMAND. Di seguito, mostriamo il file delle risorse relativo al modulo dei fornitori.

```
#include "c:\utenti\stefano\forn.h"
CHIAVI ICON    CHIAVI.ICO
rcinclude c:\utenti\stefano\about.dlg

CODICE DIALOG 60, 69, 111, 61
STYLE WS_POPUP | WS_DLGFRAME | WS_SYSMENU | 0x80L
```

```
BEGIN
    CONTROL "Inserisci Codice Fornitore:", 101, "STATIC", WS_CHILD
     | WS_VISIBLE, 8, 9, 98, 14
    PUSHBUTTON "Ok", ID_OK, 12, 43, 32, 12
    PUSHBUTTON "Annulla", IDD_ANN, 59, 43, 44, 12
    EDITTEXT IDD_COD, 24, 22, 16, 12, ES_LEFT | WS_CHILD
     | WS_VISIBLE | WS_BORDER | WS_TABSTOP
END
miomenu MENU
 {
  MENUITEM "Info..", IDM_ABOUT
  POPUP "Gestione"
  {
   MENUITEM "Immissione\tAlt+i", IDM_IMMISSIONE
   MENUITEM "Aggiornamento\tAlt+a", IDM_AGGIORNAMENTO
   MENUITEM "Cancellazione\tAlt+c", IDM_CANCELLAZIONE
   MENUITEM SEPARATOR
   MENUITEM "Uscita\tAlt+u", IDM_USCITA
  }

  POPUP "Interrogazioni"
  {
   MENUITEM "Visualizza\tAlt+v", IDM_VISUALIZZA
   MENUITEM "Stampa\tAlt+s", IDM_STAMPA
  }
  MENUITEM "Aiuto", IDM_HELP
 }

tabella ACCELERATORS
 {
  "i", IDM_IMMISSIONE,ALT
  "a", IDM_AGGIORNAMENTO,ALT
  "c", IDM_CANCELLAZIONE,ALT
  "u", IDM_USCITA,ALT
  "v", IDM_VISUALIZZA,ALT
  "s", IDM_STAMPA,ALT
 }
```

Si possono notare: la definizione della risorsa **CHIAVI** che richiama il file bitmap chiavi.ico, la risorsa **MIOMENU** che definisce il menu' associato alla finestra principale, la risorsa **TABELLA** che definisce i tasti di scelta rapida

per il menu' precedente, la dialog box **ABOUT** (about.dlg)e la dialog box **CODICE**.

Da notare soprattutto gli identificatori associati alle voci del menu', sono gli stessi che intercetta la funzione 'principale' tramite il messaggio WM_COMMAND. In questo modo, windows sa, ad esempio, che quando l'utente sceglie la voce del menu' 'immissione' deve spedire un messaggio WM_COMMAND con wparam impostato con l'identificatore IDM_IMMISSIONE. Il programmatore ha quindi un duplice compito: stabilire i collegamenti tra l'interfaccia utente, o meglio ciò che l'utente fa sulle risorse e il programma ed implementare le operazioni da compiere per ogni azione dell'utente stesso.

3.5 Dimostrazione della correttezza del programma

Dopo aver realizzato in pratica il programma, esso deve essere testato in modo accurato, prima di consegnarlo all'azienda. Cerchiamo, in questa fase, di verificare il comportamento del pacchetto come reazione ad un insieme di eventi.

Il problema è quello di trovare questo insieme. Questo non è certo banale, sappiamo tutti che non si può mai dimostrare, con certezza assoluta, la correttezza di un software; in quanto l'insieme degli eventi, o delle prove, dovrebbe comprendere tutte le possibilità esistenti. La soluzione adottata, è stata quella di scegliere un insieme finito di eventi, che sia il più rappresentativo possibile; ossia comprenda le maggiori tipologie di situazioni in cui l'utente può trovarsi.

Per fare questo, ho dovuto compiere un notevole sforzo mentale dal punto di vista della fantasia e dell'inventiva; questo per immedesimarmi nell'utente inesperto che utilizza il programma.
Sono state fatte una serie di prove, per così dire, normali, che non hanno creato particolari problemi; un esempio di queste prove è elencato di seguito:

partenza del programma senza la presenza dell'archivio;
scelta delle varie voci di menù;
spostamento tra i campi della maschera tramite la tastiera;
riutilizzo del programma dopo una modifica alla finestra (movimento, ridimensionamento) o dopo l'utilizzo contemporaneo di altre applicazioni;
effettivo salvataggio e caricamento dei dati;
immissione errata di ogni singolo campo;
aggiornamento, cancellazione di record non esistenti;
immissione di record già presenti (stessa chiave primaria) o con l'assenza della chiave;
effettivo slittamento dei fornitori nel modulo dei ricambi.

Da queste prove si sono avuti risvolti interessanti, non previsti in un primo tempo, ad esempio si è osservato che in fase di aggiornamento, si può modificare il campo chiave, senza che il programma rilevi la delicata operazione. In realtà, il programma sostituisce il campo e modifica l'indice di chiave primaria sul campo in questione.

Dopo queste prove normali, si sono cercati dei casi notevoli, cioè delle situazioni ai limiti delle possibilità del pacchetto, che possono mettere in crisi il programma. Per fare questo, occorrerebbe lavorare su una certa mole di dati, che io non ho a disposizione, per cui i test su questi casi notevoli sono sicuramente meno affidabili.
Di seguito elenco alcuni casi notevoli, regolarmente testati, relativi soprattutto alle note, alle liste di ricambi e alla distinta base, che sono state le implementazioni più delicate, dal punto di vista informatico.

inserire un numero di righe di note che superi la memoria disponibile, oppure inserire una lista di ricambi che superi la memoria, in questo caso il programma visualizza un messaggio di errore e non permette altri inserimenti;
presenza di un ciclo tra i legami di distinta base durante una operazione di esplosione riepilogativa, in questo caso il programma non fa nessun controllo sulla presenza del ciclo in se stesso, ma controlla che i livelli esplorati non superano il numero di nove, ritenuto livello di difficile raggiungimento. Se si superano i nove livelli, viene visualizzato un messaggio che indica la presenza di un probabile ciclo; la sua scoperta e la sua correzione è a carico dell'utente.

CAPITOLO 4
Documentazione

4.1 Manuale utente

Inseriamo ora il manuale utente, così come consegnato all'azienda, quindi la numerazione dei capitoli, in questa sezione, non seguirà più quella dell'intera tesi.

CAPITOLO 1 - Introduzione al pacchetto

1.1 Di cosa si occupa il pacchetto

Il pacchetto permette ad un'azienda di gestire, in modo semplice ed efficace, tutte le attività interne all'azienda stessa. Questo significa che esso si occupa di tutto ciò che ruota intorno al problema di come manutenere i cespiti, il pacchetto gestisce le manutenzioni ordinarie e straordinarie che l'azienda deve compiere, con la possibilità di riferirsi a fornitori di servizi esterni.
Il pacchetto prevede la memorizzazione e la gestione dei seguenti archivi:

Richieste di lavoro, una traccia storica dei lavori di manutenzione.
Fornitori, l'elenco dei fornitori di servizi di manutenzione all'azienda.
Ricambi, l'elenco dei ricambi che l'azienda utilizza nei lavori di manutenzione.
Strutture, l'elenco dei soggetti (macchine, impianti, strumenti) da manutenere.
Calendari, una serie di calendari di fabbrica che si riferiscono al lavoro di manutenzione sulle strutture.
Distinta base, una piccola distinta base che mette in relazione le strutture e i ricambi.

1.2 Installazione del pacchetto

Il pacchetto è stato sviluppato per l'ambiente Microsoft Windows, quindi l'installazione si effettua nel medesimo ambiente.
Seguire questi passi per l'installazione:

- Dal sistema operativo DOS, spostarsi sul drive del disco di installazione, ad esempio A:.
- Creare una directory nel disco dove si vuole installare il pacchetto, ad esempio md c:\manut.
- Copiare i file del pacchetto nella directory creata, ad esempio copy *.* c:\manut.
- Spostarsi nel disco c, ad esempio C:
- Digitare win per entrare in ambiente windows.
- Selezionare dal menù 'file' del program manager, la voce 'nuovo'.
- Scegliere 'gruppo di programmi', premere il bottone OK.
- Inserire la descrizione desiderata, è consigliato "Manutenzioni", inserire il nome del file del gruppo presente nel disco installazione "manut.grp".

A questo punto apparirà il gruppo con tutte le icone del pacchetto. Assicurarsi di uscire da windows con l'opzione 'salva impostazioni' attiva, altrimenti il gruppo non verrà memorizzato.

