3.1 Modello di programmazione modulare

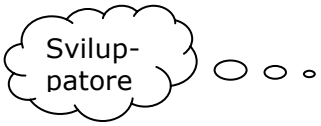
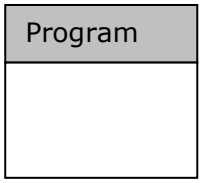
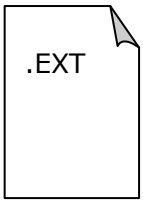
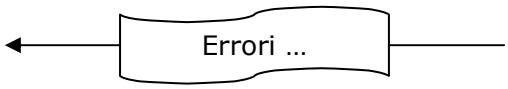
Ambiente di sviluppo

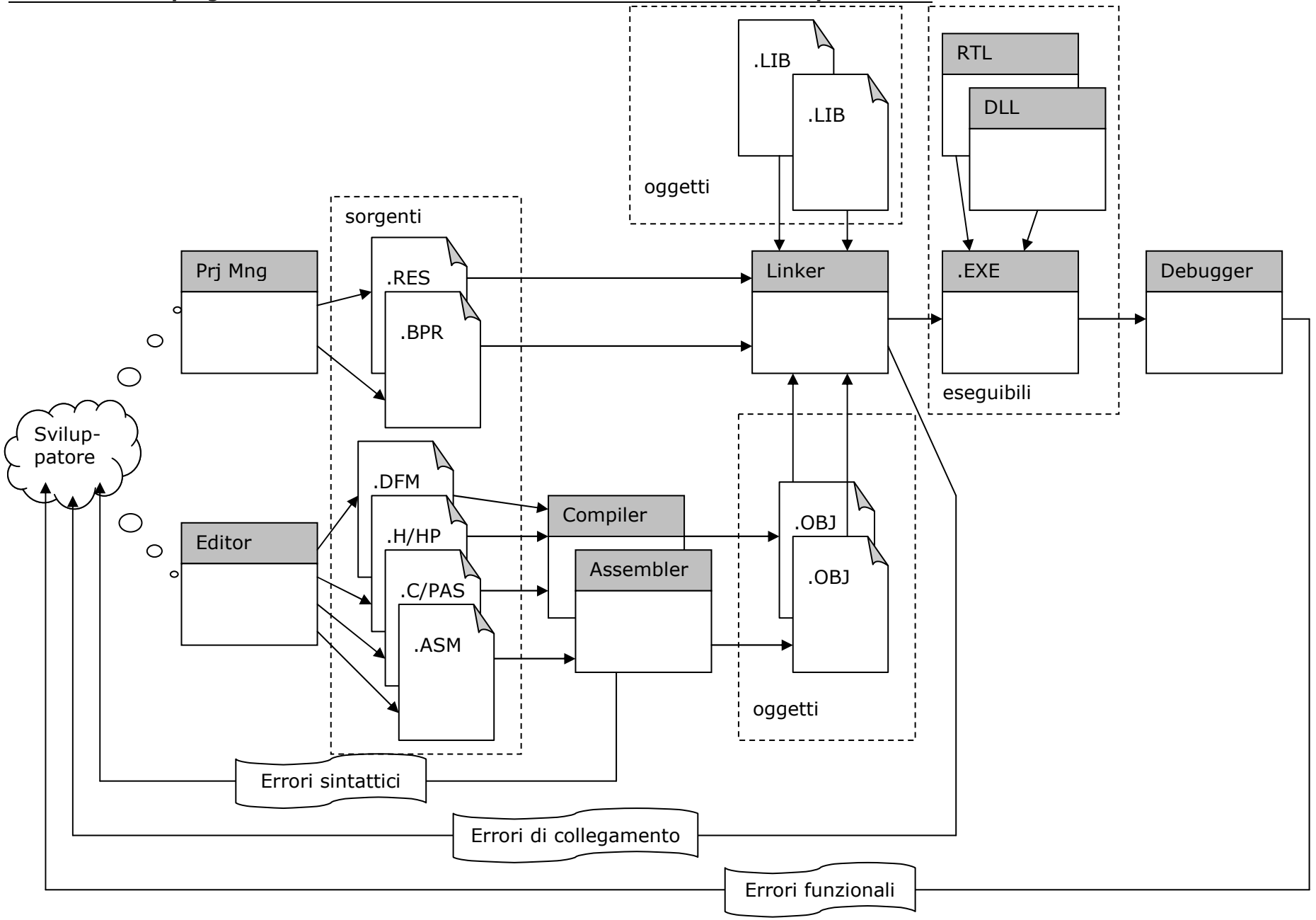
In questa unità viene analizzata la struttura di un ambiente di sviluppo (DE = Development Environment).
Un ambiente di sviluppo è un insieme di programmi che vengono utilizzati insieme per produrre altri programmi. Il termine "sviluppo" che viene usato al posto del termine "progetto" pone proprio in evidenza questa caratteristica peculiare dell'informatica che l'unica tecnologia in cui gli strumenti di progettazione e il risultato del progetto sono della stessa natura (si producono programmi usando come strumenti di produzione dei programmi).
In alcuni casi le varie componenti dell'ambiente di sviluppo sono talmente integrate tra loro da dare l'impressione di essere un unico programma (IDE = Integrated Development Environment). E' questo il caso dell'ambiente Borland Builder C++ che viene usato come esempio in questo studio.
Nonostante BBC appaia allo sviluppatore come un unico programma in realtà il programma che viene lanciato all'inizio della sessione di lavoro e che mostra il pannello dei comandi non è altro che un "Framework" (cornice) che serve per la esecuzione di altri programmi indipendenti che fanno parte dell'ambiente.

