Linguaggio C++

Primi passi

Computer e programmazione

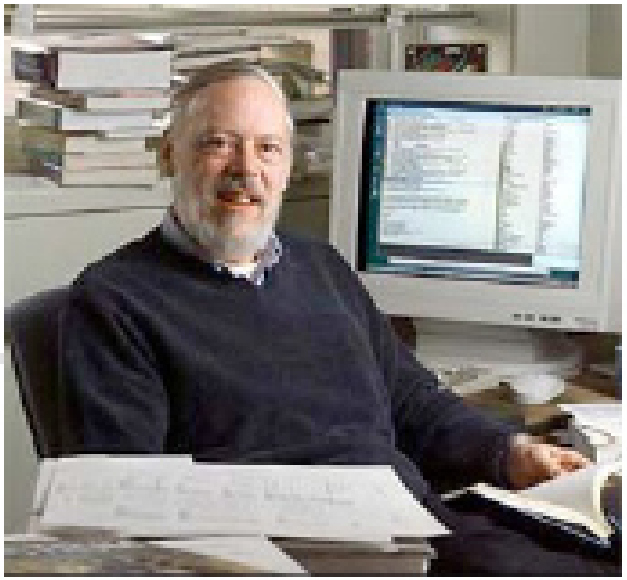
Se volessimo definire cos'è un computer, potremmo dire che un *computer* è una macchina elettronica *programmabile* al fine di svolgere diverse funzioni. La caratteristica fondamentale che distingue un computer, ad esempio, da una calcolatrice, è appunto la sua *programmabilità*: avendo in ingresso un programma e dei dati, un computer produce risultati secondo il procedimento di trasformazione descritto dal programma.

Equivalentemente, possiamo definire un computer come una macchina in grado di eseguire un processo computazionale sulla base di regole specificate tramite un *programma*. Il programma è costruito a partire da istruzioni elementari che il computer è in grado di comprendere ed eseguire. L'insieme di tali istruzioni è generalmente piuttosto ristretto; la potenza e la vasta applicabilità dei computer è data dalla capacità di operare in maniera estremamente veloce ed accurata. Combinando azioni elementari in lunghe sequenze di istruzioni è possibile far eseguire a un computer compiti complessi.

La *programmazione* è l'attività che consiste nell'organizzare istruzioni elementari, direttamente comprensibili dall'esecutore, in strutture complesse, i *programmi*, al fine di svolgere determinati compiti. Quando introdurremo il concetto di *compilatore* vedremo che, fortunatamente, non è necessario scrivere programmi nel linguaggio, estremamente povero, comprensibile dal processore di un computer, ma possiamo utilizzare linguaggi ad alto livello, comprensibili da esecutori *astratti*.

Linguaggio C: un po' di storia

- Fu ideato nei Bell Laboratories della AT&T nel 1972 da Dennis Ritchie come evoluzione del linguaggio B di Ken Thompson.
- La potenza e flessibilità del C apparve subito evidente e per questo il sistema operativo di Unix, scritto in assembly, venne riscritto immediatamente in C..
- La definizione formale si ha nel 1978 a cura di B. W. Kernighan e D. M. Ritchie (K&R).
- Nel 1983 iniziò il lavoro di definizione di uno standard da parte dell'American National Standards Institute, che rilasciò nel 1990 lo Standard ANSI C (ISO C89).



Perché il C

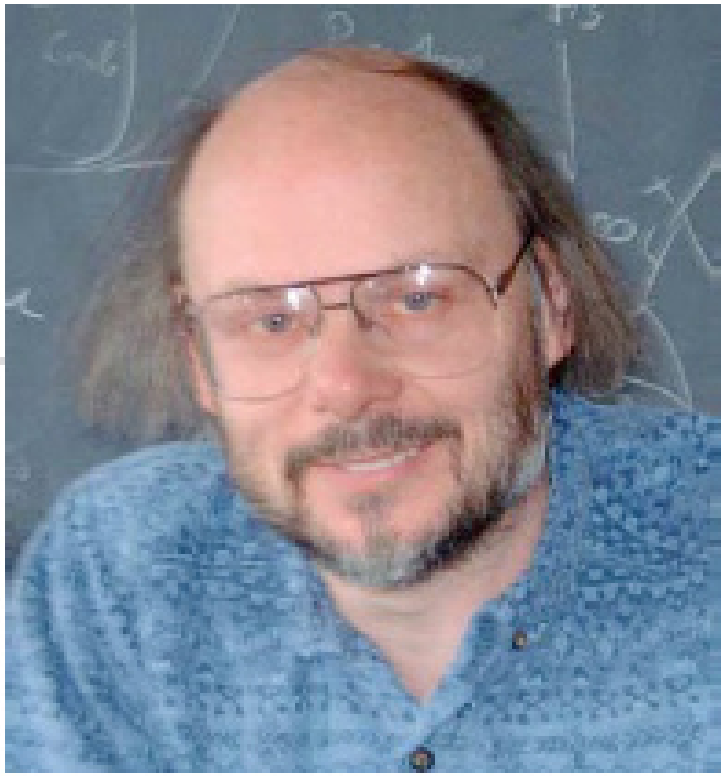
- E’ un linguaggio diffuso in ambito professionale
- E’ un linguaggio general purpose (cioè ci si può produrre qualunque tipo di applicazione)
- Permette di sviluppare programmi efficienti
- E’ il linguaggio in cui è praticamente stato scritto il S.O. Unix
- E’ un linguaggio ad alto livello