CAPITOLO 2 - Modulo dei fornitori

Per eseguire il modulo fornitori, effettuare un'operazione di click sull'icona "fornitori". A questo punto appare una finestra, a tutto schermo, che presenta una barra con il titolo della finestra, un menù e un'area con una maschera. Vediamo ora come si utilizzano questi strumenti. Accanto al titolo sono presenti i tre bottoni classici di windows: per minimizzare, massimizzare e il menù di sistema. Poi c'è il menù, con le seguenti voci:

info
immissione
aggiornamento
cancellazione
uscita
visualizzazione
stampa

Si può accedere al menù effettuando un'operazione di click sul mouse o premendo F10 o ALT. L'altra possibilità, offerta a chi deve compiere un'operazione del menù, è usare i tasti di accelerazione; ogni voce, a cui è associato un tasto, visualizza alla propria destra la combinazione di tasti che realizzano la stessa funzione (in genere Alt+carattere). Vediamo come comportarsi in ogni situazione:

❑ immissione

Scegliendo questa opzione, si abilita il menù e si può iniziare a inserire i dati. Vedere la sezione che riguarda l'utilizzo della maschera, per capire come inserire i dati.

❑ aggiornamento

Anche questa opzione abilita la maschera, ma prima, chiede il codice del fornitore, attraverso un'apposita finestra. Si può inserire il codice, che deve essere numerico e si possono tralasciare eventuali zeri davanti al codice stesso; premendo OK si conferma l'aggiornamento, scegliendo ANNULLA si torna alla situazione iniziale.

❑ cancellazione

Anche in questo caso viene richiesto il codice del fornitore, come in aggiornamento; vengono visualizzati i dati sulla maschera e si chiede la conferma per la cancellazione, premendo OK si può effettivamente cancellare il record.

❑ uscita

esce dal programma, chiedendo comunque conferma prima, premendo OK si esce effettivamente dal programma.

❑ visualizzazione

visualizza una lista con tutti i record dell'archivio, con il mouse o con le frecce è possibile scorrere la lista, oppure stampare (bottone STAMPA) o ritornare all'inizio (ANNULLA). Se la lista è troppo lunga per essere tenuta totalmente in memoria, sarà abilitato un bottone che permette di visualizzare il resto dei record.

❑ Stampa

Stampa l'intero archivio.

❑ utilizzo della maschera

È possibile, con il mouse, accedere direttamente a ogni campo della maschera, semplicemente effettuando un click sul campo desiderato, dopodiché si può scrivere l'informazione desiderata. È possibile usare la tastiera, in questo caso con il pulsante INVIO ci si può spostare al campo successivo in forma ciclica, quindi il prossimo campo dell'ultimo è il primo. Con il pulsante TAB si può accedere ai bottoni, in questo caso con lo stesso TAB si continua a spostarsi tra i bottoni, mentre con l'INVIO si preme il bottone stesso. Per chiarire meglio il concetto, diciamo che ci sono due gruppi di oggetti: i campi e i bottoni, con il TAB si passa da un gruppo ad un altro, mentre, con l'INVIO si accede in sequenza ai campi di immissione.
Per quanto riguarda le finestre di dialogo, quelle finestre più piccole, che appaiono solo per inserire pochi dati, si può accedere agli elementi direttamente con il mouse o spostarsi col tasto TAB.

Ricordiamo che mentre si stanno inserendo delle informazioni nella maschera, non si può accedere al menù, l'unico modo di accedere al menù è rappresentato dall'uscita senza salvataggio.

❑ Salvataggio

Sulla maschera sono presenti due bottoni, 'esci e salva' ed 'esci senza salvare'. Premendo il secondo bottone, si perde tutto ciò che è presente in quel momento nella maschera, mentre per il primo bottone il contenuto viene salvato nel record, sia in aggiornamento che in immissione.

CAPITOLO 3 - Modulo dei ricambi

Questo modulo ha un funzionamento identico al precedente, adesso mettiamo in evidenza solamente le differenze. Innanzi tutto la maschera comprende una serie di informazioni che non possono essere modificate, precisamente i due più vecchi fornitori; in secondo luogo al momento del salvataggio vengono fatti una serie di controlli:
viene controllato il codice del ricambio, che deve essere formattato nel seguente modo: xx.xxxxx.xx.xxxx; questo significa che devono essere inseriti i punti nella giusta posizione; inoltre viene controllata la data, che sia una data corretta, senza nessun carattere di separazione tra i numeri e il costo che deve essere numerico. Nel caso che un campo sia errato, verrà visualizzata una finestra che specifica il nome del campo in questione.

Infine ricordiamo che basta una modifica a un elemento del fornitore ultimo, per ottenere uno spostamento del medesimo nella posizione successiva (fornitore 1); il risultato si può vedere in un successivo aggiornamento.

CAPITOLO 4 - Modulo delle strutture

Anche in questo modulo le operazioni sono simili, in aggiunta si ha una ricerca per intervallo di data, la gestione di un gruppo di radio button e un editor per le note.
Qui la gestione della maschera è un po' più complessa; si può accedere ai campi di immissione attraverso il tasto INVIO, con il tasto TAB si passa al secondo gruppo di elementi. Quando si è posizionati su un radio button, premendo la barra si può attivare o disattivare il bottone, i due bottoni si escludono a vicenda. Il secondo gruppo di oggetti comprende i bottoni e la finestra di edit delle note; quando si è posizionati sulle note, si possono inserire le note senza preoccuparsi di andare a capo (gestito automaticamente), con il pulsante TAB ci si può spostare al campo successivo.

Se si sceglie l'interrogazione per intervallo di data, si presenta una finestra di dialogo. In questo caso bisogna inserire la data di inizio e di fine, premere OK; successivamente, se le date sono giuste, viene visualizzata la solita finestra di visualizzazione.

Una particolarità di questo modulo è rappresentata dalla richiesta di associare una struttura a un calendario; infatti, in fase di salvataggio, il programma richiede di inserire il nome del calendario. Appare una dialog box che, dapprima propone il nome del calendario standard, premendo il bottone OK si conferma quel calendario, altrimenti si può comunque modificare questo nome a piacere.

CAPITOLO 5 - Modulo Richieste di lavoro

Questo modulo è il più complesso, presenta una maschera più completa e una serie di ricerche su diversi campi.

❑ Gestione maschera

La maschera presenta due gruppi, nel primo gruppo ci si può spostare tra i campi tramite il tasto INVIO. Quando si entra all'interno di un gruppo di scelte che si autoescludono, si può scegliere il bottone giusto attraverso le frecce direzionali, dopodiché con l'INVIO si esce dal gruppo.

Quando si entra su una listbox si possono selezionare gli elementi tramite le frecce (o con il mouse sulla scrollbar); una volta selezionato un elemento, con il tasto INVIO si apre una finestra di dialogo che permette la modifica diretta dell'elemento selezionato. All'interno di questa finestra di dialogo è possibile modificare il codice ricambio e premere OK, altrimenti è possibile cancellarlo con il bottone CANCELLA; con il bottone ANNULLA si ignora qualsiasi operazione.

Per inserire un nuovo ricambio bisogna selezionare l'elemento <nuovo ric> e premere INVIO (o fare un doppio click). Attenzione che l'unico modo per uscire da una listbox è premere il tasto TAB (oltre al solito spostamento con il mouse).

Anche in questo modulo sono gestite le note, come nel modulo delle strutture. Il secondo gruppo di oggetti comprende, oltre ai soliti bottoni, anche le note e la seconda listbox (ricambi consuntivi).

❑ Ricerche diverse

Sono presenti, come voce nel menù, le seguenti ricerche: per codice struttura, per reparto e per intervallo di data; oltre alla visualizzazione e alla stampa completa.

Richiamando queste voci, appare una finestra di dialogo che chiede di inserire la chiave di ricerca, in particolare il codice struttura o il reparto o le due date, e presenta una serie di bottoni, in cui è possibile scegliere i tipi di record da visualizzare. Una volta selezionato il bottone, con la barra spaziatrice si può cambiare l'impostazione; infine premendo OK si conferma la scelta.

La stessa cosa vale per la visualizzazione di tutto l'archivio. Se non si vogliono filtrare i tipi di record, basta lasciare tutto come impostato alla presentazione della maschera.

CAPITOLO 6 - Modulo dei calendari

Fino ad ora abbiamo visto dei moduli molto simili fra di loro, ora ci sono due moduli leggermente diversi: i calendari e la distinta base. Le operazioni presenti nel menù sono: apri calendario, nuovo calendario, cancella calendario, visualizza lista dei calendari e stampa calendario. La maschera è leggermente diversa dalle altre.

❑ Gestione della maschera

La maschera presenta due parti, una parte riguarda i 31 giorni del mese, in cui è possibile inserire un carattere, tra queste caselle ci si sposta mediante i tasti direzionali; la seconda parte riguarda le operazioni di salvataggio, di spostamento tra i mesi e di copia da un altro calendario. Con il tasto TAB si passa tra i due diversi gruppi di oggetti presenti sullo schermo. È visualizzata sullo schermo una legenda dei possibili caratteri da immettere.

❑ Operazioni

Quando si richiede di creare un nuovo calendario, viene chiesto, mediante una finestra di dialogo, di immettere il nome del nuovo calendario e l'anno al quale il medesimo calendario si riferisce, premendo OK si conferma la scelta fatta.

Quando si chiede, invece, l'apertura di un calendario già esistente, viene richiesto semplicemente il nome del calendario. In ambedue i casi viene visualizzato il mese numero 1.

