Funzioni e Procedure

Dott.ssa Veronica Rossano
rossano@di.uniba.it
http://www.di.uniba.it/~rossano

Sviluppo top-down

- Costruzione del programma per livelli successivi
  - Corrispondenza con la scomposizione del problema cui è relativo
    - Strumento concettuale per la costruzione di algoritmi
  - Raffinamento successivo delle parti in cui viene scomposto (sottoprogrammi) fino al codice finale
    - Strumento operativo per l'organizzazione e lo sviluppo di programmi complessi

Sottoprogramma

- Insieme di istruzioni
  - Individuate da un nome
  - Che concorrono a risolvere un problema
    - Ben definito
    - Sensato
  - Non necessariamente fine a sé stesso
    - Supporto per la risoluzione di problemi più complessi
- Esempi:
  - Scambio, Ricerca del Minimo, Ordinamento,...

Procedure vs Funzioni

- Procedure
  - Sottoprogrammi il cui compito è quello di produrre un effetto
    - Modifica del valore di variabili
    - Comunicazione di informazioni all'utente
- Funzioni
  - Sottoprogrammi che hanno come risultato il calcolo di un valore
    - All'interno della dichiarazione della funzione vi è un'istruzione che ritorna il risultato

Sottoprogramma in C

- Il C consente la definizione di sottoprogrammi (sia procedure che funzioni) utilizzando sempre lo stesso costrutto
- Durante la costruzione di un programma sarà possibile definire una funzione o una procedura per ogni sottoprogramma definito durante la fase di progettazione

Dichiarare una funzione o una procedura

- La dichiarazione di una funzione/procedura in C:
  - prende il nome di prototipo
  - è obbligatoria
  - deve essere inserita subito dopo le direttive del precompilatore (#include e #define) e prima del main
- La sintassi è la seguente:

```
tipo_risultato nome_sottoprogramma (elenco_argomenti)
```

- `void calcola_max (int a, int b)`
- `int fattoriale (int n)`

Procedure vs Funzioni in C

- Poiché in C esiste un unico costrutto per costruire sia funzioni che procedure si usa la parola chiave VOID per segnalare che un sottoprogramma non restituisce nulla ad un programma chiamante

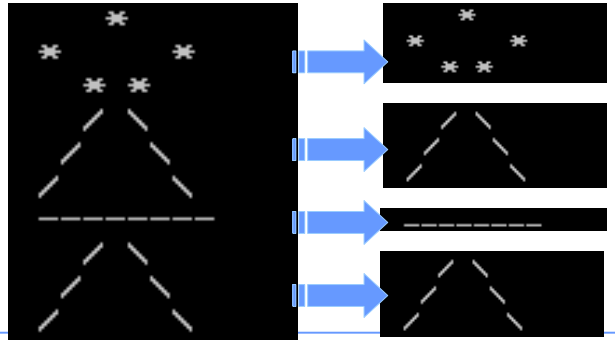
- **`int fattoriale (int n)`**
  - **È una funzione che restituisce il fattoriale di un numero n**
- **`void draw_cerchio (int n)`**
  - **È una procedura che disegna n volte un cerchio**

Schema di un programma C

```
/*Direttive per il preprocessore */
#include <stdio.h>

/*Dichiarazione dei prototipi */
tipo_risultato nome_sottoprogramma (elenco_argomenti);

int main()
{
  nome_sottoprogramma (elenco_argomenti);
  /* Chiamata del sottoprogramma*/

}

tipo_risultato nome_sottoprogramma (elenco_argomenti)
{
  …
}
```

Esempio…

- Supponendo di voler scrivere un programma che disegni la figura seguente sullo schermo



…Esempio

- Scomponiamo il problema in diverse funzioni che siano in grado di:
  - Disegnare un cerchio con degli asterischi
  - Disegnare una intersezione
  - Disegnare una linea di base
- I prototipi delle procedure saranno
  - void cerchio (void)
  - void intersezione (void)
  - void base (void)

N.B. La parola chiave void si utilizza quando non si prevede nessun dato (sia per l'argomento che per il risultato)

