Introduzione alla programmazione C

Struttura di un programma

Iniziamo esaminando il programma del seguente listato.

```
#include<stdio.h>
main()
{
printf("abc");
printf("def");
printf("ghi");
printf("lmn");
printf("opqrs");
printf("tuvz");
}
```

Eseguendolo verrà visualizzata la stringa delle lettere dell'alfabeto italiano:

abcdefghilmnopqrstuvz

Dalla parola main, seguita da parentesi tonda aperta e chiusa, inizia l'esecuzione del programma. Il corpo del programma, che comincia dalla parentesi graffa aperta e finisce alla parentesi graffa chiusa, è composto da una serie di istruzioni printf che verranno eseguite sequenzialmente. L'istruzione printf permette la stampa su video di ciò che è racchiuso tra parentesi tonde e doppi apici. Per esempio:

```
printf("abc");
visualizza
abc
```

Ogni istruzione deve terminare con un carattere di punto e virgola.

Per poter utilizzare printf, come le altre funzioni di entrata/uscita, si deve inserire all'inizio del testo la linea

```
#include <stdio.h>
```

che avverte il compilatore di includere i riferimenti alla libreria standard di input/output (stdio sta per standard input/output). Il C distingue tra lettere maiuscole e minuscole; dunque occorre fare attenzione, se si scrive MAIN() o Main() non si fa riferimento a main().

La struttura del programma C che abbiamo usato nell'esempio è:

```
inclusione_librerie
main()
{
istruzione1
istruzione2
istruzione3
.....
istruzioneN
}
```

Il punto e virgola conclude l'istruzione. Se si desidera che l'uscita di ogni istruzione printf venga prodotta su una linea separata, si deve inserire \n al termine di ogni stringa e prima della chiusura dei doppi apici, come nel seguente listato.

```
#include <stdio.h>
main()
{
printf("abc\n");
printf("def\n");
printf("ghi\n");
printf("lmn\n");
printf("opqrs\n");
printf("tuvz\n");
}
```

Eseguendo il programma si otterrà la visualizzazione delle seguenti stringhe di caratteri:

```
abc
def
ghi
lmn
opqrs
tuvz
```

In questo caso la prima stringa (abc) viene stampata su video a partire dalla posizione attuale del cursore. Se si vuole cominciare la stampa delle stringhe da una nuova riga basta inserire \n anche all'inizio come in:

```
printf("\nabc\n");
```

Qui prima si passa ad una nuova riga, poi si stampa la stringa specificata e quindi si posiziona il cursore in una nuova riga. In generale è bene tenere presente che ogni printf stampa a partire dalla posizione in cui si trovava il cursore (in generale a destra dell'ultima stampa). Per poter modificare tale comportamento è necessario inserire gli opportuni caratteri di controllo. In effetti la sequenza \n corrisponde ad un solo carattere, quello di nuova linea (newline).
Nella printf possono inserirsi altri caratteri di controllo di cui si forniscono di seguito quelli che possono avere utilizzo più frequente:

- \n porta il cursore all'inizio della riga successiva
- \t porta il cursore al prossimo fermo di tabulazione (ogni fermo di tabulazione è fissato ad 8 caratteri)
- \a (alert) fa emettere un suono dallo speaker
- \' stampa un apice
- \" stampa le virgolette

Le variabili

Dunque le variabili in C devono sempre essere dichiarate prima di essere usate (prima del "main ( )"). Esistono vari tipi di variabili, ecco le più comuni:

- Character: va da -128 fino a 127 e viene dichiarata: char
- Unsigned character: va da 0 fino a 255, e viene dichiarata: unsigned char
- Integer: va da -32768 fino a 32767 e viene dichiarata: int
- Unsigned integer: va da 0 fino a 65535, e viene dichiarata: unsigned int
- Long integer: va da -2147483648 fino a: 2147483647 e viene dichiarata: long int o solo long
- Unsigned long integer: va da 0 a 4294967295 e viene dichiarata: unsigned long int o solo unsigned long
- Floating point: va da 3.4*10-38 a 3.4*1038 e viene dichiarata: float
- Double precision: va da 1.7*10-308 a 1.7*10308 e viene dichiarata: double
- Long double precision: va da 3.4*10-4932 a 1.1*104932 e viene dichiarata: long double