Quando si sceglie la lista calendari, viene creata la solita finestra con la visualizzazione di tutti i nomi dei calendari, con la possibilità di stampare (bottone STAMPA).

Infine scegliendo la voce di menù 'stampa' si ha una stampa del calendario corrente.

CAPITOLO 7 - Modulo della distinta base

Anche questo modulo è un po' originale, presenta una maschera composta da due campi di immissione, una lista e un bottone.
Le operazioni sono: creazione legame, cancellazione legame, esplosione e implosione.

❑ Operazioni

Scegliendo l'opzione crea legame, viene chiesto il codice padre e il codice figlio, viene controllata immediatamente la loro esistenza, infatti appare la descrizione sulla destra del codice inserito. Dopo viene richiesta la conferma della creazione e viene chiesto se si vuole creare un altro legame sotto la stessa struttura padre. Rispondendo di si, viene data la possibilità di inserire un altro codice figlio.
Da notare che è possibile creare legami tra strutture e ricambi e legami tra ricambi e ricambi, o meglio sottoricambi.
Scegliendo l'opzione cancella legame, è possibile inserire il codice padre, apparirà così una lista dei figli, selezionando un figlio con i tasti direzionali o con il mouse è possibile cancellarlo, basta premere INVIO o fare un doppio click, dopodiché si chiede la conferma della cancellazione.
Scegliendo l'opzione di esplosione o implosione viene chiesto rispettivamente il codice padre o il codice figlio, dopodiché si ottiene la lista dei figli o dei padri del primo livello. A questo punto premendo INVIO, o facendo un doppio click su di un elemento, è possibile avere tutte (o quasi) le informazioni sulla struttura o sul ricambio.

❑ Gestione maschera

La maschera non permette molta libertà all'utente, quindi ogni volta ci si trova con il cursore sul campo da riempire. È comunque possibile, in qualsiasi momento, annullare ciò che si sta facendo tramite il bottone ESCI o con il tasto ESC.

All'interno della listbox è possibile spostarsi tra gli elementi con i tasti direzionali e compiere una operazione su di un elemento tramite il tasto INVIO o il doppio click.

CAPITOLO 8 - Stampare

Quando si richiede un'operazione di stampa, l'utente non si deve preoccupare di stabilire il tipo di stampante, in quanto, il programma è in grado di capire quale stampante è in quel momento impostata nell'ambiente windows. Quindi è compito dell'utilizzatore assicurarsi che sia impostata la stampante corretta. Se non sia ha la minima idea di che stampante sia impostata, consultare il programma, a corredo del sistema windows, PANNELLO DI CONTROLLO nel gruppo principale. Scegliendo STAMPANTI è possibile scegliere la propria stampante. Una volta scelta la stampante premere i bottoni IMPOSTA COME STAMPANTE PREDEFINITA e poi CHIUDI. A questo punto il programma stamperà sulla stampante prescelta.

APPENDICE - Messaggi di errore

Ci sono due tipi di messaggi di errore nel programma: quelli che derivano dalle operazioni sugli archivi, per esempio l'impossibilità di aprire un archivio, e quelli che riguardano l'utilizzo del pacchetto, ad esempio un campo errato. I primi sono in inglese, mentre i secondi sono in italiano. Vediamo un elenco di ambedue i messaggi:

❑ Messaggi che riguardano l'utilizzo del pacchetto

"Applicazione già in esecuzione";

questo messaggio appare quando si cerca di eseguire un modulo già in esecuzione.

"Errore nella stampante";

questo messaggio appare quando si tenta di stampare e non esiste il driver per la stampante, oppure la stampante non è collegata.

"Creazione nuovo archivio";

messaggio che indica che l'archivio non esiste e viene ricreato.

"Errore nel campo";

questo messaggio appare se è presente un errore nel campo di immissione in una dialog box, da questo messaggio derivano i prossimi 6 messaggi:

"Errore nella data iniziale", "Errore nella data finale", "Errore nel codice", "Errore nel campo qta'", "Errore nel nome calendario", "Errore nel campo anno".

Messaggi che riguardano le note:
"*Non e' possibile scrivere più di 2 pagine*" quando si inseriscono troppe note, in questo caso bisogna cancellare le ultime lettere inserite;
"*Le note hanno superato il limite di memoria, salvate solo 40 righe*" se sono state immesse più di 40 righe;
"*L'archivio delle note non esiste più creato nuovo archivio*".

Gli stessi messaggi delle note esistono per le liste di ricambi del modulo delle richieste di lavoro: "*L'archivio dei ricambi non esiste più, creato nuovo archivio*", "*I ricambi hanno superato il limite di memoria salvati solo 80 codici*".

Per quanto riguarda il modulo dei calendari, si possono incontrare questi messaggi di errore:
"*Calendario non esistente, usare 'Nuovo'*" se si cerca di aprire un calendario che non esistente, "*Calendario già esistente, usare 'Apri'*" se si cerca di creare un calendario già esistente, "*Calendario non esistente*" se il calendario richiesto non esiste.
"*Non e' possibile cancellare il calendario standard*" quando si cerca di cancellare il calendario standard, il che non è permesso.
"*Carattere non valido nella casella evidenziata*", è stato immesso un carattere non valido nel calendario, vedere la legenda sullo schermo.
"*Calendario associato non esistente, creare il calendario, il record salvato lo stesso*", nel modulo delle strutture si è associato un calendario non esistente, cambiare l'associazione o creare il calendario, altrimenti ci potrebbero essere problemi in futuro.

Infine, per quanto riguarda il modulo della distinta base:
"*Codice non esistente*" se il codice del ricambio o della struttura non esiste, "*Legame già esistente*" se si inserisce un legame che esiste già, "*Legami non presenti*" se dopo un'operazione di implosione o esplosione non si ottiene nessun elemento, "*Ci sono legami con codici non esistenti. Il legame viene ignorato*" se durante un'operazione di implosione o esplosione si trovano codici non esistenti, "*Legame non trovato*" se un legame non esiste.

❑ Messaggi che riguardano le operazioni sugli archivi

Questi messaggi mettono in evidenza problemi con il disco o con i file contenenti i dati:
"*Drive not ready*", "*Directory not found*", "*No access to directory*", "*Could not find file*", "*Disk is write protected*", "*Not enough disk space to complete operation*", "*Invalid file name*", "*Invalid directory name*", "*General hardware error*", "*Previous fatal error; cannot proceed*".

Questi messaggi riguardano problemi durante l'inizializzazione del sistema, in questo caso ci potrebbero essere grossi problemi nel seguito del lavoro:
"*Engine not initialized*", "*Engine already initialized*", "*Engine not initialized* with *PXNetInit*" se si cerca di lavorare in rete.

Questi messaggi riguardano problemi con gli indici:
"*Primary index is corrupted*", "*Primary index is out of date*", "*Sort for index different from table*", "*Secondary index is out of date*", "*Secondary index was not found*", "*Secondary index is corrupted*", "*Secondary index already open*"
"*Record too big for index*", "*Invalid sort order table*".

Questi messaggi riguardano le tabelle:
"*Table is corrupted*", "*Invalid table handle*", "*Table structures are different*", "*Unable to perform operation on open table*", "*Table IS indexed*", "*Table is NOT indexed*", "*Table name is invalid*", "*End of table*", "*Start of table*", "*Table is busy*" se si sta eseguendo altre applicazioni che accedono a tabelle già in uso, "*Table was not found*", "*Table is full*", "*Can't modify table open on non-maintained secondary index*", "*Operation on empty table*" se la tabella è vuota, "*No more table handles available*".

Questi messaggi riguardano i record:
"*Record was not found*" se il record non esiste, "*Invalid record handle*" se si sta riferendo a un record non valido, "*No more record handles available*", "*Invalid field name*" se si sta riferendo a un campo non valido, "*No more file handles available*", "*Data type mismatch*" se il dato con cui si lavora non è compatibile con il record dell'archivio, "*Invalid date given*" se la data inserita non è valida, "*Argument is out of range*", "*Invalid argument*", "*Not enough memory to complete operation*", "*Key violation*" se si cerca di inserire un record con chiave primaria che esiste già, "*Buffer is too small for result*", "*Not enough stack space to complete operation*", "*Not enough swap buffer space to complete operation*".

Questi messaggi riguardano l'utilizzo in ambiente windows:
"*Too many clients for the Engine DLL*", "*Exceeds limits specified in WIN.INI*", "*Don't run Engine in Windows real mode*".

4.2 Documentazione per programmatori

Questa sezione, in netto contrasto con quella precedente, è scritta per esperti di informatica, in questa sede intendo dare la possibilità, a un programmatore, di capire in dettaglio come è stato scritto il programma, quindi si cerca di analizzare i sorgenti del pacchetto.
Iniziamo la documentazione con una parte introduttiva sulla struttura dei sorgenti, infatti tutti i moduli hanno i sorgenti organizzati nello stesso modo. Successivamente, vedremo alcune considerazione generali su come sono organizzati i moduli, dal punto di vista del codice scritto, puntando l'obiettivo su l'implementazione delle operazioni sugli archivi, sull'interfaccia grafica e sulle risorse di windows.