Alcune caratteristiche

- Linguaggio strutturato, imperativo
- Usabile anche come linguaggio di sistema: adatto a s/w di base (sistemi operativi, compilatori, …)
- Portabile, efficiente, sintetico (ma a volte poco leggibile)

Linguaggio C++: un po’ di storia

- Il C++ fu "inventato" nel 1980 dal ricercatore informatico danese **Bjarne Stroustrup**, che ricavò concetti già presenti in precedenti linguaggi (come il Simula67) per produrre una versione modificata del C, che chiamò: "C con le classi". Il nuovo linguaggio univa la potenza e l'efficienza del C con la novità concettuale della programmazione a oggetti.
- Il nome C++ fu introdotto per la prima volta nel 1983, per suggerire la sua natura evolutiva dal C, nel quale ++ è l'operatore di incremento (taluni volevano chiamarlo D, ma C++ prevalse, per i motivi detti).



Linguaggio C++: un po' di storia

- All'inizio, comunque, e per vari anni, il C++ restò un esercizio quasi "privato" dell'Autore e dei suoi collaboratori, progettato e portato avanti, come egli stesso disse, "per rendere più facile e piacevole la scrittura di buoni programmi".
- Tuttavia, alla fine degli anni 80, risultò chiaro che sempre più persone apprezzavano ed utilizzavano il linguaggio e che la sua standardizzazione formale era un obiettivo da perseguire. Nel 1990 si formò un comitato per la standardizzazione del C++, cui ovviamente partecipò lo stesso Autore. Da allora in poi, il comitato, nelle sue varie articolazioni, divenne il luogo deputato all'evoluzione e al raffinamento del linguaggio.
- Finalmente l'approvazione formale dello standard si ebbe alla fine del **1997**. In questi ultimi anni il C++ si è ulteriormente evoluto, soprattutto per quello che riguarda l'implementazione di nuove classi nella libreria standard.

Caratteristiche

- Espansione linguistica del C: modifiche non strutturali al linguaggio.
- Le classi: introduzione dei tipi di dati astratti definibili dal programmatore e utilizzabili come quelli predefiniti.
- OOP: implementazione dei meccanismi propri della programmazione orientata agli oggetti.

Introduzione

- Alfabeto:
  - A..Z, a .. z (26 + 26 caratteri)
  - 0 .. 9 (10 caratteri)
  - underscore: _
  - blank: ‘ ’
  - caratteri speciali: !, #, %, &, *, …..
- E’ un linguaggio “case sensitive” (maiuscole ≠ minuscole)

Introduzione

- Gestione errori:
  - E: errore
  - W: avvertimento
- Blocchi di codice: { …..}
- Il punto e virgola è il simbolo di fine istruzione.
- Commenti:
  - /* anche su più
    righe */
  - //su una sola riga
- indentazione

Introduzione

- Parole chiave/riservate (minuscolo)
- Identificatori di variabili, costanti, … . Utilizzare nomi significativi. Ricordare le regole di costruzione e le convenzioni di scrittura.
- Spazi e parentesi $\Rightarrow$ pgm più leggibili

Struttura di un programma C/C++

```
//Inclusione file (direttive al preprocessore)
//Dichiarazioni/definizioni di costanti, tipi e variabili
//Dichiarazione di funzioni (prototipi)
int main(){
//Dichiarazioni/definizioni di costanti, tipi e variabili
    //Istruzione1;
    //Istruzione2;
    …..
    //istruzioneN;
    return 0;
}
//Definizioni funzioni
```

Primo programma in C

```
#include <stdio.h>

int main() {
    printf("Hello world\n");
    return 0;
}
```

Primo programma in C++

```
#include <iostream>

using namespace std;

int main() {
    cout<<"Hello world\n“;
    return 0;
}
```

Osservazioni

- PGM C/C++ = {F_0, F_1, F_2, …, F_n}
  dove:
- F_0 = funzione principale: main()
- F_i = funzioni di libreria (standard) +
  funzioni utente

Osservazioni

- Tutti i pgm C/C++ devono contenere una funzione main(), che segnala l'inizio dell'esecuzione.
- L'esecuzione del pgm termina alla fine della funzione main().