%u = indica un intero senza segno

%f = indica un dato di float

%d = indica un intero con segno

%s = indica una stringa

%c = indica un carattere

%x = un carattere in esadecimale scritto in piccolo

%X= un carattere in esadecimale scritto in grande

Facciamo un piccolo esempio di programma in C con le variabili:

```
#include <stdio.h>                                          /* 1 */
int Spirito;                                                  /* 2 */
main()                                                     /* 3 */
{                                                          /* 4 */
Spirito =          10;                                     /* 5 */
printf("mhhh che bel numero e' il  %d vero?\n",Spirito );  /* 6 */
}                                                          /* 7 */
```

Spiegazione:

- **/*1*/** Include la solita libreria "stdio.h"

- **/*2*/** Dichiariamo che la variabile Spirito è una variabile di tipo integer (ricordate il C è case-sensitive c'è una grande differenza tra Spirito e spirito ""NON SBAGLIATE!"").

- **/*3*/** La funzione main avvia il programma e non richiama altre funzioni.

- **/*4*/** Inizio della funzione main( )

- **/*5*/** Qui diciamo che la variabile Spirito(che è integer) assume il valore del numero 10

- **/*6*/** Allora iniziamo a dire che una volta avviato il programma stamperà questa frase:

" mhhh che bel numero e' il 10 vero? "

Comando define

Ora ogni volta che dobbiamo scrivere 16489254 basterà scrivere NG

Esempio:

```
#include <stdio.h>
#define co 994543
main( )
{
printf("eccote un numero %d ok?\n",co);
}
```

Naturalmente questo printf stamperà : *eccote un numero 994543 ok?*

Comando scanf

```
#include <stdio.h>
int  numero;
main( )
{
printf ("inserisci un numero\n");
scanf("%d",&numero);
printf(" il numero che  hai scelto e' %d .\n",numero);
}
```

Spiegazione:

come sicuramente avrete visto nella riga di scanf vi troviamo il %d che naturalmente vuol significare che deve leggere un numero , ha anche il " &numero ". Ciò vuol dire che il numero che noi abbiamo dato (8) lo salva nella variabile chiamata "numero" e che dopo lo stampa con l'opzione printf, chiaro no?.

I segni

<	*minore*
>	*maggiore*
<=	*minore\uguale*
>=	*maggiore\uguale*
=	*uguale*
!=	*diverso*

Comando switch

```
#include <stdio.h>
int a;
main()
{
printf ("inserisci il numero\n");
scanf  ("%d",&a);
switch (a)
{
   case 1:
       printf ("hai scelto il numero 1! \n");
    scanf("%d");
    break;
    case 2:
        printf("hai scelto il numero 2!!\n");
        scanf("%d");
        break;
        default:
       printf("non hai scelto ne il numero 1 ne il 2 ma hai scelto il numero:%d !!! \n",a);
   scanf ("%d");
}
}
```

Allora una cosa importante da osservare è la " a" tra parentesi dopo " switch " , la quale vuole indicare che in base al valore che abbiamo assegnato alla variabile " a " verranno effettuati i vari casi. Ad esempio se noi scriviamo 1, allora apparirà l'istruzione dichiarata in "case 1:" cioè il printf che dice: hai scelto il numero1!!! , se invece scriviamo il numero 2 allora eseguirà l'istruzione che abbiamo assegnato a " case 2: " (il printf che dice : hai scelto il numero 2!!), mentre se scriviamo numeri diversi da 1 e 2 verrà eseguita l'operazione di " default: " (la quale dice: non hai scelto ne il numero 1 ne il 2 ma hai scelto il(numero alla quale si ha assegnato alla variabile).

Comando while

```
#include <stdio.h>
int spir;
main()
{
printf(" ciao metti il valore di spir\n");
scanf ("%d",&spir);
if (spir<10)
  {
   while (spir<=10)
       {
    printf(" azz spir e' = %d ih ih\n",spir);
    spir++;
       }
   scanf("%d");
```

```
    }
else

            {
    printf(" spir è maggiore o uguale a 10!!\n");
    scanf ("%d");
            }

}
```