4.2.1 Organizzazione dei sorgenti

I sorgenti sono divisi in **sezioni**, questo per rendere più semplice la manutenzione del software in una successiva fase. Spieghiamo in dettaglio ogni sezione.

➔ Sezione files di inclusione

I programmatori **'c'** capiranno benissimo perché questa sezione è la prima della lista, infatti in 'c' è indispensabile includere gli *header files*, ossia i files con i prototipi delle funzioni delle librerie che si utilizzano.
Io ho sempre incluso, oltre alle librerie del 'c' standard, anche *windows.h*, che serve per usare le funzioni di windows e *pxengine.h*, file di inclusione, fornito

con paradox engine, che permette l'utilizzo delle funzioni di paradox engine stesso.

Inoltre, per ogni modulo, ho un file di inclusione personale, con lo stesso nome del modulo, contenente una serie di costanti, utilizzate per lo più per gli identificatori delle finestre di windows, per i campi delle dialog box e per gli identificatori all'interno dei messaggi.

➔ Sezione delle costanti

In questa sezione sono raccolte tutte le macro (#*define*) del programma, che nel mio caso hanno esclusivamente funzione di costanti.

Sono presenti costanti che mi definiscono le posizioni a video dei vari campi (*col1,col2,riga1,riga2* ecc...); le lunghezze dei campi, lunghezze che servono sia per definire quanto deve essere lungo il campo a video e sia per definire la lunghezza del campo nel record; infine gli stati in cui si può trovare l'elaborazione (immissione, aggiornamento, visualizzazione ecc...)

In tutti i moduli sono definite : *nfin* per il numero di finestre della maschera, *ncampi* per il numero dei campi del record, *ntesti* per il numero dei testi della maschera, *lriga_listbox* per la larghezza della listbox, *ltotale* per la larghezza della riga di stampa, *linterlinea* per lo spazio tra le righe di stampa.

Non dimentichiamo il nome dell'applicazione, come titolo della finestra madre, il nome del pacchetto da visualizzare in ogni message box (secondo lo standard CUA) e una costante '*punto*', che serve per controllare il carattere di separazione dei codici formattati (ricambi e strutture).

➔ Sezione dei tipi

In questa sezione vengono definiti i tipi utilizzati dal programma (istruzione *typedef*); sono sempre presenti i seguenti tipi:

- un tipo record (struct in 'c') con tutti campi del record dell'archivio;
- un tipo struct con una struttura contenente le informazioni per la creazione delle finestre, in tutti i moduli esso è il seguente:

```
typedef struct
{
 LPSTR nome_classe,nome_finestra;
 DWORD stile;
 int x,y,larg,altez;
 HWND hfin_padre;
 HMENU hmenu;
 HANDLE hinstance;
 LPSTR lpparam;
} caratteristica_edit;
```

- un tipo struct contenente le informazioni per scrivere una stringa a video, in tutti i moduli è il seguente:

```
typedef struct
{
 int x,y;
 LPSTR stringa;
} caratteristica_testo;
```

➔ **Sezione dei prototipi**

Questa sezione comprende l'elenco di tutte le funzioni del sorgente, raggruppate in questo punto come prevede il 'C' per poter controllare eventuali errori di passaggio dei parametri.

➔ **Sezione variabili globali**

Qui sono definite tutte le variabili globali del programma, esse giocano un ruolo importante per chi programma sotto windows. Dico questo, perché in windows l'unico modo di passare parametri a una window function è la dichiarazione della variabile come globale, infatti non c'è passaggio diretto di parametri.

In ogni modulo sono definite queste variabili globali:

- due variabili che indicano la larghezza e l'altezza della font da me scelta per la applicazione, queste servono per decidere la grandezza in pixel dei campi (*int xcar,ycar*);
- un array contenente i puntatori delle window function predefinite delle finestre, figlie della finestra principale, questi servono per mandare i messaggi, che non vengono elaborati da me, alle legittime funzioni (*FARPROC procvecchie[nfin]*);
- il puntatore (handle) all'istanza corrente dell'applicazione (*HANDLE hinst*);
- gli handle alle finestre figlie (*HANDLE h[nfin]*);
- l'handle della finestra principale (*HWND hwnd*);
- l'handle della tabella dei tasti di accelerazione (*HANDLE hacc*);
- l'handle del buffer di paradox engine (*RECORDHANDLE rechandle*);
- l'handle della tabella di paradox engine (*TABLEHANDLE tblhandle*);
- gli array contenenti i nomi dei campi e i tipi per la creazione della tabella (*static char *campi[ncampi]={...} e static char *tipi[ncampi]={...}*);
- il nome della tabella (*static char nome_tabella[12]={...}*);
- alcune variabili per i dati inseriti mediante dialog box;

- un'array di elementi di tipo caratteristica_edit, opportunamente inizializzati per creare le finestre figlie, ossia i campi della maschera (*caratteristica_edit figlie[nfin]={...}*);
- un'array di elementi di tipo caratteristica_testi, opportunamente inizializzati per visualizzare i testi a video, ossia i campi di output (*caratteristica_testi[ntesti]={...}*).
- un puntatore alla funzione elabora_tab che filtra i messaggi delle edit control; viene assegnato alle finestre al momento della loro creazione (*FARPROC puntatore_elabora_tab).*

➔ Sezione routine varie

Questa sezione comprende un file di inclusione contenente una serie di routine comuni a tutti i moduli, questo file è variwin.inc è verrà spiegato in dettaglio successivamente.

➔ Sezione winmain

Questa sezione contiene il *winmain*, che è l'unico punto di entrata del programma. Il suo compito, vitale per l'applicazione, è quello di registrare la presenza della nuova finestra a windows, creare la finestra madre e tutte quelle figlie ed infine attivare il message loop (vedi capitolo 2).

➔ Sezione di windows, dialog e subclassing function

Questa sezione contiene tutte le funzioni associate alle finestre dell'applicativo, tutte le funzioni associate alle dialog box e tutte le funzioni che svolgono filtro (*subclassing*) sulle finestre figlie.

Chiaramente, la prima di queste funzioni è quella principale, che ha il compito di elaborare i messaggi diretti alla finestra madre. Questa funzione è formata da un'istruzione di condizionamento multiplo (*switch*) sul messaggio, questa istruzione permette di indirizzare l'elaborazione su una delle innumerevoli funzioni presenti, quali quelle del menù (immissione, aggiornamento ecc..), a volte quelle dai campi della maschera, ad esempio aggiungere o togliere elementi da una list box.
Le dialog functions seguenti sono uniche per tutti i moduli, le prime due gestiscono, rispettivamente, la dialog box di informazione e quella di visualizzazione di un archivio; la terza è una funzione che effettua subclassing :

BOOL FAR PASCAL aboutproc(HWND,WORD,WORD,LONG);
ha solamente il compito di visualizzare la dialog box, trasferire il focus al bottone OK e attendere la sua pressione per eliminare la dialog box.
BOOL FAR PASCAL visualproc(HWND,WORD,WORD,LONG);
dialog function che gestisce la dialogbox VISUAL, essa ha il compito di fare una lettura sequenziale dell'archivio e visualizzare ogni record in una listbox. Si può uscire dalla finestra premendo il bottone annulla o stampando il contenuto della listbox.

long FAR PASCAL elabora_tab(HWND,WORD,WORD,LONG);
filtra i messaggi che arrivano ai campi di edit, intercetta quelli che riguardano tasti premuti sulla tastiera e sposta il focus, implementando così l'utilizzo della tastiera da parte dell'utente.

➔ Sezione procedure varie

Questa sezione contiene tutte le funzioni che sono richiamate in modo classico, cioè mediante salto a subroutine. Queste derivano dalla suddivisione in moduli delle sequenze di codice, all'interno delle scelte "*case*" delle window function. In questa sezione sono presenti le funzioni che registrano la classe, le funzioni che creano le finestre e quelle che controllano gli errori.

Vediamo meglio le funzioni comuni a quasi ogni modulo:

BOOL initapplication(HANDLE);

registra la classe della finestra, associandogli l'icona e il menù, restituisce true se tutto è andato bene, false altrimenti.

BOOL initinstance(HANDLE,int);

crea la finestra principale, crea tutte le finestre figlie, dimensionandole a seconda delle lunghezze definite in caratteristica_edit.

void mostra_errore(int);

mostra in una messagebox un messaggio indicante la presenza di un errore in un campo sullo schermo. Per far capire all'utente il campo errato si visualizza il testo che appare sullo schermo vicino al campo in questione.

void stampa_listbox(HWND);

prende gli elementi dalla list box di visualizzazione e li stampa sulla stampante impostata in windows.

*int numerico(char *);*

controlla se una stringa è numerica, restituisce true se è numerica o nulla, false altrimenti.

*int formattato(char *);*

controlla se la stringa è formattata tramite i punti, come richiesto dalla codifica standard.

*int form_data(char *);*

controlla che la stringa in ingresso sia numerica e sia una data, restituisce true in caso affermativo, false in caso contrario.

➔ Sezione funzioni che riguardano paradox engine

Questa sezione contiene tutte le funzioni che utilizzano gli archivi di paradox engine direttamente, quindi funzioni di creazione delle tabelle, di salvataggio e caricamento dati ecc...

