# SOTTOPROGRAMMI [1]

- SOPPROGRAMMA
  - Parte di codice logicamente separata dal resto del programma che esegue un insieme di operazioni sui dati;
  - Racchiudo porzioni di codice riutilizzabili più volte all'interno dello stesso programma o, importandole, in altri programmi (librerie);
  - Oltre a costruirmi tipi di dato posso costruirmi delle operazioni sugli stessi;
  - Rendo parti di codice parametriche rispetto ai dati, sempre mantenendo un controllo di tipo.
- IMPLEMENTAZIONI LOGICHE
  - Funzioni;
  - Procedure.

# SOTTOPROGRAMMI [2]

- ALCUNI ESEMPI
  - sin(num);
  - scanf("…",a);
  - strcmp(s1,s2).
- DEFINIZIONE DI UN SOTTOPROGRAMMA
  - Definire un identificatore univoco (Prototipo – Header)
  - Definire tipi di ingressi e uscite (Prototipo – Header)
  - Definisco le operazioni svolte (Implementazione)
- ELEMENTI COSTITUTIVI
  - Header (può essere separato dall'implementazione);
  - Parte dichiarativa locale (nell'implementazione);
  - Parte esecutiva (nell'implementazione).

# SOTTOPROGRAMMI [3]

- HEADER
  - Definisco tipo del risultato, identificatore del sottoprogramma, lista dei **parametri formali**.
  - Es:

```
int MiaSomma(int x, int y);
int RadInt(int par);
Stipendio CalcolaStipendio(Dipendente Dip);
```

  - Il tipo dei parametri e del risultato può essere uno qualsiasi tra quelli *built-in* o *user-defined* (+ **void**)
- PARTE DICHIARATIVA LOCALE
  - Posso dichiarare alcune variabili che sono locali al sottoprogramma e come tali esistono e sono conosciute solo al suo interno

# SOTTOPROGRAMMI [4]

- PARTE ESECUTIVA
  - Contiene la reale implementazione del prototipo del sottoprogramma, con le operazioni da eseguire
  - Può essere contenuta in altri file a patto che il prototipo si trovi o sia incluso nel programma principale
- ES: RADICE INTERA [1]

```
#include <stdio.h>
int RadInt(int numero);
main()
{
     int Num;
     printf("\nIns Num:");
     scanf("%d",&Num);
     printf("\nLa radice intera di %d è: %d",Num, RadInt(Num));
}
```

# SOTTOPROGRAMMI [5]

- ES: RADICE INTERA COMPLETO [2]

```
#include <stdio.h>
int RadInt(int numero)
{
     int Cont=0;
     while(!(Cont^2<=numero && (Cont+1)^2>numero))
     {
         Cont++;
     }
     return Cont;
}

main()
{
     int Num;
     printf("\nInserisci Numero:");
     scanf("%d"&Num);
     printf("\nLa radice intera di %d è: %d",Num, RadInt(Num));
}
```

# SOTTOPROGRAMMI [6]

- CHIAMATA DI UN SOTTOPROGRAMMA
  - La chiamata di un sottoprogramma copia i parametri **attuali** all'interno dei parametri **formali,**
  - L'istruzione **return** assegna alla variabile risultato un valore e provoca la terminazione del sottoprogramma;
  - Il risultato è identificato col nome del sottoprogramma
  - La chiamata può essere effettuata all'interno di qualsiasi espressione purché sia visibile da essa i parametri attuali possono essere espressioni qualsiasi purché sia rispettata la compatibilità di tipo;
  - La corrispondenza tra parametri formali e parametri attuali segue l'ordine della dichiarazione;

# SOTTOPROGRAMMI [7]

- PASSAGGIO PARAMETRI
  - Per Valore (l'unico possibile in C)
  - Per Indirizzo (simulabile tramite puntatori)
- CHIAMATA
  - Passaggio parametri
  - Passaggio del controllo
  - Ritorno al chiamante
- AMBIENTE LOCALE DI UN SOTTOPROGRAMMA
  - Variabili dichiarate nella parte dichiarativa globale
  - Tutte le variabili dichiarate al suo interno
  - Parametri formali