Definizione di una funzione/procedura

- La definizione di una funzione/procedura ricalca la definizione del main
- La sintassi:

```
tiporisultato
nomesottoprogramma (elenco argomenti)
  {
     /* dichiarazioni locali */
     /* istruzioni del sottoprogramma */
  }
```

NB. Alcune volte il tipo di dato restituito è posto sulla riga che precede il nome del sottoprogramma. La sintassi è comunque corretta

Esempio

- La procedura che disegna un cerchio di asterischi sarà definita come segue:

```
void
cerchio (void)
  {
     printf ( "        *        \n ");
     printf ( "   *         *          \n ");
     printf ( "      *   *      \n ");
  }
```

Le dichiarazioni locali

- Gli eventuali identificatori dichiarati all'interno del sottoprogramma sono nomi locali e possono essere usati (sono visibili) solo ed esclusivamente all'interno di esso
- Le istruzioni eseguibili descrivono le operazioni che effettua il sottoprogramma
- L'ordine delle definizioni non influisce sull'ordine delle esecuzioni che è determinato dall'ordine delle istruzioni di chiamata dei sottoprogrammi nel programma chiamante

Prototipi delle procedure

Procedure.c

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

void cerchio (void);
void intersezione (void);
void base (void);
void triangolo (void);

int
main (void)
{
      cerchio();
      triangolo();
      intersezione ();
      system ("pause");
      return (0);
}
```

```
void
cerchio (void)
{
        printf("      *   \n");
        printf("  *      * \n");
        printf("    * *  \n");
}

void
intersezione (void)
{
     printf("    / \\   \n");
     printf("   /   \\  \n");
     printf("  /     \\  \n");
}

void
base (void)
{
     printf("  --------\n");
}

void triangolo (void)
{
     intersezione();
     base();
}
```

Chiamate delle procedure

Dichiarazione delle procedure

Funzioni e Procedure con parametri (argomenti)

- I parametri (argomenti) sono utilizzati per consentire l'interazione tra i vari sottoprogrammi e il main
  - In altre parole per portare informazioni dal main ai sottoprogrammi e viceversa
- I parametri consentono di costruire sottoprogrammi più versatili che possono essere utilizzati in contesti completamente differenti
  - Argomenti di input usati per passare informazioni ad un sottoprogramma
  - Argomenti di output usati per passare informazioni al programma chiamante

Parametri di input

- I parametri devono essere dichiarati dopo il nome del sottoprogramma e devono essere racchiusi tra parentesi
- Per ogni parametro è necessario dichiarare il tipo

```
void
calcola_area (double base, double altezza)
  {
      double area;
      area=base*altezza;
      printf("l'area e' pari a %.2f metri", area);
  }
```

Chiamata di una procedura

- Per chiamare una procedura è sufficiente inserire nel main (o in una qualunque altro sottoprogramma) il nome della procedura seguito dai parametri reali (o attuali) da sostituire a quelli formali
- La corrispondenza è determinata dalla posizione

```
void calcola_area (double base, double altezza);

main(void)
{
   double b, h;
    ….
    calcola_area (b, h);
   ….
}
```


Chiamata di una funzione

- La chiamata di una funzione avviene sempre a destra di una istruzione di assegnazione

```
double calcola_area (double base, double altezza);

int main(void)
{
 double b, h, area_rett;
 …
 area_rett=calcola_area (b, h);
 printf("L'area del rettangolo è: %.2f", area_rett);
 …
 return (0);
}
```


Restituzione del risultato in una funzione

- Nella dichiarazione di una funzione è necessario indicare il tipo di risultato che la funzione deve restituire
- L'istruzione Return consente di restituire il valore computato dalla funzione al programma chiamante