4.2.2 Trattamento archivi

Vediamo come sono state da me utilizzate le funzioni di paradox engine per gestire gli archivi. Abbiamo già accennato all'utilizzo, da parte di paradox engine, di un buffer in memoria; io ho utilizzato questo buffer, in quanto obbligatorio, ma ho sempre utilizzato un record, definito come struttura globale, contenente i dati del record appena letto o scritto. Quindi si può affermare che la mia applicazione interagisce con gli archivi attraverso **tre** livelli: lo schermo, *il* record in memoria e il buffer.

In tutti i moduli si può osservare che sono presenti una serie di funzioni che implementano questa stratificazione; precisamente queste funzioni:

- *schermo_record* per leggere i campi sullo schermo (finestre figlie) e metterli nel record da me definito in memoria;

- *record_buffer* per leggere i dati nel record e metterli nel buffer, una volta nel buffer, con una sola istruzione di PXRecAppend o PXRecUpdate possono essere salvati nel file;
- *buffer_record* per prelevare dal buffer i dati e salvarli nel record in memoria;
- *record_schermo* per scrivere sullo schermo i dati del record in memoria.

Vediamo come vengono utilizzate queste funzioni: al momento del caricamento dei dati, per l'aggiornamento ad esempio, viene cercato il record, caricato nel buffer, poi nel record in memoria e visualizzato a video.
Al momento del salvataggio vengono letti i dati sul video, salvati nel record in memoria, passati nel buffer e salvati nel file, tutto questo se non ci sono errori nei dati immessi; infatti ogni modulo prima di salvare controlla i dati inseriti e informa l'utente se qualche informazione inserita ha un formato non accettabile, ad esempio una data non numerica.
L'unico modulo che salta uno di questi strati è quello dei fornitori, che passa le informazioni direttamente dallo schermo al buffer di paradox engine (in quanto non fa nessun controllo sui dati).

In tutti i moduli si è seguita la medesima politica di gestione delle tabelle, infatti ogni modulo apre la propria tabella all'inizio dell'applicazione (ricevimento del messaggio WM_CREATE), compie tutte le operazione durante l'esecuzione del programma e la chiude alla fine (ricevimento del messaggio WM_DESTROY). Per questo fatto, il pacchetto non prevede l'esecuzione di più istanze o l'esecuzione contemporanea di diversi moduli che condividono stesse tabelle.

Infine, aggiungiamo che in tutti i moduli, la prima operazione sulla tabella è la verifica della sua esistenza, in caso di risposta negativa, ogni modulo provvede alla creazione, informando l'utente tramite un messaggio sullo schermo.

4.2.3 Interfaccia grafica

Anche per quanto riguarda l'interfaccia grafica, i moduli sono strutturati nello stesso modo, come già dimostrato quando abbiamo parlato della modularizzazione del problema. Sono definite una serie di finestre figlie (campi di edit, bottoni, list box, radio button e semplici testi), alle quali è associato un identificatore di riferimento. Quindi, le funzioni che leggono i dati sullo schermo non fanno altro che chiedere il testo corrente di ogni finestra di edit, tramite i propri identificativi.

Tutti i moduli hanno un file sorgente, che riguarda la definizione delle risorse della applicazione (*file.rc*); in questi file sono definite queste risorse:

- l'icona dell'applicazione;
- le dialog box utilizzate dall'applicazione;
- la tabella per i tasti di scelta rapida, per accedere velocemente alle opzioni dei menù;
- i menù associati alla finestra madre dell'applicativo, con la dichiarazione degli identificatori, per capire così, la scelta compiuta dall'utente nel menù.

Concludiamo il discorso relativo all'interfaccia grafica parlando dello *stato* in cui si può trovare l'applicazione. Ho implementato un meccanismo di stato

per capire, in qualsiasi momento, l'utente cosa sta facendo. Infatti, sviluppando in ambiente windows non si controllano direttamente le operazioni dell'utente, quindi non si può sapere se chi immette una stringa nel campo numero 3, sta inserendo un nuovo record o ne sta modificando uno già esistente. Per questo, ogni modulo ha una variabile globale "*stato*", che ha il compito di conservare un identificatore che indichi quale operazione sta eseguendo l'utente.

4.2.4 Descrizione di variwin.inc

Abbiamo detto che è presente un file di inclusione di nome variwin.inc contenente una serie di funzioni comuni a tutti i moduli; ecco una spiegazione del comportamento di ogni sua funzione.
Sono definite due costanti: la larghezza in pixel che richiede il margine del testo in una finestra edit rispetto alla cornice e il rapporto tra l'altezza della finestra di edit e il testo interno alla finestra stessa. Ecco l'elenco delle funzioni contenute in variwin.h:

dim_larg(device context); e dim_altez(device context);
restituiscono la larghezza e l'altezza della font associata al device context in ingresso. Servono per dimensionare le finestre di edit, ossia i campi della maschera.
assegna_font_a_finestra(handle finestra,oggetto font);
carica la font specificata e spedisce un messaggio alla finestra per settarla con quella font, viene utilizzata per assegnare una font fissa ai campi della maschera.

scegli_font(device context,oggetto font);

carica la font specificata e la associa a quel device context, viene utilizzata per settare la font e scrivere stringhe sul device context.

abilita_tutto(flag);

abilita o disabilita, a seconda del flag, i campi della maschera.

er(numero errore);

visualizza sullo schermo una finestra di messaggio (message box), contenente il messaggio di errore di paradox engine di numero dato. Questa funzione fa da filtro a ogni chiamata a funzioni di paradox engine, all'interno dei moduli; in questo modo se ci saranno errori non previsti, per lo meno si vedranno sullo schermo dei messaggi di errori.

azzera_edit(identificatore iniziale, identificatore finale);

spedisce una stringa vuota a ogni finestra di edit della maschera, esattamente nell'intervallo stabilito dagli identificatori.

abilita_menu(finestra,inizio,fine,flag);

abilita o disabilita, a seconda del flag, le voci del menu della finestra data, precisamente le voci con identificatori compresi tra inizio e fine.

ridipingi_tutto(finestra);

invalida tutto il rettangolo della finestra madre e spedisce un messaggio di WM_PAINT, in questo modo costringe l'applicazione a ridisegnare tutte le stringhe sullo schermo.

crea_dc_stampante_default()

restituisce un device context associato alla stampante impostata all'interno di windows. La funzione va a leggere il tipo della stampante, il driver e la porta di comunicazione dal file WIN.INI, precisamente sotto la voce

"device", successivamente crea il device context. Restituisce -1 se non è stato possibile creare il device context.

unisci(stringa,numero);

unisce alla stringa il numero opportunamente trasformato in valore alfanumerico. Serve per generare le stringhe di formattazione per la visualizzazione e la stampa.

4.2.5 Modulo dei fornitori

Come in tutti i moduli, è presente il record in memoria, che in questo caso è così definito:

```
typedef struct
{
char cod_forn[lcod_forn+1],rag_soc[lrag_soc+1],
    ind[lind+1],tel[ltel+1],fax[lfax+1],pers[lpers+1];
} recforn;
```

L'unica variabile globale presente nel modulo, oltre a quelle comuni già definite è *char codice[lcod_forn]* che serve per ottenere il codice fornitore immesso in una dialog box.

Descriviamo ora ogni singola funzione, realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

BOOL FAR PASCAL codiceproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box CODICE, visualizza la finestra, dimensiona il campo di input, attende la pressione del tasto OK per

eliminare la finestra stessa. Restituisce il codice fornitore nella variabile globale codice.

*void crea_tabella(char *);*

crea la tabella fornitori e l'indice di chiave primaria sul codice fornitore.

void prendi_buffer(RECORDHANDLE);

carica i dati dal buffer e li visualizza sullo schermo.

void metti_buffer(RECORDHANDLE);

prende i dati sullo schermo e li salva nel buffer;

*void aggiusta_cod_forn(char *);*

aggiunge, davanti al codice fornitore, tanti zeri fino a raggiungere la lunghezza stabilita dal codice stesso.

4.2.6 Modulo dei ricambi

Come in tutti i moduli, è presente il record in memoria, che in questo caso è così definito:

```
typedef struct
{
char cod_ric[lcod_ric+1],desc_ric[ldesc_ric+1], note[lnote+1],
    ubicaz[lubicaz+1],cod_for1[lcod_for+1], cod_ric_for1[lcod_ric_for+1],
    costo1[lcosto+1],data1[ldata+1], cod_for2[lcod_for+1];
char cod_ric_for2[lcod_ric_for+1],costo2[lcosto+1],data2[ldata+1],
    cod_for3[lcod_for+1],cod_ric_for3[lcod_ric_for+1],
    costo3[lcosto+1],data3[ldata+1];
} recric;
```

Descriviamo ora le variabili globali presenti nel modulo, oltre a quelle comuni già definite:

char codice[lcod_ric];

variabile globale che serve per ottenere il codice ricambio immesso in una dialog box.