Tipi e variabili

- Una variabile prima di essere utilizzata deve essere definita.
- Accanto al nome della variabile devo indicare anche il suo tipo.
- Il tipo di una variabile è un attributo che specifica:
  - Un insieme di valori
  - Un insieme di operazioni/funzioni
  - L'implementazione in memoria

Tipi fondamentali

- char
- int
- float
- double
- bool

Unità di misura di memoria

bit (binary digit)= 0/1

byte = 8 bit

1Kb = 2^{10} byte

1Mb = 2^{20} byte

1Gb = 2^{30} byte

1Tb = 2^{40} byte

16-bit data types, sizes, and ranges

(Dove E = 10 elevato a ...)

Type	Size (bits)	Range	Sample applications
unsigned char	8	0 to 255	Small numbers and full PC character set
char	8	-128 to 127	Very small numbers and ASCII characters
enum	16	-32,768 to 32,767	Ordered sets of values
unsigned int	16	0 to 65,535	Larger numbers and loops
short int	16	-32,768 to 32,767	Counting, small numbers, loop control
int	16	-32,768 to 32,767	Counting, small numbers, loop control
unsigned long	32	0 to 4,294,967,295	Astronomical distances
long	32	-2,147,483,648 to 2,147,483,647	Large numbers, populations
float	32	3.4E-38 to 3.4E38	Scientific (7-digit precision)
double	64	1.7E-308 to 1.7E308	Scientific (15-digit precision)
long double	80	3.4E-4932 to 1.1E4932	Financial (18-digit precision)

32-bit data types, sizes, and ranges

N.B. La dimensione dei tipi dipende dalle carateristiche del processore della macchina

Type	Size (bits)	Range	Sample applications
unsigned char	8	0 to 255	Small numbers and full PC character set
char	8	-128 to 127	Very small numbers and ASCII characters
short int	16	-32,768 to 32,767	Counting, small numbers, loop control
unsigned int	32	0 to 4,294,967,295	Large numbers and loops
int	32	-2,147,483,648 to 2,147,483,647	Counting, small numbers, loop control
unsigned long	32	0 to 4,294,967,295	Astronomical distances
enum	32	-2,147,483,648 to 2,147,483,647	Ordered sets of values
long	32	-2,147,483,648 to 2,147,483,647	Large numbers, populations
float	32	3.4E-38 to 1.7E38	Scientific (7-digit) precision)
double	64	1.7E-308 to 3.4E308	Scientific (15-digit precision)
long double	80	3.4E-4932 to 1.1E4932	Financial (18-digit precision)

Modificatori di tipo

- **signed** / **unsigned** (numeri con segno (per default) o senza segno (guarda il range in entrambi i casi))
- **short** / **long** (dimunisce (o resta uguale) o aumenta il range (o resta uguale))

Esempi

- int count;
- unsigned int count;

Inizializzazione di una variabile

- int count = 0;

Istruzione di assegnazione

Esempio n.1:

```
int count;
    .....
    count = 0;
```

Esempio n.2:

```
int area;
int base;
int altezza;
    .....
    area = base * altezza;
```

Costanti

```
#define BASE 10
```

oppure

```
const int BASE = 10;
```

```
#define PI_GRECO 3.14
```

Convenzione: usare solo caratteri maiuscoli più il carattere di underscore per i nomi composti.

Operazioni standard di input e output in C++

- Operazioni di output (esempi) #include<iostream>
  - cout<< area;
  - cout<<“Il numero non è primo”;
  - cout<<“Digita il raggio\n”;
  - cout<<“Digita il raggio”<<endl;
  - cout<<“Il volume della sfera e’ ”<<volume;
  - cout<<base<<altezza<<area;
  - cout<<“Base = “<<base<<“Altezza = “<<altezza<<“Area = “<<area<<endl;

Operazioni standard di input e output in C++

- Operazioni di input (esempi) #include<iostream>

- cin>>base;
- cin>>base>>altezza;

Istruzioni di I/O in C

Standard input → Tastiera

Standard output → Video

printf() → * è una funzione della libreria standard

* invia il risultato del pgm al video

Esempi:

printf(“CIAO”); → CIAO

printf(“%d %d %d”, base, altezza, area); → 10 13 130

printf(“Area: %d”, area); → Area: 130

printf(“Area: %d”, base*altezza); → Area: 130

Istruzioni di I/O in C

printf(“%d\n%d\n%d\n”,base, altezza, area);

Output:
10
13
130

\n → è un carattere → ‘\n’ → carattere di new line

printf(“\nCorso di C \n\n”);

1^ riga
2^ riga Corso di C
3^ riga
4^ riga

Istruzioni di I/O in C

In generale:

printf(“stringa di formato”, elenco parametri); //stampa formattata

La stringa di formato stabilisce i formati con i quali “qualcosa” deve essere stampato. C’è sempre.
L’elenco dei parametri o argomenti della funzione (printf()) può anche non esserci.