```
double
calcola_area (double base, double altezza)
  {
      double area;
      area=base*altezza;
      return (area);
  }
```


Regola fondamentale

- Quando si usano funzioni/procedure a più argomenti è necessario assicurarsi che siano verificate le corrispondenze tra parametri formali e parametri attuali relativamente a:
  - Numero di argomenti
  - Ordine degli argomenti
  - Tipo di ciascun argomento

Esercizio

- Riscrivere il programma per calcolare il massimo tra tre numeri usando una funzione che calcola il massimo tra due numeri.

Massimo tra tre numeri senza funzioni

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

int main ()
{
 int a, b, c, max;
 printf("Inserire il primo valore ----> ");
 scanf("%d", &a);
 printf("Inserire il secondo valore ----> ");
 scanf("%d", &b);
 printf("Inserire il terzo valore ----> ");
 scanf("%d", &c);
 if (a>b)
    max=a;
 else
     max=b;
 if (c>max)
       max=c;
 printf("Il massimo e' %d \n", max);
 system("pause");
 return 0;
}
```


Massimo tra tre numeri con funzioni

Max3Funzioni.c

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

int maxcoppia(int primo, int secondo);

int main ()
{
 int a, b, c, massimo;
 printf("Inserire il primo valore ----> ");
 scanf("%d", &a);
 printf("Inserire il secondo valore ----> ");
 scanf("%d", &b);
 printf("Inserire il terzo valore ----> ");
 scanf("%d", &c);
 massimo=maxcoppia(a,b);
 massimo=maxcoppia(massimo,c);
 printf("Il massimo e' %d \n", massimo);
 system("pause");
 return 0;
}

int maxcoppia(int primo, int secondo)
{
    int max;
    if (primo>secondo)
        max=primo;
    else
        max=secondo;
    return (max);
}
```


Esempio

- Scrivere un algoritmo e il relativo programma che calcoli la temperatura ad una certa profondità in gradi Celsius e Fahrenheit chiedendo in input la profondità (in km) e utilizzando le formule seguenti:
  - Celsius = 10 * profondita + 20
  - Fahrenheit = 1.8 * Celsius + 32
- Utilizzare due funzioni

CelsiusFahrenheit.c

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

double celsius(double km);
double fahrenheit(double temp_cels);

int
main (void)
{
 double profondita,temperatura_c, temperatura_f;
 printf("Inserisci la profondita' espressa in km --> ");
 scanf("%lf", &profondita);
 temperatura_c=celsius(profondita);
 printf("\n\nLa temperatura alla profondità di %.2f km in gradi Celsius e' %.3f",
                                        profondita, temperatura_c);
 temperatura_f=fahrenheit(temperatura_c);
 printf(" mentre in gradi Fahrenheit e' %.3f\n\n", temperatura_f);
 system("pause");
 return(0);
}

double celsius(double km)
{
        return(10*km+20);
}
double fahrenheit(double temp_cels)
{
        return(1.8*temp_cels+32);
}
```

I valori restituiti dalla funzione possono anche essere calcolati direttamente nella return senza usare variabili locali

Generazione di numeri casuali

- Per poter generare una serie di numeri casuali si usa la funzione rand()
  - È inclusa nella libreria standard <stdlib.h>
  - Restituisce un intero compreso tra 0 e RAND_MAX che è una costante definita nella libreira standard
  - Solitamente il valore di RAND_MAX è 32767 il massimo valore rappresentabile per un intero di due byte

Esercizio

Monetine.c

- Realizzare l'algoritmo e il programma per simulare il lancio di una monetina. Consentire all'utente di scegliere quante volte lanciare la monetina e per ogni lancio il programma dovrà visualizzare Testa o Croce e contare il numero di occorrenze per la comparsa di ogni faccia della monetina. Usare una funzione distinta che non riceverà argomenti e che restituirà il risultato del lancio della monetina (croce o testa)

Esercizio

TavolaPitagorica.c

- Scrivere un programma che aiuti uno studente di scuola elementare ad allenarsi nel ricordare la tavola pitagorica. Utilizzare la funzione rand() per produrre due interi positivi di una cifra. Il programma dovrà chiedere "Quanto fa 4 per 5?". In seguito lo studente dovrà digitare la risposta. Nel caso lo studente risponda correttamente visualizzare un feedback positivo (es. BRAVO!!!) in caso contrario un feedback di incoraggiamento con la risposta corretta.
- Consentire all'utente di riprovare più volte fino a quando non vorrà uscire dal programma

La randomizzazione…

- Si noterà che la funzione rand produce sempre la stessa serie di numeri
- Per rendere realmente casuale la produzione dei numeri sarà necessario "inizializzare" la funzione rand() utilizzando la funzione srand()
  - Prende in input un valore unsigned (intero senza segno)