Descriviamo ora ogni singola funzione, realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

BOOL FAR PASCAL codiceproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box CODICE, visualizza la finestra, dimensiona il campo di input, attende la pressione del tasto OK per eliminare la finestra stessa, dopo aver controllato che il codice inserito sia corretto. Restituisce il codice del ricambio nella variabile globale codice.

*void crea_tabella(char *);*

crea la tabella dei ricambi e l'indice di chiave primaria sul codice del ricambio.

void buffer_record(RECORDHANDLE);

carica i dati dal buffer e li memorizza nel record recric. Converte il formato DATE di paradox engine in una data in formato stinga e converte i dati numerici in formato stringa; questo perché il record recric mantiene tutte le informazioni come stringhe, in modo da scriverle sullo schermo immediatamente.

void record_schermo(RECORDHANDLE);

prende i dati del record recric e li visualizza a video.

*void schermo_record(recric *);*

prende i dati sullo schermo e li memorizza nel record recric.

void record_buffer(recric,RECORDHANDLE);

prende i dati memorizzati nel record recric e li trasferisce nel buffer. Converte le stringhe riferite a campi numerici in numeri e converte le date sotto forma di stringhe in formato DATE.

*void shifta_fornitori(recric *);*

sposta i gruppi di dati riferiti ai tre fornitori, precisamente sposta il fornitore 1 nel fornitore 2 e il fornitore ultimo nel fornitore 1.

*void azzera_rec(recric *);*

pulisce il record recric mettendo a NULL tutte le stringhe;

int controlla(recric);

controlla che i campi presenti nel record recric siano corretti; precisamente controlla che il codice ricambio sia formattato, che il costo ultimo sia numerico e che la data ultima sia corretta.

int modificato_fornitore();

verifica se è stato modificato l'ultimo fornitore, in caso affermativo restituisce true, false in caso contrario.

4.2.7 Modulo delle strutture

In questo modulo sono presenti alcune costanti aggiuntive, soprattutto relative alla tabella 'note' e 'calendari', esse sono: il nome della tabella calendario, l'identificativo delle note (*idnote*), il nome del calendario di default (*caldefault*), il numero dei campi della tabella note (*ncampi2*) e le lunghezze dei campi della tabella note.

Come in tutti i moduli è presente il record in memoria, che in questo caso è così definito:

```
typedef struct
{
char cod_int[lcod_int+1],cod_cespite[lcod_cespite+1],
    centro_costo[lcentro_costo+1],centro_costo_int[lcentro_costo_int+1],
    costo_orario[lcosto_orario+1],desc[ldesc+1], flag_ap[lflag_ap+1];
char data_ap[ldata+1],cod_forn[lcod_forn+1],ore_g_lav[lore_g_lav+1],
    ore_g_sab[logsab+1], ore_anno[lore_anno+1], potenza[lpotenza+1];
int note;
char ubicazione[lubicazione+1],reparto[lreparto+1],
    cod_forn_contr[lcod_forn_contr+1], nome_calendario[lcalendario+1];
} recstr;
```

Descriviamo ora le variabili globali presenti nel modulo, oltre a quelle comuni già definite:

*static char *campi2[ncampi2]={...};*

definisce i nomi dei campi del record di paradox engine, relativi alla tabella delle note.

*static char *tipi2[ncampi2]={...};*

definisce i tipi dei campi del record di paradox engine, relativi alla tabella delle note.

static char nome_tabnote;

il nome della tabella delle note.

int modificate_note;

flag che indica se le note sono state modificate oppure no.

char codice[lcod_str+1];

variabile globale che serve per ottenere il codice della struttura immesso in una dialog box.

char calendario[lcalendario+1];

variabile globale che serve per ottenere il nome del calendario immesso in una dialog box.

char data_iniziale[ldata+1],data_finale[ldata+1];

variabili globali che servono per ottenere la data iniziale e finale immesse in una dialog box.

Descriviamo ora ogni singola funzione realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

BOOL FAR PASCAL codiceproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box CODICE, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il codice della struttura nella variabile globale codice.

BOOL FAR PASCAL nomecalproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box APRI, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il nome del calendario nella variabile globale calendario.

BOOL FAR PASCAL visual_dataproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialogbox VISUAL, essa ha il compito di fare una ricerca, tramite l'indice sulla data di acquisto o produzione, dei record

compresi tra la data iniziale e la data finale. L'operazione è fatta generando le varie date ed effettuando la ricerca per ogni data dell'intervallo. Infine, i record ottenuti, vengono visualizzati in una listbox. Si può uscire dalla finestra premendo il bottone 'annulla' o stampando il contenuto della listbox.

BOOL FAR PASCAL dateproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box DATE, visualizza la finestra, dimensiona i campi di input relativi alla data iniziale e alla data finale, attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce le date nelle variabili globali data_iniziale e data_finale.

int trova_nuovo_puntatore();

trova un nuovo puntatore alle note, praticamente tenta di accedere all'archivio con ogni numero naturale (partendo da 0), appena ne trova uno che non esiste nell'archivio, lo restituisce.

*void crea_tabella_strutture(char *);*

crea la tabella delle strutture e gli indici ad essa associata.

void crea_tabella_note();

crea la tabella note e l'indice di chiave secondaria.

*void azzera_record(recstr *);*

pulisce il record recstr,

*void schermo_record(recstr *);*

trasferisci i dati dallo schermo al record recstr.

*int conta(int,char *,char);*

conta il numero di caratteri, uguali a quello dato, nella stringa in ingresso.

int calcola_ore(recstr);

calcola le ore totali di funzionamento e le restituisce nel record recstr.

void record_buffer(recstr,RECORDHANDLE);

trasferisce i dati dal record recstr al buffer, convertendo i valori numerici e le date in un giusto formato.

int controlla(recstr);

controlla che le informazioni nel record recstr siano corrette, più precisamente controlla: che siano formattati i codici di struttura, che sia in formato data la data di acquisto o produzione, che siano in formato numerico il costo orario, le ore e la potenza.

*void buffer_record(RECORDHANDLE,recstr *);*

trasferisce i dati dal buffer al record, convertendo in stringa i valori numerici e le date.

void record_schermo(recstr);

trasferisce le informazioni contenute nel record recstr sullo schermo.

void azzera_tutto(int,int);

pulisce lo schermo.

void carica_e_visualizza_note(int);

carica le note di puntatore dato, cercandole nella tabella 'note'.

void salva(int);

salva le note, aggiornando dapprima i record già presenti in archivio, aggiungendo poi gli eventuali nuovi record o cancellando i record, nel caso che le righe da salvare siano in numero minore di quelle presenti in archivio.

4.2.8 Modulo dei calendari

Questo modulo presenta aspetti differenti da quelli visti fino ad ora, comunque vediamo le costanti particolari di questo modulo: il nome della tabella dei calendari, il nome della tabella di default, l'identificatore della prima casella e dell'ultima casella, il numero delle parti (dei mesi) in cui è composto il record in memoria.

Come in tutti i moduli è presente il record in memoria, che in questo caso è così definito:

```
typedef
struct
 {
  char nome_calendario[lcalendario+1];
  short anno,mese,numgiorni;
  char giorni[32];
 } recmese;
```

Descriviamo ora le variabili globali presenti nel modulo, oltre a quelle comuni già definite. In questo modulo ho usato un particolare, fino ad ora non discusso, di paradox engine; cioè la possibilità di dichiarare dei manipolatori (handle) dei campi del buffer. Così sono dichiarate le variabili globali contenenti questi manipolatori: *fnomecal, fanno, fmese, fnumgiorni, fgiorni*. Sono dichiarate alcune variabili globali relative ai campi di input o di output: *int anno, char anno_corrente[lanno], char codice[lcod_int], char calendario_corrente[lcalendario]*.

Infine, è presente il record che contiene tutto il calendario in memoria chiamato '*record[npezzi*lmeserec]*'. Questo record ha il compito di tenere tutti i caratteri di ogni mese del calendario; in realtà è un'enorme stringa di **416** (13X32) caratteri, che contiene tutti i mesi separati da un carattere **'\0'**.

Descriviamo ora ogni singola funzione realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

BOOL FAR PASCAL apriproc(HWND,WORD,WORD,LONG);

funzione che gestisce la dialog box APRI, dimensiona il campo di input, se premuto OK elimina la dialog box e restituisce il nome del calendario inserito in calendario_corrente

BOOL FAR PASCAL nuovoproc(HWND,WORD,WORD,LONG);

funzione che gestisce la dialog box NUOVO, dimensiona il campo di input, se premuto OK elimina la dialog box e restituisce il nome del calendario inserito in calendario_corrente. Inoltre gestisce l'immissione dell'anno del calendario, controllandolo prima di restituirlo.

void crea_tabella();

crea la tabella 'calendari' e l'indice di chiave secondaria.

void buffer_record(TABLEHANDLE,RECORDHANDLE);

trasferisce i mesi del calendario dal buffer al record in memoria.

void record_schermo(int);

trasferisce il mese corrente dal record allo schermo.

void inizializza_record();

inizializza il record in memoria assegnano NULL a ogni stringa (o meglio a ogni pezzo di stringa).

int schermo_record(int);

trasferisce il mese corrente dallo schermo al record.

void record_buffer(RECORDHANDLE,TABLEHANDLE);

trasferisce i mesi del calendario dal record in memoria al buffer.

void azzera_schermo();

pulisce i campi della maschera.