Alcuni codici di formato:

%d → int
%u → unsigned int
%c → char
%s → stringa
%ld → long [int]
%f → float, double
%o → ottale (unsigned int)
%x → esadecimale (unsigned int)
…..

E’ indispensabile che il numero di argomenti o parametri sia uguale a quello dei codici di formato.

printf() → è un esempio di funzione con la lista di argomenti di lunghezza variabile

Istruzioni di I/O in C

- Formattazione dell'output
  - printf("%5d%5d%5d", base, altezza, area);
  - printf("%-5d%-5d%5d", base, altezza, area);
- Alcune sequenze di escape
  - \n → va a nuova linea
  - \t → salta di un tabulatore
  - \b → ritorna un carattere indietro (backspace)
  - \a → fa emettere un beep alla macchina
  - \\ → stampa il carattere \
  - \" → stampa il carattere "

Istruzioni di I/O in C

- Per usare la funzione d libreria printf() devi scrivere:
- #include <stdio.h> //.h → header file
- E' una direttiva che viene data al preprocessore: essa segnala al preprocessore di includere nel pgm sorgente il file "stdio.h" prima di effettuare la compilazione.
- Questo file, che viene incluso, contiene informazioni indispensabili al pgm per poter garantire il corretto funzionamento delle funzioni di I/O contenute nella libreria standard del C.

Istruzioni di I/O in C

scanf() → * è una funzione della libreria standard

* legge i dati immessi dalla tastiera

scanf("stringa di formato",elenco argomenti);

Alcuni codici formato:

%f → float

%lf → double

….

Istruzioni di I/O in C

- Tutte le variabili utilizzate per ricevere valori con scanf() vanno passate per indirizzo.

```
int base;
scanf("%d",&base);
```

- Il carattere & indica l'indirizzo di memoria in cui si trova la variabile di nome base.

```
scanf("%d%d", &base,&altezza);
```

- Nella stringa di formato non mettere spazi, ma solo codici di formato.

Funzioni (di libreria)

- Una funzione è costituita da un insieme di istruzioni che realizzano un particolare compito.
- Esempio: abs()
  - Funzione svolta: calcola il valore assoluto
  - Input: numero intero
  - Output: valore assoluto del numero

Funzioni (di libreria)

- Esempi di utilizzo della abs():
  - valoreAss = abs(num);
  - cout<<abs(num); //in C++
  - printf("%d",abs(num)); //in C (%d, codice di formato per il tipo int)
  - var = num2 * abs(num) + 4;

Funzioni (di libreria)

- Un altro esempio di funzione:
- sqrt()
  - Funzione svolta: calcola la radice quadrata
  - Input: numero reale (≥ 0)
  - Output: radice quadrata del numero
- Gli esempi di utilizzo sono uguali a quelli della funzione abs() (usando la printf() si deve cambiare il codice formato da %d (si usa per int) a %f (si usa per double))

Funzioni (di libreria)

- Per utilizzare correttamente una funzione di libreria devo conoscere il suo prototipo (dichiarazione della funzione):
  - int abs(int num);
  - double sqrt (double num);
- I prototipi delle funzioni di libreria sono contenuti negli header file che devo includere all'inizio di ogni pgm.

- abs() → #include <cmath> in C++
- sqrt() → #include <math> in C++

Funzioni (di libreria)

- Una funzione (di libreria) viene vista come una scatola nera (**black box**).
- Come programmatori possiamo utilizzare una funzione di libreria come una **scatola nera** cioè senza conoscere alcunché del suo funzionamento interno: ci interessa solo sapere cosa passarle in ingresso e cosa restituisce in uscita.
- Attraverso il prototipo della funzione (interfaccia) sappiamo cosa fa e come interagisce con l'esterno ma non come lo fa.

Funzioni (di libreria)

- Riepilogando, relativamente l'utilizzo delle funzioni, sono interessato a conoscere:
  - che cosa fa (non come lo fa)
  - i dati in input
  - il dato in output

- Esempio: sqrt()
  - Funzione svolta: calcola la radice quadrata
  - Input: numero reale (≥ 0)
  - Output: radice quadrata del numero