Lettura e scrittura da file

Come la lettura e la scrittura da I/O (tastiera e video) per poter leggere e/o scrivere da/su file è necessario includere le definizioni del tipo fstream e le operazioni su di esso.

```
#include <fstream>
#include <iostream>
using namespace std;
    main() {
    double vettore[5];
        ifstream InFile;                    /* Dichiarazione di tipo */
        InFile.open ("dati.dat");           /* Apertura del file */
        if (!InFile) {
        cout << "Errore di apertura del file" << endl;   /* controllo */
    } else {
        for(int i=0;i<5;i++) {
                InFile >> vettore[i];       /* lettura dati */
                cout << vettore[i] << endl;
        }
        InFile.close();                     /* chiusura file */
    }
```

istruzione necessaria per le ultime versioni di compilatori

Lettura e scrittura da file

Completamente analoga è la scrittura

```
#include <fstream>
#include <iostream>
using namespace std;
    main() {
    double vettore[5]={0,1,2,3,4 };
        ofstream OutFile;                   /* Dichiarazione di tipo */
        OutFile.open ("dati.dat");          /* Apertura del file */
        if (!OutFile) {
        cout << "Errore di apertura del file" << endl;   /* controllo */
    } else {
        for(int i=0;i<5;i++) {
                OutFile << vettore[i]<<endl;        /* scrittura dati */
        }
        OutFile.close();                    /* chiusura file */
    }
```

Se il file non esiste viene creato, se esiste è sovrascritto. Ci può essere errore se non c'è sufficiente spazio su disco

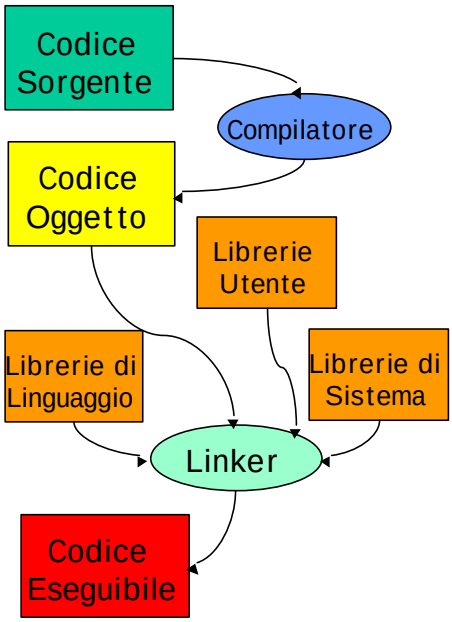
Compilazione e link

Il Codice Sorgente è scritto in linguaggio evoluto; usa funzioni complesse e nomi simbolici per variabili e functions.

Il Codice Oggetto è scritto in linguaggio machina, ma contiene riferimenti simbolici a variabili e subroutines. Usa subroutines per eseguire funzioni complesse.

Il Codice Eseguibile è scritto in linguaggio macchina e usa indirizzi di memoria per indicare variabili e functions.

Nel Sorgente e nell'Oggetto possono esistere riferimenti a variabili o functions esterne, cioè non definite nel codice medesimo. Il Codice Eseguibile non contiene riferimenti esterni.



Istruzioni per la compilazione

La compilazione di un sorgente C++ ed il link sono eseguiti dallo stesso comando: g++

La sola compilazione si ottiene con

g++ –c –o <oggetto>.o <sorgente>.c

Se si vuole eseguire compolazione e link si usa

g++ –o <eseguibile> <sorgente>.c <oggetti>.o –l<librerie>

Le librerie si aggiungono con l'opzione –l ed hanno nomi del tipo lib<nome>.a oppure lib<nome>.so Le librerie di linguaggio e quelle di sistema sono aggiunte automaticamente dal comando g++.

Elementi di un linguaggio evoluto

In ogni linguaggio evoluto si possono identificare diverse componenti:

- Sintassi: indica l'insieme delle regole generali che governano la scrittura del programma.
- Struttura: indica il modo in cui è possibile suddividere un programma complesso in elementi più semplici e fornisce le regole per la comunicazione tra i sottoprogrammi.
- Variabili: sono le regole per la definizione e l'utilizzo delle aree di memoria destinate ai dati da elaborare.
- Controllo di flusso: sono le istruzioni per l'esecuzione di cicli,test, etc…
- Funzioni: sono i comandi che si usano per eseguire funzioni complesse (input/output, funzioni matematiche…).