Schema a blocchi del processo di sviluppo

Lo schema che segue rappresenta, riferito all'ambiente BBC5, il processo di sviluppo di una applicazione modulare.

Spiegazione del significato dei simboli:

Simbolo	Significato
	Rappresenta simbolicamente il progettista che agisce attraverso i comandi dell'ambiente di sviluppo per la produzione del programma.
	Rappresenta un componente dell'ambiente IDE. E' un programma che riceve in ingresso comandi e documenti e produce documenti in un altro formato.
	Rappresenta un documento prodotto da un programma dell'ambiente IDE o usato da un programma dell'ambiente IDE. Il tipo di documento è rappresentato dall'estensione scritta nel documento
	Rappresenta un gruppo di documenti aventi caratteristiche omogenee, cioè documenti dello stesso tipo.
	Rappresenta una trasformazione di un documento operata da un programma
	Rappresenta una segnalazione di errore generata da un programma che opera su un documento.


.LIB
.LIB
oggetti
RTL
DLL
sorgenti
Prj Mng
.RES
.BPR
Linker
.EXE
Debugger
eseguibili
Svilup-
patore
.DFM
.H/HP
.C/PAS
.ASM
Compiler
Assembler
Editor
.OBJ
.OBJ
oggetti
Errori sintattici
Errori di collegamento
Errori funzionali

Descrizione del processo

Produzione dei sorgenti
I sorgenti sono documenti prodotti dallo sviluppatore e costituiscono la descrizione del progetto. I sorgenti sono in genere file di puro testo che contengono in forma simbolica le istruzioni che andranno a formare il programma. I files di testo vengono prodotti con un editor di puro testo integrato nell'ambiente di sviluppo.
Appartengono a questa categoria i documenti di tipo:
- .**C** : file sorgente contenente un modulo di funzioni in linguaggio C
- **.CPP**: file sorgente contenente un modulo do funzioni in linguaggio C++
- **.ASM**: file sorgente contenente un modulo di istruzioni in linguaggio ASM
- **.PAS**: file sorgente contenente un modulo procedure in linguaggio PASCAL
- **.H**: file sorgente contenente tipi e prototipi da includere in un modulo di file sorgente C e CPP in modo che un modulo possa usare variabili e funzioni definite in altri moduli.
- **.HPP**: file sorgente contenente tipi e prototipi provenienti da un modulo PASCAL da includere in un modulo di file sorgente C e CPP

