Sistemi Operativi

Unità 3: Programmazione in C

Le funzioni

Martino Trevisan
Università di Trieste
Dipartimento di Ingegneria e Architettura

Argomenti

1. Le funzioni
2. La funzione `main`

Le funzioni

Le funzioni

Una è un insieme di istruzioni che svolge un compito comune.

- Per rendere il codice ordinato
- Permettere il riuso del codice
- Avere codice generico

Il *copia e incolla* è da evitare.

- Disordinato
- Difficile correggere errori

Le funzioni

Una funzione delimita un frammento di codice riutilizzabile.

- Può ricevere dei argomenti in ingresso
- Può fornire un valore di ritorno

Dopo essere definita la funzione viene **invocata**, ovvero utilizzata.

Il `main` è una funzione. Viene invocata dal SO quando viene avviato il programma.

- Riceve degli argomenti (non sempre, vedremo)
- Ritorna un `int`

Le funzioni

Definizione di una funzione:

```
tipoDiRitorno nome (argomenti){
    ...
    istruzioni
    ...
    return valoreDiRitorno; // Opzionale
}
```

Esempio:

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

Le funzioni

Argomenti di una funzione:

Specificano i dati sui quali la funzione deve lavorare

- Rendono la funzione generica
  - Ma una funzione può non ricevere argomenti
- La funzione non deve operare **unicamente** sugli argomenti

Sintassi:

```
tipoDiRitorno nome (tipo nome, tipo nome, ...) {...
```

Esempio:

```
float radice ( float numero ){...}
```

Le funzioni

Argomenti di una funzione:

Se la funzione non riceve argomenti, si indica `void`.

Esempio:

```
int pigreco(void){
    return 3.14;
}
```

Altre funzioni che non richiedono parametri:

- Dimmi l'ora corrente: `int time(void)`

Le funzioni

Argomenti di una funzione:

Se non si indica niente come argomento, la funzione può ricevere un numero arbitario di argomenti.

- Sistema utilizzato per funzioni come `printf` o `scanf`.
- Difficile creare funzioni con numero variabile di argomenti.
  - Non ce ne occuperemo

```
void stampa(){
    printf("ciao!\n");
}

stampa();
stampa(2, 3); // Corretto, parametri ignorati
```

Le funzioni

Valore di ritorno:

Specifica il tipo di dato ritornato dalla funzione come risultato

Se non deve ritornare un risultato, si indica `void`

Esempio:

```c
void stampaCiao(void){
    printf("ciao\n");
}
```

```c
int somma(int a, int b){} // Ritorna un intero
```

Le funzioni

Valore di ritorno:

L'istruzione `return` termina istantaneamente la funzione.

- Specifica il valore di ritorno (se previsto)
- Non è necessaria se la funziona non ha valore di ritorno (ritorna `void`)

```c
int somma(int a, int b){
    return a+b; // Necessario ritornare un intero
}
```

```c
void stampaCiao(void){
    printf("ciao\n");
    return; // Non necessario!
}
```

Le funzioni

Valore di ritorno:

L'istruzione `return` può sempre essere usata per far terminare la funzione prima della fine delle istruzioni.

```c
void stampaCiao(void){
    printf("ciao\n");           // Sempre eseguito
    return;
    printf("Mai Eseguito!\n"); // Non viene eseguito
}
```

```c
int radice(int n){
    if (n<0)          // In caso di errore
        return -1;  // Termina e ritorna -1
    ... calcolo...
    return r;         // Ritorna la radice se tutto ok
}
```

Le funzioni

Importanza della definizione:

La prima riga di una funzione definisce il valore di ritorno e gli argomenti.
Fondamentale per capire *input* e *output* della stessa.

Essa è utilizzata per documentare il codice

- Solo le istruzioni non sono incluse nella definizione
- Non interessa nella documentazione.

Esempio:

`int strlen(const char *s)` *calculate the length of a string*

La funzione `main`

La funzione `main`

La funzione `main` viene eseguita dal SO quando il processo viene avviato.

- Ovvero quando il programma viene messo in esecuzione

Riceve come argomenti i parametri delle linea di comando.

- Ovvero il testo scritto in coda al nome del programma quando lanciato

```
./myprog arg1 arg2 ...
```

Fornisce un `int` come valore di ritorno, detto **exit code**.