Esempio di un programma in C++

```
main () {
   int i;
   double x;
   for (i=0; i<100; i++) {
      x = i/100.0;
      cout << "cos(" << i << "/100) = ",cos(x), endl;
   }
}
```

Subroutine

Variabili

Controllo di flusso

Funzione I/O

Funzione matematica

Output:

```
cos(0/100) = 1.000000
cos(1/100) = 0.999950
cos(2/100) = 0.999800
…
```

Codifica in memoria

Le variabili di un linguaggio evoluto sono porzioni della memoria che contengono i dati dell'utente, scritti in una qualche rappresentazione binaria.

I formati possibili sono:

- Interi con o senza segno
- "Reali" cioè numeri rappresentati in virgola mobile
- "Stringhe" di caratteri alfanumerici
- Variabili logiche
- Tipi composti
- Rappresentazioni definite dall'utente…

Le variabili

Una variabile è una porzione della memoria che contiene dati (di ingresso o di uscita) codificati in un qualche modo.

Nei linguaggi evoluti, le variabili sono identificate da un nome, nei linguaggi macchina da un indirizzo. Posseggono le seguenti caratteristiche:

- Tipo: specifica il tipo di codifica usato
- Dimensione: indica la quantità di memoria occupata (dim = n_comp*dim_comp)
- Modo di allocazione: indica la strategia usata per riservare lo spazio in memoria
- Scope (visibilità): indica quali pezzi di un programma possono vedere (e usare) una variabile.
- Valore iniziale: il valore della variabile all'inizio dell'esecuzione del programma.

Il “tipo” di una variabile

La codifica utilizzata per rappresenare i dati nella memoria occupata da una variabile si dice tipo. In C sono disponibili diversi tipi predefiniti:

- int, unsigned int: intero con o senza segno, 4 bytes.
- float: floating point, 4 bytes.
- double: floating point, 8 bytes.
- char: carattere alfanumerico, 1 byte.

Il programmatore deve essere sicuro che tutte le parti di un programma che utilizzano una data variabile siano d’accordo sul suo tipo; in caso contrario i risultati saranno imprevedibili.

TIPI aritmetici e loro RANGE

Interi

TIPO	BIT	BYTE	Contenuto	Range
char	8	1	Carattere ASCII	-127 – 127
unsigned char	8	1		0 - 255
short int	16	2	intero con segno	-2^15 - 2^15-1
int	32	4	intero con segno	-2^31 - 2^31-1
long int	32	4	intero con segno	-2^31 - 2^31-1
unsigned short	16	2	intero senza segno	0 - 2^16-1
unsigned int	32	4	intero senza segno	0 - 2^32-1
unsigned long	32	4	intero senza segno	0 - 2^32-1

Floating-point

TIPO	BIT	BYTE	Valore minimo (non standard)	Valore massimo (non standard)
float	32	4	1.1755 10^-38	3.403 10^38
double	64	8	2.2251 10^-308	1.7977 10^308
long double	80	10	3.3621 10^-4932	1.1897 10^4932

Il codice ASCII

Per rappresentare il testo si usa un codice, detto codice ASCII, in cui a ogni combinazione di otto bits (1 byte) si fa corrispondere un simbolo; ad esempio:

"A" = 65	"a" = 97
"B" = 66	"b" = 98
"0" = 48	"(" = 40
"1" = 49	"+" = 43

Ci sono ovviamente 256 simboli possibili, solo in parte occupati da lettere, numeri e simboli vari.

Un insieme di caratteri ASCII si dice stringa.

Variabili Logiche

Le variabili logiche possono assumere solo i valori "vero" o "falso". In C++ si possono usare variabili intere come variabili logiche, con la convenzione che

0 = falso 1 = vero.

Le espressioni logiche sono comunque molto usate; nel comando:

```
if (a>5) {
  ...
}
```

a>5 un espressione logica che vale 0 o 1 (cioè

```
int i;
i = (a>5);
```

assegna a i il valore 0 o 1 a seconda del valore di a).

Esistono poi vere e proprie variabili logiche, di tipo bool, che possono assumere i valori true e false.

```
bool test = true;
...
if (test && a==0) {
    ...
}
```

Istruzione sizeof

L'istruzione **sizeof(espressione)**

restituisce un intero che corrisponde al numero di byte occupati in memoria dall'espressione tra le parentesi.

E' molto utile nella scrittura dei programmi per calcolare lo spazio di memoria occorrente.