*int correggi(char *);*

controlla che la stringa con i caratteri del mese sia corretta, restituisce l'identificatore del carattere errato.

void casella_errata(int);

evidenzia il carattere errato, ottenuto da correggi, e manda un messaggio di errore.

int esiste_calendario(TABLEHANDLE,RECORDHANDLE);

controlla che il nome del calendario sia esistente.

void salva_record_default(TABLEHANDLE,RECORDHANDLE);

crea un calendario di default (tutti i giorni come lavorativi) e lo salva nella tabella calendari con il nome specificato.

void stampa_calendario_corrente();

stampa il calendario in memoria.

4.2.9 Modulo delle richieste di lavoro

In questo modulo sono presenti alcune costanti aggiuntive, soprattutto relative alla tabella 'note' e alla tabella 'ric', esse sono:
l'identificativo delle note, l'handle del campo note nel record di paradox engine, l'identificativo del campo di output, gli identificativi delle due list

box, l'elemento <*nuovo ric.*> delle listbox, il numero dei campi della tabella note (*ncampi2*), le lunghezze dei campi della tabella note, il numero dei campi della tabella ric (*ncampi3*), le lunghezze dei campi della tabella ric, il nome della tabella dei ricambi.

Come in tutti i moduli è presente il record in memoria, che in questo caso è così definito:

```
typedef struct
{
long progressivo;
char cod_str[lcod_str+1],tipo_manutenzione[ltipo_manutenzione+1],
    data_prevista[ldata_prevista+1],data_reale[ldata_reale+1],
    ore_previste[lore_previste+1];
int ricambi_previsti;
char flag_stato_rdl[lflag_stato_rdl+1],flag_ie[lflag_ie+1],
    cod_forn[lcod_forn+1],costo_tot[lcosto_tot+1],
    costo_mdo[lcosto_mdo+1],costo_ricambi[lcosto_ricambi+1],
    ore_consuntivate[lore_consuntivate+1],
    ore_fermo_macchina[lore_fermo_macchina+1],reparto[lreparto+1];
int ricambi_consuntivi,note;
} recrdl;
```

Descriviamo ora le variabili globali presenti nel modulo, oltre a quelle comuni già definite:

*static char *campi2[ncampi2]={...};*

definisce i nomi dei campi del record di paradox engine, relativi alla tabella delle note.

*static char *tipi2[ncampi2]={...};*

definisce i tipi dei campi del record di paradox engine, relativi alla tabella delle note.

*static char *campi3[ncampi3]={...};*

definisce i nomi dei campi del record di paradox engine, relativi alla tabella 'ric'.

*static char *tipi3[ncampi3]={...};*

definisce i tipi dei campi del record di paradox engine, relativi alla tabella 'ric'.

static char nome_tabnote[12];

il nome della tabella delle note.

int modificate_note,modificate_lb1,modificate_lb2;

flags che indicano se le note o se le due listbox sono state modificate.

int prossimo_progressivo;

variabile globale che mantiene sempre il prossimo numero progressivo da utilizzare.

char codice[lcod_str+1];

variabile globale che serve per ottenere il codice della struttura o del ricambio, immesso in una dialog box.

char data_iniziale[ldata+1],data_finale[ldata+1];

variabili globali che servono per ottenere la data iniziale e finale, immesse in una dialog box.

int ispettiva,programmata,urgente,evaso,incorso,attesa,interno,esterno;

variabili globali che servono per verificare quali record devono essere visualizzati, informazioni che vengono inserite mediante una dialog box.

Descriviamo ora ogni singola funzione realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

BOOL FAR PASCAL progressivoproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box PROG, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il numero progressivo nella variabile globale prossimo_progressivo.

BOOL FAR PASCAL visual_dataproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialogbox VISUAL, essa ha il compito di fare una ricerca, tramite l'indice sulla data reale, dei record compresi tra la data iniziale e la data finale. L'operazione è fatta generando le varie date ed effettuando la ricerca per ogni data dell'intervallo. Infine i record ottenuti vengono visualizzati in una listbox. Si può uscire dalla finestra premendo il bottone 'annulla' o stampando il contenuto della listbox.

BOOL FAR PASCAL dateproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box DATE, visualizza la finestra, dimensiona i campi di input relativi alla data iniziale e alla data finale e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce le date nelle variabili globali data_iniziale e data_finale.

BOOL FAR PASCAL tutteproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box TUTTE, visualizza la finestra, inizializza tutti i radiobutton e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce la situazione dei radiobutton nelle variabili definite in precedenza (ispettiva, programmata, urgente ecc..).

BOOL FAR PASCAL codstrproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box CODSTR, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il codice della struttura nella variabile globale codice.

BOOL FAR PASCAL visual_codstrproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialogbox VISUAL, essa ha il compito di fare una ricerca di tutti i record con codice struttura dato e visualizzare ogni record in una listbox. Si può uscire dalla finestra premendo il bottone annulla o stampando il contenuto della listbox.

BOOL FAR PASCAL codricproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box CODRIC, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il codice del ricambio nella variabile globale codice.

BOOL FAR PASCAL visual_repproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialogbox VISUAL, essa ha il compito di fare una ricerca di tutti i record con reparto dato e visualizzare ogni record in una listbox. Si può uscire dalla finestra premendo il bottone 'annulla' o stampando il contenuto della listbox.

BOOL FAR PASCAL repproc(HWND,WORD,WORD,LONG);

dialog function che gestisce la dialog box REP, visualizza la finestra, dimensiona il campo di input e attende la pressione del tasto OK per eliminare la finestra stessa. Restituisce il reparto nella variabile globale codice.

*void crea_tabella_rdl(char *);*

crea la tabella 'rdl', l'indice di chiave primaria sul numero progressivo e tutti gli indici di chiave secondaria (struttura, data reale, reparto).

*void azzera_record(recrdl *);*

pulisce il record, mettendo a NULL tutte le stringhe.

void buffer_record(RECORDHANDLE);

carica i dati dal buffer e li memorizza nel record recrdl. Converte il formato DATE di paradox engine in una data in formato stringa e converte i dati numerici in formato stringa; questo perché il record recrdl mantiene tutte le informazioni come stringhe, in modo da scriverle sullo schermo immediatamente.

void record_schermo(RECORDHANDLE);

prende i dati del record recrdl e li visualizza a video.

*void schermo_record(recrdl *);*

prende i dati sullo schermo e li memorizza nel record recrdl.

void record_buffer(recrdl,RECORDHANDLE);

prende i dati memorizzati nel record recrdl e li trasferisce nel buffer. Converte le stringhe, riferite a campi numerici, in numeri e converte le date, sotto forma di stringhe, in formato DATE.

int controlla(recrdl);

controlla che le informazioni del record recrdl siano corretto, precisamente controlla che il codice struttura sia formattato, la data prevista e la data reale siano numeriche e possibili date, le ore previste siano numeriche; così come il costo totale, il costo della manodopera e dei ricambi; infine controlla che le ore consuntivate e le ore di fermo macchina siano numeriche.

void azzera_tutto(int,int);

pulisce i campi sullo schermo.

*void assegna_progressivo(recrdl *);*

legge nel record 0 dell'archivio 'rdl' il prossimo numero progressivo e lo assegna al record in ingresso.

void visualizza_progressivo(recrdl);

visualizza sullo schermo il prossimo progressivo.

void salva_progressivo(recrdl);

cerca il record 0 dalla tabella 'rdl', prende il progressivo memorizzato, lo incrementa e lo aggiorna nel record stesso.

void crea_tabella_ricambi();

crea la tabella 'ric' e l'indice di chiave secondaria.

void carica_e_visualizza_lb(int,int);

carica i ricambi di puntatore dato, cercandoli nella tabella 'ric'.

void cancella_ricambi(int);

cancella tutti i ricambi di puntatore dato; utilizzato quando è cancellata una richiesta di lavoro.

void salva_lb(int,int);

salva la lista dei ricambi, aggiornando dapprima i record già presenti in archivio, aggiungendo poi gli eventuali nuovi record o cancellandoli, nel caso che gli elementi da salvare siano in numero minore di quelli presenti in archivio.

void inizializza_lb();

inserisce nella listbox solamente l'elemento <nuovo ric.>.

*int trova_ambedue_puntatori_lb(int *,int *);*

trova un primo puntatore ai ricambi, come visto precedentemente in trova_puntatore, poi continua la ricerca per un secondo puntatore libero.

Infine, tutta la gestione delle note è identica al modulo delle strutture, per questo facciamo riferimento a tale modulo.

4.2.10 Modulo della distinta base

Questo modulo presenta parecchi aspetti particolari, per questo alcune funzioni sono differenti. In questo modulo sono presenti tutte le costanti del modulo delle strutture e dei ricambi, infatti questo modulo deve accedere alle due tabelle, oltre al proprio archivio 'db'. Questo significa che sono definiti i nomi, i campi e i tipi per ognuna delle tre tabelle.