In alcuni casi i documenti sorgenti non sono di puro testo ma possono avere altri formati. Questi files vengono prodotti con comandi integrati nell'ambiente di sviluppo.
- **.DFM**: file sorgente contenente la descrizione dei componenti visuali di un progetto che utilizza la libreria VCL. Può essere sia in formato puro testo che codificato.
- **.BPR**: file di progetto che contiene l'elenco dei componenti sorgente e delle librerie che formano il progetto ed i parametri di compilazione, assemblaggio e linkaggio. E' in formato di puro testo codificato in linguaggio XML.
- **.RES**: file di risorse che contiene, in forma binaria, informazioni aggiuntive sul progetto quali icone e cursori. In questo caso non esiste un vero e proprio sorgente perché i comandi dell'ambiente integrato generano direttamente un file binario. Altri ambienti di sviluppo invece sono dotati di un vero e proprio "resource Compiler" e in tale caso esiste un vero e proprio file sorgente che ha estensione .RC.

In ogni caso questi files costituiscono gli originali per rigenerazione del progetto e quindi vanno archiviati in modo da poter riprodurre il processo produttivo.

Produzione degli oggetti
Gli oggetti sono documenti contenenti la traduzione in codice macchina dei moduli sorgente generati dallo sviluppatore.
Per ogni linguaggio esiste un diverso traduttore. Ad esempio nel caso di BBC5 che è in grado di tradurre sorgenti C/C++, PASCAL ed ASSEMBLY x86 ci sono tre traduttori:
- **ASSEMBLATORE**: l'assemblatore è un programma che accetta in ingresso un file sorgente contenente un modulo scritto in linguaggio ASSEBLY e lo traduce in istruzioni in linguaggio macchina generando un file binario di tipo .OBJ. Esiste una corrispondenza 1-a-1 tra linguaggio assembly e linguaggio macchina, cioè ogni istruzione assembly simbolica viene tradotta in una istruzione codificata in binario in linguaggio macchina. Esistono però direttive del linguaggio assembly che non generano istruzioni ma governano il funzionamento dell'assemblatore e del linker.
- **COMPILATORE C/C++**: il compilatore C/C++ è un programma che accetta in ingresso un file sorgente contenente un modulo scritto in linguaggio C/C++ e lo traduce in istruzioni in linguaggio macchina generando un file binario di tipo .OBJ. Non esiste una corrispondenza 1-a-1 tra linguaggio C/C++ e linguaggio macchina, quindi ogni istruzione C/C++ viene tradotta in una sequenza di una o più istruzioni codificate in binario in linguaggio macchina. Esistono anche direttive del linguaggio C/C++ che non generano istruzioni ma governano il funzionamento dell'assemblatore e del linker. Inoltre il compilatore C/C++ utilizza prima della traduzione vera e propria un programma precompilatore che effettua le sostituzioni di testo richieste dalle direttive di precompilazione. Ad esempio in questa fase le costanti simboliche vengono sostituite dai loro valori indicati nelle direttive #define e è i file di inclusione vengono inseriti nella posizione che loro compete in base alle direttive #include. Al termine di questa fase sul file di testo trasformato inizia la compilazione vera e propria.
- **COMPILATORE PASCAL:** il compilatore PASCAL è un programma che accetta in ingresso un file sorgente contenente un modulo scritto in linguaggio PASCAL e lo traduce in istruzioni in linguaggio macchina generando un file binario di tipo .OBJ. Non esiste una

corrispondenza 1-a-1 tra linguaggio PASCAL e linguaggio macchina, quindi ogni istruzione PASCAL viene tradotta in una sequenza di una o più istruzioni codificate in binario in linguaggio macchina. Esistono anche direttive del linguaggio PASCAL che non generano istruzioni ma governano il funzionamento dell'assemblatore e del linker. Il linguaggio PASCAL non usa i files di inclusione; le dichiarazioni di tipo e di prototipo sono quindi sempre integrate nel file sorgente .