…La randomizzazione

- Per usare la funzione srand() sarà necessario aggiungere le seguenti righe di codice all'inizio del programma

```
unsigned seme;
scanf("%u", &seme);
srand(seme);
```

- È possibile inizializzare la funzione rand usando anche la funzione time() che restituisce l'ora corrente del pc in questo caso le righe da aggiungere saranno

```
#include <time.h>
. . .
srand(time(NULL));
```

Il parametro di input NULL consente alla funzione time() di restituire un intero senza segno invece della stringa con l'indicazione dell'ora

Parametri di un sottoprogramma

- L'elenco degli argomenti fornisce i legami di comunicazione tra il programma chiamante e il sottoprogramma chiamato
- I parametri permettono di mandare in esecuzione il sottoprogramma, ogni volta che viene chiamato, con valori differenti.

Parametri di output

- L'istruzione return consente alla funzione di restituire un risultato al programma chiamante
- È possibile definire parametri di output anche per le procedure che consentano di restituire anche più di un risultato al programma chiamante
- È necessario che il programma crei delle celle di memoria in cui conservare i parametri formali per renderli disponibili successivamente al programma chiamante

Passaggio di parametri per indirizzo…

- Quando una procedura deve restituire un risultato è necessario che si usi il passaggio di parametri per referenza (o indirizzo)
- L'elenco dei parametri formali deve essere presente sia nel prototipo che nella definizione della procedura
- Anche nelle funzioni è possibile usare il passaggio di parametri per indirizzo
- La sintassi per la dichiarazione di un parametro passato per indirizzo è:

```
Tipo risultato
Nome_funzione(tipo_par_in  nome_par_in, tipo_par_out  *nome_par_out)
```


…Passaggio di parametri per indirizzo

- Nell'istruzione di dichiarazione l'operatore * dichiara che il parametro formale indicato è un puntatore ad una variabile del tipo indicato prima del suo nome
- La dichiarazione di un puntatore comunica al compilatore che la variabile dichiarata conterrà l'indirizzo della variabile del tipo indicato

I puntatori in C…

- La dichiarazione del tipo:

```
double *temperatura;
```

  - Indica che la variabile nome sarà un puntatore ad una variabile di tipo double, in altre parole conterrà l'indirizzo della cella di memoria in cui è conservata una variabile di tipo double

…I puntatori in C…

- Nelle computazioni se ci si vuole riferire:
  - all'indirizzo della cella di memoria che contiene la temperatura dovrà essere utilizzato l'operatore di indirizzo &

```
indirizzo=&temperatura;
```