Come in tutti i moduli è presente il record in memoria, ma in questo caso sono definiti anche i record per le strutture e i ricambi, facciamo riferimento per la definizione ai precedenti moduli:

```
typedef struct
{
char cod_int[lcod_int+1],cod_ric[lcod_ric+1],qta[lqta];
} recdb;
```

Descriviamo ora le variabili globali presenti nel modulo, oltre a quelle comuni già definite. È definito un nome della tabella della distinta base, un array contenente gli handle delle finestre di testo in cui vengono scritte le descrizione ottenute dalla ricerca.

Abbiamo un array di puntatori a funzioni (*puntatore_funzione[nfin]*), contenente i puntatori alle funzioni che filtrano i messaggi delle finestre di edit (*filtro*). Infine, abbiamo una serie di variabili globali, che servono per ottenere i dati di input dalle dialog box; esse sono: *descrizione, qta e codice*.

Descriviamo ora ogni singola funzione realizzata all'interno del modulo, tralasciando il winmain, la funzione principale e le altre funzioni, descritte in precedenza:

long FAR PASCAL filtro(HWND,WORD,WORD,LONG);

filtra i messaggi alle finestre di edit. Se viene premuto INVIO genera un MIOMESSAGGIO, di tipo NOTIFICA_INVIO_x_xxx, a seconda del campo a cui il messaggio si riferisce. Se viene premuto ESC spedisce un messaggio di ID_ESCI alla finestra principale. La stessa funzione controlla che il codice inserito sia corretto.

long FAR PASCAL filtro_list_box(HWND,WORD,WORD,LONG);

questa funzione ha le stesse competenze della precedente, solo che intercetta i messaggi inviati alla listbox. Questa permette di trasformare la pressione del tasto INVIO in un messaggio di LBN_DBLCLK sull'elemento attualmente selezionato. Anche questa funzione permette di intercettare il tasto ESC per ritornare al menù (messaggio ID_ESCI alla finestra madre).

BOOL FAR PASCAL qtaproc(HWND,WORD,WORD,LONG);

questa funzione gestisce la dialog box QTA, dimensiona il campo di input, se premuto OK elimina la dialog box e restituisce la quantità nella variabile globale qta.

BOOL FAR PASCAL infostrproc(HWND,WORD,WORD,LONG);

questa funzione gestisce la dialog box INFOSTR che visualizza le informazioni di una struttura. Si ottiene il codice attualmente selezionato nella list box, si cerca il record, si caricano le informazioni nel buffer e si visualizzano nella listbox. Se premuto il bottone OK si elimina la dialog box.

BOOL FAR PASCAL inforicproc(HWND,WORD,WORD,LONG);

questa funzione gestisce la dialog box INFORIC, il funzionamento è identico alla funzione precedente, solo che la ricerca avviene sulla tabella dei ricambi.

*void crea_tabella(char *);*

crea la tabella 'db' e i suoi indici: quello di chiave primaria sulla coppia di codici e i due di chiave secondaria sui due differenti codici.

*int restituisci_descrizione(char *,char *,char *);*

cerca il record con codice ricambio o codice struttura dato (la funzione cerca sulla tabella in base allo stato). Carica la descrizione e la restituisce in una stringa di output. La funzione restituisce true se la ricerca ha avuto successo, false in caso contrario.

void crea_legame(TABLEHANDLE,RECORDHANDLE,recdb);

controlla che il legame non sia già esistente; in caso affermativo visualizza un messaggio di errore, altrimenti richiama la dialog box per chiedere la quantità e aggiunge un record con il nuovo legame.

void abilita_tutti_menu(int);

abilita o disabilita, a seconda del flag in ingresso, tutte le voci del menù.

int riempi_list_box(TABLEHANDLE,RECORDHANDLE,recdb);

dato il codice struttura o il codice ricambio, ricerca tutti i record che hanno un legame con il codice in ingresso. A seconda dello stato, la funzione cerca tramite l'indice sul codice padre o sul codice figlio. Successivamente, aggiunge ogni elemento letto nella listbox, dopo aver ottenuto la descrizione, anch'essa visualizzata nella list box.

*void buffer_record_ricambi(recric *);*

apre la tabella dei ricambi, cerca il record con codice ricambio dato, e restituisce un record recric con tutte le informazioni sul ricambio; usata per visualizzare le informazioni nelle dialog box di informazione.

*void buffer_record_strutture(recstr *);*

questa funzione ha gli stessi compiti della precedente, solo che opera sulla tabella delle strutture; usata per visualizzare le informazioni nelle dialog box di informazione.

void cancella_legame(TABLEHANDLE,RECORDHANDLE,recdb);

chiede conferma per la cancellazione del legame, se essa viene confermata, cancella il record relativo a quel legame.

void pulisci_schermo();

pulisce i campi di edit e la listbox.

CAPITOLO 5
Conclusioni

5.1 Risultati ottenuti

Siamo giunti alla fine del lavoro, ora abbiamo un pacchetto funzionante, che permette all'azienda di automatizzare la gestione delle manutenzioni.
In questo capitolo intendo discutere **due** aspetti: se le scelte effettuate sono state quelle giuste e se si sono raggiunti i tre obiettivi che c'eravamo posti all'inizio del lavoro.

❑ Scelte effettuate

Posso affermare, con soddisfazione, che le scelte effettuate prima dell'analisi e lo sviluppo del pacchetto si sono rivelate giuste.
Dico questo, per due motivi:

- La scelta del compilatore della borland, per la realizzazione del programma, mi ha favorito parecchio. Inizialmente ho lavorato con la versione 2.0 del compilatore, questa versione aveva solamente l'ambiente di sviluppo sotto DOS; ma durante la realizzazione del pacchetto è uscita la versione 3.0 (e subito dopo la **3.1**), con ambiente di programmazione sotto windows.

Questo mi ha reso le cose molto più semplici, almeno dal punto di vista dell'efficienza degli strumenti di lavoro. Di estrema importanza è stata la possibilità di usare, ad esempio, un programma di *debug* per applicazioni windows.

- La scelta di paradox engine, per sviluppi futuri, risulta molto interessante. Infatti è da tempo che la borland ha annunciato la preparazione del suo diffuso DBMS "*Paradox*", totalmente sviluppato in ambiente Microsoft Windows.
  Questo potrebbe portare a interessanti utilizzi degli archivi del pacchetto, direttamente sotto il medesimo ambiente di sviluppo del programma stesso; con grossi vantaggi per l'azienda, in termini di trasporto di dati fra applicazioni diverse.

❑ **Obiettivi prefissati**

All'inizio della tesi, ci siamo prefissati **tre** grandi obiettivi: organizzare la gestione delle manutenzioni, determinare i costi e partecipare alla programmazione aziendale.

Purtroppo, mentre sto scrivendo questa parte finale, non posso sapere i giudizi e le critiche di chi utilizzerà il software, in quanto il programma non è ancora installato presso la Coop Bilanciai; aldilà di questo, posso azzardare conclusioni personali, dovute soprattutto alle prove effettuate sul pacchetto.

Sicuramente il primo obiettivo è stato pienamente raggiunto, infatti, adesso il responsabile delle manutenzioni ha innumerevoli vantaggi, ad esempio:

- Interrogando giornalmente le richieste di lavoro, per intervallo di data, egli può sapere quali manutenzioni devono essere compiute in quel dato giorno;
- Interrogando la distinta base, il responsabile delle manutenzioni può sapere quali ricambi servono per una data struttura, inoltre può ottenere automaticamente l'eventuale fornitore a cui riferirsi per ordinare il materiale.

Il secondo obiettivo non è stato realizzato completamente, in quanto il responsabile delle manutenzioni può avere a disposizione tutti i costi desiderati, ma non ha possibilità di collegamento con la contabilità industriale.
Il discorso appena concluso è valido anche per il terzo obiettivo, in quanto non è stato realizzato alcun modulo di tipo statistico, ma le fondamenta ci sono; infatti, le anagrafiche comprendono tutti i dati che servono per la programmazione aziendale, occorre solo l'implementazione di un modulo adeguato.

5.2 Sviluppi futuri

Il pacchetto, indubbiamente, non è completo, e non ha neanche la pretesa di esserlo, però è stato concepito, fin dalla fase di analisi, per essere integrato in qualsiasi momento.

Già la sua struttura modulare ci mostra come è possibile aggiungere un nuovo modulo, senza modificare quelli già esistenti.
Sicuramente, in futuro, il pacchetto si espanderà, soprattutto in **tre** direzioni:

- un modulo statistico che permetterà di completare il terzo obiettivo che c'eravamo posti;
- un modulo che permetterà l'elaborazione di dati sintetici, sui costi che gravano sulle manutenzioni;
- la possibilità di acquisizione automatica dei dati da parte della contabilità industriale.

Per i primi due punti, il problema è solamente la realizzazione pratica dei moduli, il terzo punto, invece, comporta una serie di problematiche, relative al trasferimento di dati da macchine di tipo Personal Computer a macchine di tipo **MINI**, come appunto l'**AS400**.