Provate le seguenti istruzioni:

```
cout << sizeof(double) ;

cout<<  sizeof(char);

cout<< sizeof(bool);
```

Dimensione di una variabile

E' possibile definire collezioni di variabili con uno o più indici, che possono essere manipolate come singoli oggetti nei programmi. In analogia con la notazione matematica, una variabile ad un indice si dice vettore, una a due o più indici matrice.

Il vantaggio dei vettori rispetto a gruppi di n variabili distinte, e che gli elementi di un vettore possono essere acceduti uno a uno per mezzo di un'altra variabile. Ad esempio:

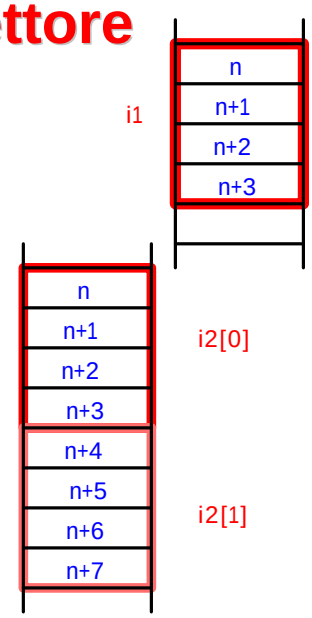
```
int i;
float dati[10];
for (i=0; i<10; i++) {
   cout << "Il dato numero " << i << " vale " << dati[i] << endl;
}
```

Gli indici vanno da 0 a n-1

Posizione in memoria degli elementi di un vettore

Uno scalare occupa sempre n bytes contigui in memoria (4 nel caso di un int);

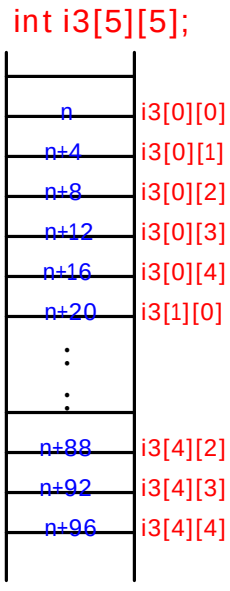
Un vettore occupa un insieme di m bytes contigui, dove m = n*size, n è la lunghezza del vettore e size è la dimensione di un elemento.



Anche in un vettore bi-dimensionale le variabili sono allocate in memoria in modo contiguo, facendo scorrere più rapidamente l'indice più a destra.

E' bene, quando si eseguono loop, rispettare l'ordine naturale delle matrici:

```
int a[1000][1000], b[1000][1000];
for (i=0; i<1000; i++) {
  for (j=0; j<1000; j++) {
    a[i][j] = a[i][j] + b[i][j]; /* corretto */
    /* a[j][i] = a[j][i] + b[j][i];  inefficiente */
  }
}
```



Dichiarazione delle variabili in C++

In C++, ogni variabile utilizzata deve essere dichiarata esplicitamente prima di venire utilizzata.

La linea che dichiara la variabile ne specifica, accanto al tipo, anche la dimensione ed eventualmente il valore iniziale. Ad esempio:

```
int i1, i2[10], i3[5][5];
float test = 0.24;
unsigned int counter[3] = { 1, 2, 3 };
string hello = “Ciao”;
```

Costanti

In C++ si possono realizzare diversi tipi di costanti:

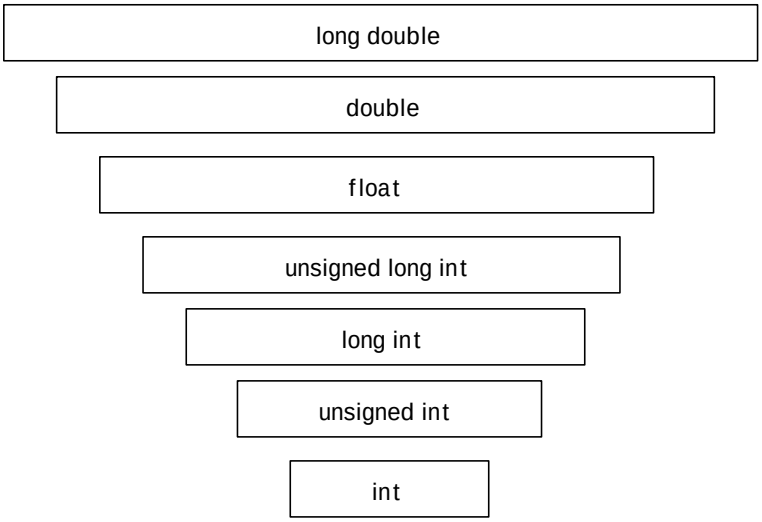
- Costanti (alfa) numeriche esplicite, come nelle istruzioni

```
double C = 5./9.*(F-32);
char c = ‘x’;
stringa += “ coda”;
```

