Dispensa C

Qui trovate tutti gli argomenti trattati quest'anno.

Variabili

Sono dei contenitori che hanno un nome che decidiamo noi e permettono di salvare un valore.

Tipi di variabile

I tipi di variabile servono per dire al computer che cosa contiene una variabile, i tipi che esistono sono:

Tipo	Significato
int	Numero intero
float	Numero decimale
double	Numero decimale preciso
char	Carattere
bool	Valore logico (vero oppure falso)

Dichiarazione

Per utilizzare una variabile bisogna prima di tutto dichiararla, la dichiarazione segue queste regola:

```
tipo nome;
```

1. **tipo** : i tipi sono elencati qua sopra
2. **nome** : il nome della variabile deve essere
    - **UNICO** : non ci devono essere altre variabili con lo stesso nome
    - **SIGNIFICATIVO** : il suo nome deve aiutare a capire cosa contiene
    - Deve essere di sole lettere e numeri e _ (underscore), il nome di una variabile non può mai iniziare con un numero
    - Le lettere maiuscole sono diverse da quelle minuscole

Input e Output

L'input serve per permettere all'utente di interagire con il nostro programma: gli permettiamo di inserire dei valori che poi salviamo in delle variabili e utilizziamo.

L'output serve al nostro programma per "comunicare" con l'utente.

Output

L'output va fatto con la funzione *printf*, questa funzione permette di stampare SOLO DEL TESTO, per stampare il valore delle variabili bisogna seguire delle regole aggiuntive.

Stampare del testo

Il testo deve essere sempre il primo parametro passato alla funzione:

```c
printf("Stampo del testo");
```

Per andare a capo bisogna inserire il carattere **\n**:

```c
printf("Stampo del testo\n e vado a capo");
```

Stampare le variabili

Per stampare il valore di una variabile bisogna seguire delle regole come per andare a capo: bisogna inserire un simbolo nella stringa:

I simboli sono in base al tipo di variabile che si vuole stampare:

Tipo Variabile	Simbolo
int	%d
float	%f
double	%lf
char	%c
bool	%d

Per stampare ad esempio una variabile intera:

```
int variabile = 5;
printf("La variabile vale %d",variabile);
```

Il simbolo inserito viene sostituito con il valore contenuto nella variabile, quindi in questo caso il simbolo ***%d*** viene sostituito dal valore contenuto nella variabile, l'output risulta:

```
La variabile vale 5
```

Esempio: Stampare un menù

Per stampare un menù con opzioni numerate basta seguire uno di questi due esempi:

1. Con printf multipli:

```
printf("Scegli un'opzione:\n");
printf(" 1) Opzione 1\n");
printf(" 2) Opzione 2\n");
printf(" 3) Opzione 3\n");
//...
printf("Inserisci il numero:");
```

2. Con una sola printf:

```
printf("Scegli un'opzione:\n 1) Opzione 1\n 2) Opzione 2\n 3)
Opzione 3\nInserisci il numero:");
```

Input

L'input va fatto con la funzione *scanf*, questa funzione permette di leggere i valori INSERITI DALL'UTENTE e

salvarli in una variabili del vostro codice.

Salvare i valori

Per chiedere e salvare il valore in una variabile bisogna seguire delle regole:

La scanf deve seguire queste regole:

```
scanf("simbolo da usare", &variabile);
```

Dove:

- "simbolo da usare" dipende dalla lista qua sotto e deve essere messo tra virgolette
- Per dividere la stringa e la variabile si usa la virgola
- La variabile deve essere preceduta dalla "E commerciale": &

Si usano i simboli e in base al tipo di variabile che si vuole leggere si sceglie da questa tabella:

Tipo Variabile	Simbolo
int	%d
float	%f
double	%lf
char	%c
bool	%d

1. Per leggere ad esempio una variabile intera:

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

2. Per leggere ad esempio una variabile double:

```
double var;
scanf("%lf", &var);
```

Il simbolo inserito dipende da che valore dobbiamo inserire nella variabile, quindi in questo caso il simbolo ***%d*** viene usato per una variabile **int** mentre ***%lf*** per una variabile **double** come scritto nella tabella.

3. Per salvare ad esempio un valore in un array

```
double lista[10];
lista[5]=30.23;
printf("%lf",lista[5]);
```

Stampa il valore **double** che si trova alla posizione **5** dentro l'array ***lista***.

Esempio: Come costruire un menù

Per stampare un menù con opzioni numerate basta fare uno dei due metodi come descritti prima:

```
printf("Scegli un'opzione:\n");
printf(" 1) Opzione 1\n");
printf(" 2) Opzione 2\n");
printf(" 3) Opzione 3\n");
//...
printf("Inserisci il numero:");

int scelta;
scanf("%d",scelta);
```

In questo modo l'opzione scelta (cioà il numero inserito dall'utente) viene salvato nella variabile **scelta**.