Beh oramai credo che gia avrete capito cosa faccia questo programmino comunque, prima ci chiede di inserire un numero, poi fa un controllo del numero che abbiamo inserito, se è un numero maggiore o uguale a 10 allora il prog ci risponderà che è maggiore o uguale al 10, altrimenti farà eseguire il comando while, il quale stamperà il valore della variabile " spir " e addizionerà la nostra variablie di + 1 fino a quando non sia maggiore o uguale a 10, ora vi faccio anche vedere l' output di questo programma :

```
ciao metti il valore di spir
4
azz spir e' = 4 ih ih
azz spir e' = 5 ih ih
azz spir e' = 6 ih ih
azz spir e' = 7 ih ih
azz spir e' = 8 ih ih
azz spir e' = 9 ih ih
azz spir e' = 10 ih ih
```

Funzioni

```
tipo_di_funzione   nome_funzione    ( dichiarazioni)
{
comandi
}
```

Esempio di una funzione semplice.

```
int add (int a , int b , int c )  /*1*/
{                                 /*2*/
int addi                          /*3*/
addi = ( a+b+c);                  /*4*/
return (addi);                    /*5*/
}                                 /*6*/
```

Spiegazione:

- 1) dichiariamo una funzione di tipo: integer (int); gli diamo il nome add e dichiariamo che la funzione ha delle variabili in entrata di tipo integer;
- 2) inizio della funzione "add";

- 3)dichiariamo una variabile (addi) di tipo integer;
- 4)diamo alla variabile (addi) il valore della somma delle tre variabili in entrata;
- 5)restituisce il valore della variabile (addi);
- 6) chiudiamo la funzione (add).

```
#include <stdio.h>                                  /*1*/
int add (int a, int b,int c)                       /*2*/
{                                                  /*3*/
int addi;                                          /*4*/
addi= (a+b+c);                                     /*5*/
return (addi);                                     /*6*/
}                                                  /*7*/
main()                                             /*8*/
{                                                  /*9*/
int a,b,c, risultato;                              /*10*/
printf ("inserisci il valore di a , b e c \n");    /*11*/
scanf ("%d",&a);                                   /*12*/
scanf("%d",&b);                                    /*13*/
scanf ("%d",&c);                                   /*14*/
risultato=add(a,b,c);                              /*15*/
printf("l'addizione dei tre numeri è %d\n",risultato);  /*16*/
scanf ("%d");                                      /*17*/
}                                                  /*18*/
```

Spiegazione:

1) la solita libreria...;

2) dichiariamo una funzione di tipo integer che ha nome (add) e 3 variabili in entrata(sa parte della funzione principale "punto:15");

3) inizio della funzione "add";

4) dichiariamo una variabile (addi) di tipo integer;

5) diamo alla variabile (addi) il valore della somma delle tre variabili in entrata;

6) restituisce il valore della variabile (addi) alla funzione principale(punto:15, e lo fa stampare al punto (16);

7) chiudiamo la funzione (add);

8) la solita funzione principale;

9) l'inizio della funzione principale;

10) dichiariamo 4 variabili di tipo integer;

11) stampa una richiesta;

12) aspetta che gli diamo il valore della variabile a;

13) aspetta che gli diamo il valore della variabile b;

14) aspetta che gli diamo il valore della variabile c;

15) dichiara il valore dell'addizione delle 3 variabili; richiama la funzione add e da come "le tre variabili in entrata" a,b,c la quale funzione gli restituisce il risultato della somma;

16) stampa a video il risultato della somma delle vriabili;

17) ci permette di leggere il risultato;

18) chiudiamo la funzione principale.

funzioni le quali non hanno ne parametri in entrata ne in uscita:

```
“   void  nome_funzione (void)   “
```

funzioni le quali hanno un solo parametro in entrata:

```
“    void  nome_fun  (int a)                “
“ {                                          “
“   printf (“ a e’ uguale a: %d\n”,a);   “
“  }                                         “
```

funzioni che hanno un parametro in uscita, ma non in entrata:

```
“    int nome_fun (void)              “
“   {                                  “
“  int a;                              “
“   a=1;                              “
“   return (a);                       “
“     }                               “
```

e naturalmente le funzioni che danno parametri sia in entrata che in uscita.... (come abbiamo visto nel programma).Solo per informarvi : “**void**” vuol dire nulla (in inglese).