  - al valore della temperatura dovrà essere utilizzato l'operatore *

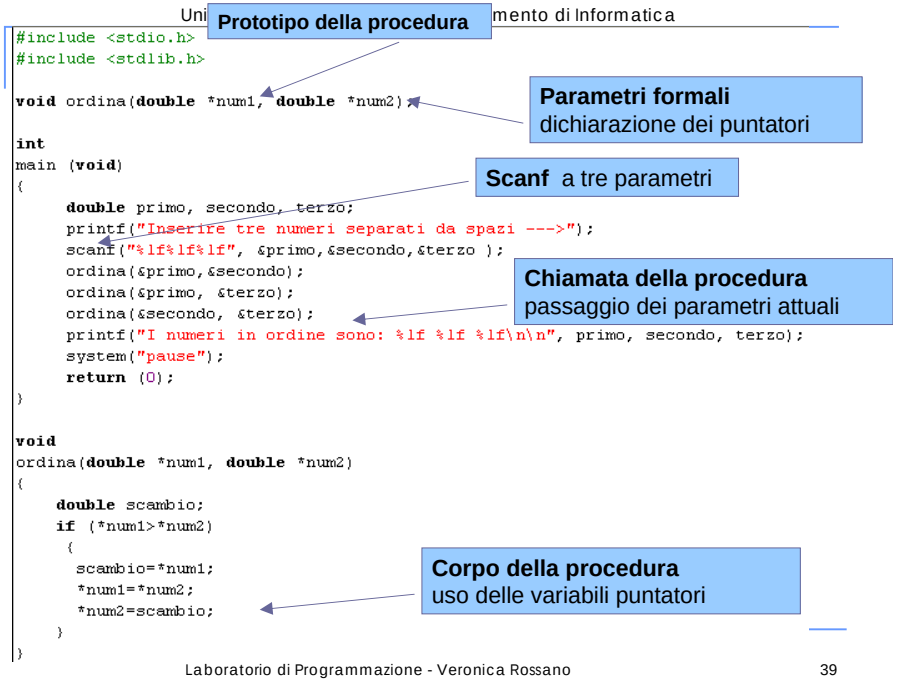
```
temp_corporea=*temperatura;
```

Puntatori e sottoprogrammi in C

- Nell'istruzione di chiamata di un sottoprogramma i parametri attuali corrispondenti ai parametri formali saranno preceduti dall'operatore di indirizzo &

Esempio

- Scrivere un programma che prenda tre numeri in input e li restituisca in ordine




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

void ordina(double *num1, double *num2);

int
main (void)
{
     double primo, secondo, terzo;
     printf("Inserire tre numeri separati da spazi --->");
     scanf("%lf%lf%lf", &primo,&secondo,&terzo );
     ordina(&primo,&secondo);
     ordina(&primo, &terzo);
     ordina(&secondo, &terzo);
     printf("I numeri in ordine sono: %lf %lf %lf\n\n", primo, secondo, terzo);
     system("pause");
     return (0);
}

void
ordina(double *num1, double *num2)
{
    double scambio;
    if (*num1>*num2)
     {
      scambio=*num1;
      *num1=*num2;
      *num2=scambio;
    }
}
```


La chiamata di un sottoprogramma con passaggio per indirizzo

- L'istruzione di chiamata al sottoprogramma ordina(&primo, &secondo) indica come i parametri attuali siano istanziati con gli indirizzi delle due celle di memoria che conterranno i parametri attuali

Significato dell'operatore *...

- Operatore aritmetico
  - Nelle istruzioni di un programma è utilizzato per calcolare la moltiplicazione tra numeri
- Definizione di puntatori
  - Nelle dichiarazioni è utilizzato per la definizione di una variabile puntatore
- Recupera il dato puntato da
  - Nelle istruzioni di un programma è utilizzato per recuperare il dato puntato da una variabile puntatore

...Significato dell'operatore *

```
int quantita, prezzo, totale;
totale=quantita*prezzo;
```

```
int quantita, prezzo;
int *totale;
…
calcola_prezzo(quantita, prezzo, &totale)
```

```
int *totale;
…
printf("La spesa totale e' %d", *totale);
```


L'istruzione scanf...

- L'istruzione rappresenta una chiamata ad una funzione che utilizza come parametri formali dei puntatori a delle celle di memoria che restituiscono l'indirizzo della cella in cui l'input è memorizzato
- È possibile introdurre più input in un'unica istruzione

```
scanf("%lf%lf%lf", &primo,&secondo,&terzo );
```


...L'istruzione scanf

- La funzione scanf restituisce un numero che consente di capire se l'istruzione è andata a buon fine. Il numero restituito rappresenta il numero di parametri inseriti dall'utente in input.

```
status=scanf("%lf%lf%lf", &primo,&secondo,&terzo );
restituisce 3 in status
```

```c
void ordina(double *num1, double *num2);


int
main (void)
{
     double primo, secondo, terzo;
     int val_input;
     char continua;
     do
     {
     printf("Inserire tre numeri separati da spazi (es. 23.7 1.32 67.5)--->");
     val_input=scanf("%lf%lf%lf", &primo,&secondo,&terzo );
     if (val_input!=3)
      {
       printf("""\nSi e' verificato un errore nell'input. Riprocedere\n\n");
       scanf("%c", &continua);
      }
     } while (val_input!=3);
     ordina(&primo,&secondo);
     ordina(&primo, &terzo);
     ordina(&secondo, &terzo);
     printf("I numeri in ordine sono: %lf %lf %lf\n\n", primo, secondo, terzo);
     system("pause");
     return (0);
}
```