Istruzioni di controllo

Le istruzioni di controllo in C consentono di influenzare il flusso di esecuzione di un programma. Queste istruzioni permettono di prendere decisioni, eseguire azioni ripetitive e gestire il flusso di esecuzione del programma in base alle condizioni specificate.

Ci sono tre tipi principali di istruzioni di controllo in C: selezione, iterazione e salto.

Istruzioni di selezione

Le istruzioni di selezione consentono di eseguire parti del codice solo se una determinata condizione è vera. Le istruzioni di selezione più comuni in C sono l'istruzione `if`, l'istruzione `else if` e l'istruzione `else`.

if

L'istruzione `if` permette di eseguire un blocco di codice solo se una condizione specificata è vera. La sintassi è:

```
if (condizione) {
    // Blocco di codice eseguito se la condizione è vera
}
```

else if

L'istruzione `else if` permette di specificare ulteriori condizioni da controllare nel caso in cui la prima condizione non sia vera. Può essere utilizzata dopo un'istruzione `if` o dopo un'altra istruzione `else if`. La sintassi è:

```
if (condizione1) {
    // Blocco di codice eseguito se la condizione1 è vera
} else if (condizione2) {
    // Blocco di codice eseguito se la condizione2 è vera (e la
```

```
    condizione1 è falsa)
}
```

else

L'istruzione `else` permette di eseguire un blocco di codice se nessuna delle condizioni precedenti è vera. È sempre usata dopo un'istruzione `if` o `else if`. La sintassi è:

```
if (condizione) {
    // Blocco di codice eseguito se la condizione è vera
} else {
    // Blocco di codice eseguito se nessuna delle condizioni precedenti è
vera
}
```

L'istruzione `else`, se usata, deve essere per forza l'ultima nell'elenco di `if/else if`.

Istruzioni di iterazione

Le istruzioni di iterazione (o cicli) consentono di eseguire ripetutamente un blocco di codice fino a quando una determinata condizione è vera o falsa. Le istruzioni di iterazione più comuni in C sono l'istruzione `for`, l'istruzione `while` e l'istruzione `do-while`.

for

L'istruzione `for` permette di eseguire un blocco di codice per un numero definito di volte. La sintassi è:

```
for (inizializzazione; condizione; aggiornamento) {
    // Blocco di codice eseguito fino a quando la condizione è vera
}
```

while

L'istruzione `while` permette di eseguire un blocco di codice finché una condizione specificata è vera. La sintassi è:

```
while (condizione) {
    // Blocco di codice eseguito finché la condizione è vera
}
```

Istruzioni di salto

Le istruzioni di salto consentono di cambiare il flusso di esecuzione del programma in modo non sequenziale. Le istruzioni di salto più comuni in C sono `break`, `continue`, `return`.

- **break**: L'istruzione `break` viene utilizzata per interrompere l'esecuzione di un ciclo o di uno switch-case. Quando viene incontrata, il controllo viene passato all'istruzione successiva al ciclo o allo switch-

case. È spesso utilizzata per uscire da un ciclo in modo prematuro.

- **continue**: L'istruzione `continue` viene utilizzata all'interno di un ciclo per interrompere l'iterazione corrente e passare all'iterazione successiva del ciclo. In altre parole, salta il resto del corpo del ciclo e passa all'iterazione successiva.
- **return**: L'istruzione `return` viene utilizzata all'interno di una funzione per restituire un valore al chiamante della funzione. Quando viene incontrata, la funzione termina immediatamente e il controllo viene restituito al punto di chiamata della funzione.

Lifetime di una variabile

Per lifetime si intende la vita di una variabile: cioè dove "nasce" e dove "muore":

- nasce quando viene dichiarata
- muore alla chiusura delle parentesi graffe entro la quale è stata dichiara

Ad esempio:

```c
int main(){

   int var1 = 50; //var1 nasce qua

   if(var1>25){
      int var2= 80; //var2 nasce qua
      printf("Var2 vale : %d",var2);
   } //var2 muore qua

   return 0;
} //var 1 muore qua
```

Questo perché **var1** è stata dichiarata dentro le graffe del main e quindi muore alla chiusura delle graffe del main. Mentre **var2** è stata dichiarata nelle graffe dell'*if* e quindi muore alla chiusura delle graffe dell'*if* stesso.

Array

Un array è una tipologia di variabile che può contenere un insieme di elementi dello stesso tipo. Gli array forniscono un metodo conveniente per memorizzare e accedere a più valori utilizzando un singolo nome di variabile.