Una particolare categoria di oggetti è costituita dalle librerie statiche (.LIB). Le librerie statiche sono files di tipo oggetto che contengono, codificate il linguaggio macchina, le funzioni di libreria di uso più comune i cui prototipi sono definiti a livello di sorgente nei files .H. Le librerie non sono quindi in genere disponibili in forma di sorgente ma solo di files oggetto già compilati forniti dal produttore dell'ambiente di sviluppo. A parte la mancanza della fase di editing dei sorgenti e della compilazione le librerie statiche si comportano a tutti gli effetti come gli altri moduli: il loro utilizzo è definito nel file di progetto (con un procedimento automatizzato), vengono inserite dal linker nell'eseguibile.

La produzione di un eseguibile con le librerie statiche porta ad un eseguibile che non richiede procedure di installazione ma produce un file eseguibile di dimensioni considerevoli. Una possibile alternativa, e in alcuni casi una necessità è l'impiego delle librerie di run-time, che rendono l'eseguibile più piccolo ma rendono il funzionamento dell'eseguibile dipendente dalla presenza nell'installazione di particolari file eseguibili che costituiscono le librerie di run-time (.RTL, .DLL, .OCX …).

La fase di traduzione può portare alla identificazione di errori sintattici. Gli errori sintattici sono errori nella scrittura delle istruzioni in forma sorgente. In presenza di errori sintattici il file oggetto non viene generato; è quindi necessario rimuovere gli errori sintattici, attraverso una riedizione dei files sorgente ed eseguire una nuova compilazione

Produzione dell'eseguibile

La produzione dell'eseguibile è effettuata dal LINKER. Il Linker è un programma che, sulla base delle indicazioni fornite dal file di progetto, unisce insieme i vari moduli che formano il progetto collegando tra loro le parti di moduli che hanno riferimenti incrociati.

Se ad esempio un modulo contiene la chiamata ad una funzione che si trova in un altro modulo o in un modulo di libreria o contiene un riferimento ad una variabile che si trova in un altro modulo il compilatore (assemblatore) nel modulo oggetto che genera non può determinare l'effettivo indirizzo della funzione o della variabile riferita. In questo caso il compilatore (assemblatore) lascia uno spazio nel codice macchina in corrispondenza dell'indirizzo mancante pari alle dimensioni dell'indirizzo. Il linker, dopo avere unito i moduli, in fase di collegamento, quando trova un indirizzo mancante, individua l'effettivo indirizzo della funzione o variabile e sostituisce l'indirizzo mancante con l'indirizzo effettivo. In questo modo genera un file eseguibile, cioè in grado di essere caricato in memoria centrale ed eseguito dalla CPU.

I componenti che vengono collegati dal linker sono:

- **.OBJ**: files oggetto che contengono funzioni codificate attraverso istruzioni in linguaggio macchina indipendentemente dal linguaggio sorgente da cui sono state generate.
- **.LIB**: files oggetto che contengono funzioni codificate attraverso istruzioni in linguaggio macchina predefinite nell'ambiente di sviluppo
- **.RES**: risorse utilizzate dall'applicazione (icone, cursori …) codificate in binario in un formato che dipende dalla risorsa.

Le librerie di run-time non vengono collegate dal linker perché prevedono un collegamento dinamico effettuato durante la fase di inizializzazione del programma.

La fase di collegamento può portare alla identificazione di errori di collegamento. Gli errori collegamento sono errori dovuti ad incompatibilità tra i moduli. Questi errori non possono emergere durante la compilazione perché in questa fase ogni modulo viene trattato separatamente. In presenza di errori di collegamento il file eseguibile non viene generato; è quindi necessario rimuovere gli errori di collegamento , attraverso una riedizione dei files sorgente ed eseguire una nuova compilazione e collegamento

Collaudo dell'eseguibile

La generazione di un eseguibile privo di errori sintattici e di collegamento non garantisce che il programma sia correttamente funzionante perché ci potrebbero essere degli errori funzionali. Esistono vari tipi di errori funzionali di diversa gravità:

- Il programma funziona correttamente ma non esegue la funzione richiesta o la esegue in un modo diverso da quello atteso.
- L'esecuzione di una funzione viene interrotta da una eccezione (interrupt interno) che impedisce il completamento della funzione ma non è fatale per l'esecuzione del resto del programma (un caso tipico sono gli errori di lettura/scrittura nei files)
- L'esecuzione del programma viene interrotta per un errore fatale e il programma viene scaricato dalla memoria centrale (un caso tipico sono le divisioni per 0 e l'accesso a zone di memoria fuori dallo spazio del programma

In alcuni casi la causa dell'errore è evidente durante l'esecuzione del programma ma nella maggior parte dei casi rimane nascosta.

Per individuare la causa di errore è necessaria la "tracciatura" del programma che consiste in una esecuzione controllata con la possibilità di sospendere temporaneamente l'esecuzione ("Breakpoint") e di esplorare il contenuto delle variabili di stato interno ("Watch").

Il programma che governa la tracciatura si chiama DEBUGGER. Il debugger è un programma che carica in memoria centrale l'eseguibile senza mandarlo in esecuzione, intercetta il vettore di interrupt interno #3. Quando lo sviluppatore imposta un breakpoint su una particolare istruzione del programma sotto test, il debugger sostituisce il codice macchina dell'istruzione di arrivo con il codice macchina della istruzione INT 3 che genera un interrupt interno # 3. Quando l'utente preme il bottone RUN il DEBUGGER passa il controllo della CPU al programma sotto test che inizia (riprende l'esecuzione) ma quando il programma arriva ad eseguire l'istruzione in corrispondenza del breakpoint, invece che trovare l'istruzione vera trova una istruzione INT 3 (TRAP), la CPU la esegue e quindi il controllo torna al DEBUGGER con il programma sotto test congelato all'istruzione di break non ancora eseguita e con la possibilità di esplorare il contenuto delle variabili. Per consentire l'esecuzione dell'istruzione in corrispondenza del break il debugger ripristina il codice macchina originale che aveva salvato nella sua memoria dati. Per garantire la possibilità di inserire sempre un TRAP in sostituzione di qualunque istruzione la istruzione INT 3, a differenza dalle altre istruzioni INT # (CDH,##H) è formata da un singolo codice operativo (CCH).

Durante la fase esecutiva, il programma può richiedere il caricamento di librerie di run-time. Le librerie di run-time sono dei veri e propri programmi che vengono caricati in memoria centrale su richiesta di un altro programma e svolgono funzioni accessorie per il programma che le ha richieste. Il vantaggio nell'utilizzo delle librerie di run-time è dovuto al fatto che più programmi in esecuzione contemporaneamente possono utilizzare la stessa libreria ottenendo quindi un considerevole risparmio nella occupazione della memoria centrale. Lo svantaggio è che questi programmi accessori devono essere presenti nell'installazione in cui il programma principale è in esecuzione.

I più comuni tipi di librerie di run-time sono:

- **.RTL**: Run Time Library librerie di run-time proprietarie dell'ambiente Borland, vengon fornite con l'ambiente di sviluppo e devono essere ridistribuite.
- **.DLL**: Dynamic Link Library: librerie di run time definite secondo lo standard Microsoft. Appartengono a questa categoria sia librerie di uso molto comune sempre presenti in qualsiasi installazione sia librerie specializzate sviluppate dal progettista per risolvere problemi specifici.
- .**OCX**: OLE Control Extension: sono componenti di tipo ActiveX secondo lo standard Microsoft. Sono veri e propri programmi che collaborano con il programma principale per la realizzazione di funzioni specifiche.

La fase di debug può portare alla identificazione di errori funzionali. Gli errori funzionali sono errori nel comportamento del programma eseguibile. In presenza di errori funzionali il programma, pur entrando in esecuzione, non rispetta le specifiche; è quindi necessario rimuovere gli errori funzionali, attraverso una riedizione dei files sorgente ed eseguire una nuova compilazione, collegamento e collaudo.