- Simboli definiti a livello di precompilazione, che vengono sostituiti con il valore numerico al momento della compilazione:

```
#define COMPONENTS 12
double a[COMPONENTS];
```

Espressioni con diversi tipi di dati - gerarchia



Conversioni implicite e casting

Abbiamo visto che in alcuni casi il compilatore esegue delle conversioni implicite tra tipi diversi.

Ad esempio, se a è un int e b un float, allora se si scrive b=a il compilatore convertirà implicitamente a in un float di ugual valore e lo assegnerà a b.

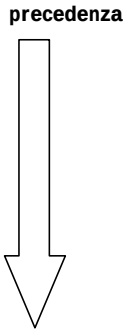
In alcuni casi è necessario richiedere esplicitamente una conversione, il che si ottiene specificando tra parentesi, prima dell'espressione, il tipo in cui l'espressione deve essere convertita:

```
int a; float b;
b = (float)a/5;
```

Attenzione alle operazioni tra costanti:

```
a=1/5;        // operazione tra due costanti intere !
cout << a;    // dà come risultato .....
```

Gerarchia degli OPERATORI
(esistono altri operatori, guardate sui manuali)



•**moltiplicazione e divisione**	***,/**
•**somma e sottrazione**	**+,-**
•**relazionali**	**<,<=,>,>=**
•**uguaglianza**	**==,!=**
•**AND logico**	**&&**
•**OR logico**	**||**
•**NOT logico**	**!**
•**assegnazione (*)**	**=**

Per la valutazione di una espressione vengono eseguite prima le operazioni con precedenza più alta, a parità di precedenza le espressioni che contengono questi operatori (escluso l'assegnazione) sono valutate da sinistra a destra.

Esempio:

a=b+c-d; è equivalente a **a=(b+c)-d;**

a=b=c; è invece equivalente a **b=c;**
a=b;

"Scope" delle variabili

Lo "scope" delle variabili in programmazione corrisponde allo loro visibilità da parte dei vari moduli (functions) che compongono un programma.

Le variabili hanno, se non specificato diversamente, solo significato nella funzione (subroutine) nella quale sono definite e modificate. Quindi, in generale, si dice che una variabile è locale.

Per passare le informazioni da una funzione all'altra esistono due metodi:

1. Il passaggio dei parametri
2. Definire le variabili al quale devono accedere più funzioni come globali.

Esempio di variabili locali e globali

```
#include <iostream>
double in, out;
quadrato() {
        out = in*in;
}
main() {
        cin >> in;
        quadrato();
        cout << out;
}
```

in e out sono variabili globali, ovvero visibili da tutte le functions del programma.

quadrato() è una function che agisce sulle variabili in ed out.

main() è una function che legge da tastiera il valore di in, chiama la function quadrato() e scrive su terminale il valore di out.

Allocazione statica e automatica

La più semplice allocazione di memoria si dice statica: una variabile viene associata ad una locazione fissa all'inizio del programma e lì rimane per tutta l'esecuzione. Questa strategia comporta alcuni problemi:

- La memoria rimane sempre allocata anche se la variabile viene usata solo per brevi periodi.
- È necessario fissare a priori la dimensione delle variabili perché il compilatore deve sapere già al momento della compilazione quanto spazio riservare.

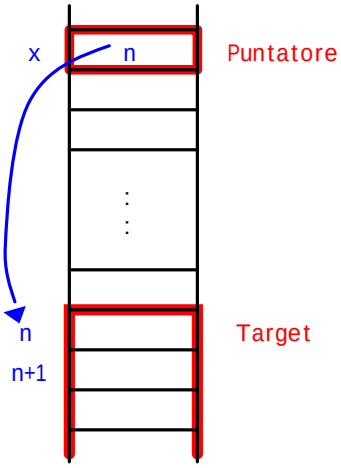
Il primo problema viene risolto dall'allocazione automatica: il compilatore fa in modo che la variabile venga allocata all'inizio del blocco che la usa e buttata via quando le istruzioni del blocco sono terminate. Le variabili all'interno di un blocco sono sempre allocate in modo automatico, a meno che non siano dichiarate con l'opzione static (es. "static float xx;").