Definizione e inizializzazione di array

Per definire un array in C, si specifica il tipo di dati degli elementi dell'array seguito da un nome per l'array e una dimensione tra parentesi quadre []. Ad esempio:

```
tipo nome[dimensione]
```

```
int numeri[5];
```

Questa istruzione definisce un array chiamato "numeri" che può contenere 5 elementi di tipo int.

Gli array possono anche essere inizializzati durante la dichiarazione:

```
int numeri[5] = {1, 2, 3, 4, 5};
```

In questo caso, l'array "numeri" viene dichiarato e inizializzato con i valori specificati.

Accesso agli elementi di un array

Gli elementi di un array hanno un indice che parte da `0` a `dimensione-1`, dove la dimensione è il numero totale di elementi nell'array. Per accedere a un elemento specifico dell'array, si utilizza la sua posizione nell'array, chiamata indice.

La sintassi è:

```
nome[posizione]
```

Ad esempio:

```
int primo_numero = numeri[0];  // Accesso al primo elemento
int secondo_numero = numeri[1];  // Accesso al secondo elemento
```

L'elemento dell'array può anche essere modificato utilizzando lo stesso operatore di accesso:

```
numeri[2] = 10;  // Modifica il terzo elemento dell'array
```

Operazioni sugli array

Gli array consentono di eseguire una serie di operazioni utili:

- **Iterazione**: Si può iterare su tutti gli elementi di un array utilizzando cicli come for o while.

```
for (int i = 0; i < 5; i++) {
    printf("%d ", numeri[i]);
}
```

- **Stringhe**: In C, le stringhe sono array di caratteri terminati da un carattere nullo ('\0'). Possono essere manipolate come array di caratteri.

```
char nome[] = "Mario";
printf("Il primo carattere del nome è: %c\n", nome[0]);  // Stampa 'M'
```

Considerazioni sugli array

- **Indici fuori limite**: Accedere a un elemento di un array con un indice al di fuori del suo intervallo può causare errori nel programma.
- **Dimensioni statiche**: Gli array in C hanno dimensioni statiche, non può essere modificata durante l'esecuzione del programma. Le funzioni sono blocchi di codice riutilizzabili che eseguono un'operazione specifica. Nella programmazione strutturata, le funzioni svolgono un ruolo fondamentale nell'organizzazione del codice, consentendo di suddividere un programma complesso in blocchi più piccoli e gestibili.

Funzioni

Definizione e chiamata di funzioni

Una funzione è definita utilizzando la seguente sintassi:

```
tipo_di_ritorno nome_funzione(parametri) {
    // Corpo della funzione
    // Dichiarazioni e istruzioni
    return valore_di_ritorno;
}
```

- **tipo_di_ritorno**: Indica il tipo di dato che la funzione restituirà al termine della sua esecuzione. Può essere un tipo di dato primitivo come int, float, char, o un tipo di dato personalizzato.
- **nome_funzione**: È il nome della funzione che stai definendo. Questo nome sarà utilizzato per chiamare la funzione da altre parti del programma.
- **parametri**: Sono le variabili che la funzione accetta come input. Possono essere zero o più parametri, separati da virgole, o eventualmente essere omessi se la funzione non richiede alcun input.
- **Corpo della funzione**: È il blocco di codice all'interno delle parentesi graffe che contiene le dichiarazioni di variabili e le istruzioni che definiscono il comportamento della funzione.
- **return**: Utilizzato per restituire un valore al termine dell'esecuzione della funzione. Il tipo di valore restituito deve corrispondere al tipo di ritorno dichiarato.

Una volta definita, la funzione può essere chiamata da altre parti del programma. La chiamata di una funzione avviene utilizzando il suo nome seguito da parentesi tonde, eventualmente con gli argomenti richiesti all'interno delle parentesi.

Ad esempio, supponiamo di definire una funzione per calcolare la somma di due numeri interi:

```
int somma(int a, int b) {
    int risultato = a + b;
    return risultato;
}
```

Per chiamare questa funzione e ottenere il risultato, puoi fare qualcosa del genere:

```
int risultato_somma = somma(5, 3);
printf("La somma è: %d\n", risultato_somma);
```

Passaggio di parametri

I parametri possono essere passati a una funzione in due modi principali: per valore e per riferimento.

- **Passaggio per valore**: I parametri sono passati per valore quando il valore effettivo dei parametri viene copiato nei parametri della funzione. Le modifiche apportate ai parametri all'interno della funzione non influiscono sui parametri originali. Questo è il metodo di passaggio di default in C.
- **Passaggio per riferimento**: I parametri sono passati per riferimento quando si passa **l'indirizzo di memoria** dei parametri alla funzione. In questo modo, le modifiche apportate ai parametri all'interno della funzione influiscono direttamente sui parametri originali.