Il linguaggio C dispone di un operatore speciale per incrementare una variabile di una unità. Scrivere:

```
contakm++;
```

equivale a scrivere:

```
contakm = contakm+1;
```

Cioè ad incrementare di una unità il valore della variabile contakm. L’operatore ++ è l’operatore di **autoincremento**. L’operatore reciproco -- (due simboli meno) decrementa di una unità il valore di una variabile:

```
altezza--; /* riduce l’altezza di 1 */
```

L’operatore -- è quindi l’operatore di **autodecremento**.
Si sono viste, quindi, due tecniche per aggiungere uno al valore contenuto in una variabile:

```
fogli = fogli+1;fogli++;Esiste anche una terza tecnica: fogli += 1;
```

La combinazione += è un esempio di **operatore di assegnazione**. L’istruzione:

```
km += 1;
```

si può leggere: aggiungi uno al valore corrente della variabile km. L’operatore += non è limitato ad aggiungere 1 ma somma il valore alla sua destra alla variabile alla sua sinistra. Ad esempio:

```
km += 37;k1 += k2;a += (b/2);
```

equivalgono rispettivamente a:

km = km+37;k1 = k1+k2;a = a+(b/2);

L'operatore += non è l'unico operatore di assegnazione, si possono usare tutti gli operatori aritmetici:

km - = 6; / toglie 6 ai km percorsi */*
lato **= 2; /** *moltiplica il lato per 2 */*
volume /= 3; / divide il volume per 3 */*

Per aggiungere uno alla variabile z si può scrivere in due modi:

z++;++z;

cioè mettere l'operatore ++ prima o dopo del nome della variabile.
In generale, le due forme sono equivalenti. La differenza importa solo quando si scrive una espressione che contiene z++ o ++z. Scrivendo z++, il valore di z viene prima usato poi incrementato:

int x,z; / due variabili intere */*
z = 4; / z vale 4 */*
x = z++; / anche x vale 4 ma z vale 5 */*

Difatti, prima il valore di z (4) è stato assegnato ad x, poi il valore di z è stato incrementato a 5.

Scrivendo ++z, il valore di z viene prima incrementato e poi usato:

int x,z; / due variabili intere */*
z = 4; / z vale 4 */*
x = ++z; / ora x vale 5 come z */*

Difatti, prima il valore di z (4) è stato incrementato a 5, poi il nuovo valore di z (5) è stato assegnato ad x.

Gli array

Array è un'insieme di variabili.Cosa vuol dire? Se io inizializzo una variabilie posso inserire un valore in quella variabile. Un array è un tipo di variabile che può accettare dentro se più variabili. Quindi una variabile semplice è come una casella. Un array è un insieme di caselle. Questi possono essere:

1 2 3 4 5 6 7 8 9 ... quindi su una sola linea.

1.1 2.1 3.1 4.1 ...
1.2 2.2 3.2 4.2 ...
1.3 2.3 3.3 4.3 ...
1.4 2.4 3.4 4.4 ...
...
... a tabella

Questi sono i tipi di array.Come si fanno a crearle? Così:
int numero[50]; ovviamente 50 è il numero di variabili che array contiene.
o a tabella:

int numero[30][30]; Anche 30 è un numero di mia invenzione.

Per utilizzarle basta fare:

```
numero[0]= 3;
numero[1]= 456675;
...
```

e così via dicendo.

Se io lo inizializzo come contenete 30 variabili, la priva sarà [0] e l'ultima [29].

Ciò vuol dire che si incomincia a contare a 0 e non da 1!

Comando if

```
if (numero>100){
printf("Il numero è maggiore di cento!!");
}
else {
printf("Il numero non è maggiore di cento!");
}

if (numero>50){
    printf("Il numero è maggiore di 50!");
    if (numero >100){
    printf("Ed è anche maggiore di 100!);
  }
    else{
    printf("Ma non è maggiore di 100!");
  }
}
else{
printf("Il numero è minore di 50!");
if (numero<0){
   prinf("Ed è anche minore di 0!");
   }
else{
       printf("Ma non è minore di 0!");
 }
}
```

Esistono infatti degli operatori detti binari. Questi sono:

= | Questo è "OR": if (A()) || (B()){... Basta che o A o B si verifichi. =
= && | Questo è "AND": if (A()) && (B()){... Devono essere vere entrambe. =
= ^ | Questo è "XOR": if (A()) ^ (B()){... Devono essere una vera e l'altra falsa =

Comando while

```
while (numero<10){
.... operazioni varie....
numero++;
}
```

L'unica differenza fra WHILE e DO e che DO fa il controllo dopo. Quindi se io faccio:

```
do {
...operazioni...
numero++;
} while(numero<10);
```

il controllo viene fatto dopo che le operazioni sono effettuate.