Il concetto di puntatore

Il puntatore è lo strumento con cui si realizza l'allocazione dinamica della memoria. Si tratta di una variabile intera il cui valore rappresenta l'indirizzo di un'altra variabile (target).

Siccome il puntatore è una variabile, il suo contenuto può essere cambiato durante l'esecuzione del programma, il che equivale a modificare la posizione in memoria del target.

Inoltre un puntatore può puntare a scalari o vettori di qualunque dimensione, il che consente di modificare dinamicamente la dimensione del target.



Uso dei puntatori nei linguaggi

Per poter utilizzare la tecnica del puntatore è necessario che il linguaggio di programmazione fornisca due strumenti:

- Deve essere possibile costruire, a partire dal nome del puntatore, un riferimento simbolico al target.
- Deve essere disponibile uno strumento che consenta al programmatore di eseguire in modo esplicito l'allocazione di una porzione di memoria che costituirà il target di un puntatore.

Implementazione dei puntatori in C++

In C++ il puntatore è una variabile con un suo nome. Esiste un operatore * che, posto davanti al nome di un puntatore, simboleggia il target. Esiste anche un operatore & che data una variabile ne specifica l'indirizzo in memoria.

La sintassi per la definizione dei puntatori è del tipo:

```
int *a;     /* a è un puntatore a int */
float *b    /* b è un puntatore a float */
```

I puntatori sono specifici di ogni tipo, e possono essere usati per scalari o vettori. Ad esempio, se p1 e p2 sono puntatori

*p1 è il target di p1

p2[7] è l'ottava componente del vettore puntato da p2

p1[0] coincide con *p1

Se f è un vettore allocato staticamente, il nome senza parentesi è il puntatore a quel vettore.

Inizializzazione dei puntatori

Definire un puntatore **NON** significa associargli alcun target. Di conseguenza un puntatore appena creato è del tutto inutilizzabile in quanto non punta a niente di sensato. E' però buona norma assegnare ai puntatori un valore iniziale fisso (di solito 0 o il simbolo NULL) in modo da poter usare questo valore per testare se il puntatore è associato o meno a un target.

```
int *pt = NULL;
...
if (pt != NULL) {
   /* il puntatore pt ha un target definito */
}
```

Associazione a una variabile statica

Un puntatore può essere associato a una variabile statica (in principio anche a una automatica, ma la cosa è molto rischiosa perché quando la variabile viene distrutta il puntatore rimane definito in modo errato).

```
int *pt=NULL, i1, v1[10];
...
pt = &i1; /* da questo momento *pt coincide con i1 */
...
pt = v1; /* pt[i] coincide con v1[i] */
...
```

Allocazione dinamica

Con "allocazione dinamica" si intende la creazione di una variabile durante l'esecuzione. Ciò consente di creare variabili di dimensione voluta, pilotata ad esempio da un input ricevuto dal programma, o di modificare dinamicamente la dimensione di vettori o strutture.

In C++ le variabili allocate dinamicamente vengono accedute tramite puntatori, e lo spazio in memoria viene riservato con la funzione di sistema new e rilasciato quando non è più necessarie con la funzione delete.

Uso di new e delete

new è una funzione di sistema che ricerca in memoria un area di dimensioni specificate implicitamente dal tipo e dalla eventuale dimensione specificati di seguito, e ritorna l'indirizzo iniziale di tale area (o NULL se non c'è spazio in memoria).

La memoria necessaria sarà data in generale dal prodotto della dimensione della variabile da allocare per la dimensione del singolo elemento. Il calcolo è comunque eseguito in modo automatico.

La memoria allocata con new deve essere resa nuovamente disponibile al sistema quando non serve più. Per fare questo si chiama la funzione di sistema **delete addr** dando come parametro l'indirizzo di inizio dell'area. Se la memoria è stat allocata come un vettore si userà **delete[]**.

Vediamo come allocare dinamicamente un vettore di double:

```
int dim;
double *vect;
...
cout << "Dimmi la dimensione del vettore : ";
cin >> dim;
...
vect = new double[dim];
if (vect != NULL) {
    int i;
    for (i=0; i<dim; i++) {
       vect[i] = ...
    }
    ...
    delete[] vect;
}
```