# SOTTOPROGRAMMI [8]

- ES: CALCOLO VOLUME

```
#include <stdio.h>
typedef struct
        {
            int lato1;
            int lato2;
            int lato3;
        } Parallelepipedo;
int CalcolaVolume(Parallelepipedo P)
    {
            int Volume;        // Superfluo
            Volume=P.lato1*P.lato2*P.lato3;
            Return Volume;
    }
Paralellepipedo Par;
main()
{
    […]
}
```

# VISIBILITÀ [1]

- BLOCCO
  - Parte di codice racchiuso tra parentesi graffe
  - Un blocco può avere una sua parte dichiarativa ed una sua parte esecutiva
  - Es:

```
main ()
{
  int a,b;
   […]
   {
    char a,c;
    […]
      {
         float a;
         […]
      }/* fine blocco 2*/
  }/*fine blocco 1*/
}/* fine main */
```

# VISIBILITÀ [2]

- AMBIENTE GLOBALE
  - Insieme di tutti gli elementi dichiarati nella parte dichiarativa globale
- AMBIENTE LOCALE DI UN SOTTOPROGRAMMA
  - Insieme di tutti gli elementi dichiarati nella parte dichiarativa del sottoprogramma e nella sua testata
- AMBIENTE LOCALE DI UN BLOCCO
  - Insieme di tutti gli elementi dichiarati nella sua parte dichiarativa

# VISIBILITÀ [3]

- REGOLE DI VISIBILITÀ (SCOPE)
  - Gli elementi dell'ambiente globale possono essere utilizzati da qualsiasi funzione o blocco
  - Gli elementi che costituiscono l'ambiente di un blocco possono essere visti da tutti i blocchi annidati
  - Se in un blocco esistono più definizioni dello stesso identificatore vale la più vicina
  - Una variabile dichiarata in un sottoprogramma o nella sua testata è visibile solo al suo interno
  - Una variabile dichiarata fuori da un sottoprogramma è visibile all'intero programma
  - Ogni funzione è visibile in qualunque punto del programma dopo la sua dichiarazione

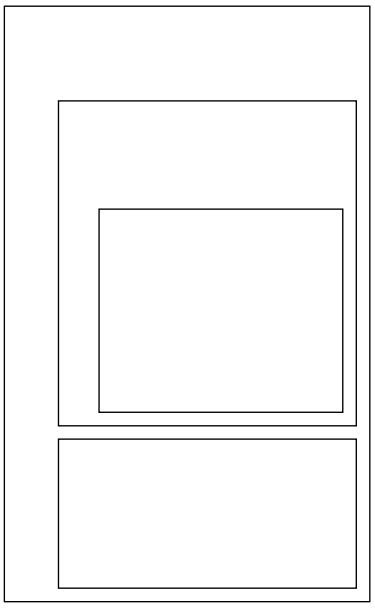
# VISIBILITÀ [4]

- ESEMPIO:

```
#include <stdio.h >
#include <math.h >
int   g1;

main()
{
   int a,b;
   a=2;
   b=a+5;
   g1 = a* b;
   printf("\n %d", b);
   {
      double a;
      a = 3.14159;
      printf("\n %lf", sqrt( a));
   }
   funz1(b);
}

void funz1(int i)
{
   int a[10];
   a[0] = (5 - g1) * i ;
   printf("\n %d", a[0]);
}
```



# DURATA [1]

- VARIABILI STATICHE
  - Create all'inizio dell'esecuzione del programma e distrutte al termine di esso;
- VARIABILI AUTOMATICHE
  - Create ogni volta che si entra nel loro ambito di visibilità e distrutte all'uscita;
- DURATA DELLE VARIABILI
  - Le variabili globali sono statiche;
  - Le variabili dichiarate in una funzione sono automatiche a meno che non vengano dichiarate **static;**
  - Se una funzione viene chiamata due volte, vengono creati due ambienti diversi aventi gli stessi identificatori;

# FUNZIONI E PROCEDURE

- FUNZIONI
  - Operano solamente sull'ambiente locale ricevendo i dati dall'esterno attraverso i parametri formali;
  - Dal punto di vista logico, dovrebbero essere autocontenute e non utilizzano variabili globali;
  - In puro stile matematico hanno un dominio ed un codominio predefiniti;
- PROCEDURE
  - Operano sull'ambiente locale e su quello globale, permettendo di modificare il contenuto di variabili globali;
  - Hanno un dominio e saltuariamente un codominio;