Comando for

Il for effettua un determinato numero di operazioni per un tot numero di volte.

Si usa così:

FOR (stato di inizio; stato di fine; aggiornamento){ ...

Che con un esempio è:

```
for (numero=0; numero<10; numero++){
...operazioni...
}
```

Esempio:

```
#include <stdio.h>
main(){
int spirito;
for (spirito=0;spirito<5;spirito++){
printf("La variabile vale %d",spirito);
  }
}
```

Stringhe

In C uno degli utilizzi degli array ad una dimensione riguarda la conservazione in memoria di stringhe di caratteri: queste in C sono array di tipo carattere la cui fine è segnalata da un carattere **null** (carattere terminatore), indicato come '\0'. Il carattere null è il primo codice ASCII corrispondente al valore binario 00000000 e non ha niente a che vedere con il carattere 0 che ha, in ASCII, codice binario 00110000.

char a[10]; dichiara un vettore costituito da un massimo di dieci caratteri e:

char frase[] = "Oggi sono felice";

dichiara l'array monodimensionale frase il cui numero di caratteri è determinato dalla quantità di caratteri presenti fra doppi apici più uno (il carattere null che chiude la stringa).

È importante notare la differenza tra le due assegnazioni:

char a = 'r';

char b[ ] = "r";

Nel primo caso viene assegnato il solo carattere r, nel secondo la stringa r\0. In definitiva: se si vuole fare riferimento ad un solo carattere, questo deve essere racchiuso fra apici singoli; se, invece, si vuole fare riferimento ad una stringa, occorre racchiuderla fra doppi apici.

Puntatori

Un puntatore si usa perchè così rendo più flessibile un programma. Lo rendo più adattabile a un sistema operativo piuttosto che ad un'altro. In più, con l'uso dei puntatori posso fare operazioni che altrimenti sarebbero letteralmente impossibili! E' ovvio il fatto che sono difficili da capire. Sono difficili da utilizzare, ma una volta entrati nell'ottica, il gioco è fatto.

Cos'è un puntatore? Un puntatore è un tipo di variabile che ha una esatta collocazione nella memoria. Questa variabile può contenere un'altra variabile o più.

Il puntatore può essere di qualsiasi tipo.

Per crearlo bisogna usare il comando:

```
int *puntatore
```

(ovviamente int può essere sostituito col tipo di variabile che noi vogliamo!)

Con il segno & io fornisco l'indirizzo di una variabile (vedi col comando scanf) mentre con il comando * io do il contenuto. Ora per semplificare il tutto vi beccate un esempio:

```
int *puntatore;
int x=1,y=2;
puntatore=&x; /*passaggio 1*/
y=*puntatore; /*passaggio 2*/
x=puntatore; /*passaggio 3*/
*puntatore=3; /*passaggio 4*/
```

Spiego:

Per capire cosa succede devo guardare come gestisce la macchina il tutto. Supponiamo che la variabile x sia allocata nello spazio di memoria 100 e quella y nella 200, il puntatore invece nella 1000. Con il passaggio 1 io faccio in modo che il puntatore si posizioni nello spazio di memoria 100, cioè quello della x. Con il secondo faccio in modo che la variabile y assumi il valore del puntatore, che, essendo collegato a x diventa 1. Quindi y vale 1. Con il terzo passaggio faccio assumere alla variabile x il valore del puntatore che è 100. Con l'ultimo passaggio, il 4 dico che il contenuto del puntatore è 3.

Possiamo quindi dire che i puntatori hanno tre tipi di informazioni:

- **puntatore** che mi indica il valore, il contenuto della variabile (considero il puntatore come una qualsiasi variabile)
- **&puntatore** che è l'indirizzo fisico della variabile all'interno della memoria
- ***puntatore** che mi indica il contenuto della locazione della memoria.

Un puntatore deve sempre essere allocato in una determinata zona della memoria. **QUESTO E' IMPORTANTISSIMO!!!** Se così fate il programma casca. Si blocca il sistema!

Quindi questo è un errore davvero grave!

```
int *errore;
*errore=100;
```

Mentre è giusto fare:

```
int *giusto;
int x;
giusto=&x;
*giusto=100;
```