- Canale di comunicazione programma-SO per comunicare errori di esecuzione

La funzione `main`

Definizione:

```
int main(int argc, char *argv[]);
```

Argomenti:

- `int argc`: numero di parametri sulla riga di comando.
  - In assenza di parametri vale 1.
  - Incrementato per ogni parametro.
- `char* argv[]`: vettore dei parametri.
  - E' un vettore di puntatori a carattere.
  - Ogni puntatore a carattere del vettore è un argomento in forma di una stringa
  - `argv[0]` è sempre il nome del programma. I parametri effettivi iniziano da `argv[1]`

La funzione `main`

Esempio:

```
./myprog ciao mondo
```

`argc` vale 3

`argv` vale "./myprog", "ciao", "mondo"

```
./myprog
```

`argc` vale 1

`argv` vale "./myprog"

La funzione `main`

Esempio: Stampa di `argc` e `argv`

```c
#include <stdio.h>
int main(int argc, char *argv[]){
    int i ;
    printf("argc = %d\n", argc) ;
    for(i=0; i<argc; i++)
        printf("argv[%d] = \"%s\"\n",i, argv[i]);
    return 0;
}
```

Esecuzione:

```
$./sample
argc = 1
argv[0] = "./sample"
```

```
$./sample ciao mondo
argc = 3
argv[0] = "./sample"
argv[1] = "ciao"
argv[2] = "mondo"
```

La funzione `main`

Osservazioni:

`argv` contiene un vettore di stringhe. Se essi devono essere interpretati come numeri, vanno convertiti tramite funzioni come `atoi`, `atof` o `sscanf`.

Se un programma non ha necessità di ricevere dei parametri, può definire il `main` senza argomanti.

```
int main(){...
int main(void){...
```

La funzione `main`

Valore di ritorno:
il `main` **può** restituire in `int` che viene esaminato dal SO.
Esso indica se c'è stato un errore nell'esecuzione. Per convenzione.

- 0 indica che non c'è stato errore
- Un numero diverso da 0 indica un errore.
  - Il significato del numero, è specifico del programma

Questo sistema viene molto utilizzato:

- In script Bash che necessitano si sapere se i programmi eseguiti hanno avuto errore
- Per i moduli del SO, che devono avviare servizi, demoni e programmi in background.

La funzione `main`

Esempio: si scriva un programma che accetta un solo parametro e lo stampa.

```c
#include <stdio.h>
int main(int argc, char *argv[]){

    /* Con un parametro, argc=2,
    siccome il primo elemento è il nome del programma */
    if (argc!=2){
        printf("Numero di parametri errato\n");
        return 1; /* Il programma termina in questo punto */
    }

    printf("%s\n", argv[1]);
    return 0;
}
```

La funzione `main`

Valore di ritorno:

Per ottenere il valore di ritorno di un programma all'interno di uno script Bash si usa la variabile `$?`.

Contiene il valore di ritorno dell'ultimo programma lanciato

```
./myprog ciao mondo
code=$? //Necessario salvarlo, altrimenti sovrascritto dopo echo
echo "MyProg ha fornito come codice di ritorno $code"
if (( $code == 0 )) ; then
    echo "Nessun errore"
else
    echo "Errore"
fi
```

Nota: il valore di `$?` viene scritto dopo ogni comando, anche dopo `echo`!

La funzione `main`

Osservazione: si può dichiarare la funzione `main` perchè non ritorni nulla. Il programma funziona ma non è corretto. Il SO riceve un valore di ritorno casuale.

```
void main (int argc, char * argv[]){...}
```

Sarebbe da evitare.

La funzione `main`

Esercizio: si scriva un programma che riceve due interi come parametri e ne stampa la somma.

```c
#include <stdio.h>
#include <stdlib.h> /* Necessario per atoi */
int main(int argc, char *argv[]){
    int a, b;

    if (argc!=3){ /* Considerare che argv[0] è il nome del programma */
        printf("Usage: somma a b\n");
        return 1; /* Codice di errore */
    }

    a = atoi(argv[1]); /* Conversione a int*/
    b = atoi(argv[2]);
    printf("%d\n", a+b);

    return 0; /* Nessun errore */
}
```

Nota: cosa succede se viene lanciato come `./somma 3 4` e come `./somma ciao mondo`?