Funzioni e Procedure

Alcuni Esempi:

```
#include <stdio.h>
void  h (int  X, int *Y);

main()
{int  A,B;
  A=0;
  B=0;
  h(A, &B); /*B e‘ un parametro
                  di uscita*/
  printf("\n %d \t %d", A, B);
}

void h (int  X, int *Y)
{
  X=X+1;
  *Y=*Y+1;
  printf("\n %d \t %d", X, *Y);
}

  1   1          (stampa di “h”)
  0   1          (stampa di “main”)
```

Esercizio:

Calcolo delle radici di una equazione di secondo grado.

$$Ax^2 + Bx + C = 0$$

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

int radici(int A, int B, int C,
           float *X1, float *X2);

main()
{ int    A,B,C;
  float  X,Y;
  scanf("%d%d%d%\n",&A,&B,&C);
  if ( radici(A,B,C,&X,&Y) )
             printf("%f%f\n",X,Y);
}
```

```
int  radici(int A, int B, int C,
            float *X1, float *X2)
{ float D;
  D= B*B-4*A*C;
  if (D<0) return 0;
  else
   {  D=sqrt(D);
      *X1 = (-B+D)/(2*A);
      *X2=  (-B-D)/(2*A);
          return 1;
       }
}
```

☞ Anche se A, B, C, X1 e X2 fossero stati tutti di tipo float, si doveva comunque definirli separatamente (diversa tecnica di legame): X1, X2, radici trasmettono valori in uscita.

✐ **Esercizio:**

Programma che stampa i numeri primi compresi tra 1 ed n (n dato).

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

#define NO 0
#define YES  1

int isPrime(int n); /*dichiarazioni*/
int primes(int n);

void main()
{
int n;

  do {
       printf("\nNumeri primi non
              superiori a:\t");
       scanf("%d",&n);
     } while (n<1);

  printf("\nTrovati %d numeri primi.\n",
            primes(n));
}
```

```
int isPrime(int n)
{
int max,i;

if (n>0 && n<4) return YES;
     /* 1, 2 e 3 sono primi     */
else if (!(n%2)) return NO;
/* escludi i pari > 2   */
max = sqrt( (double)n );
/* CAST:sqrt ha arg double */
for(i=3; i<=max; i+=2)
  if (!(n%i)) return NO;
return YES;
}

int primes(int n)
{
  int i,count;

  if (n<=0) return -1;
  else count=0;

  if (n>=1)
      { printf("%5d\t",1);
        count++;
      }
  if (n>=2)
      { printf("%5d\t",2);
        count++; }
  for(i=3;i<=n;i+=2)
    if (isPrime(i))
        { printf("%5d\t",i);
          count++;
        }
  printf("\n");
  return count;
}
```

✍ Esercizio:

Scrivere una procedura che risolve un sistema lineare di due equazioni in due incognite

$a_1x + b_1y = c_1$
$a_2x + b_2y = c_2$

$x = (c_1b_2-c_2b_1) / (a_1b_2- a_2b_1) = XN / D$
$y = (a_1c_2- a_2c_1) / (a_1b_2- a_2b_1) = YN / D$

Soluzione:

```
#include <stdio.h>
void sistema(int A1, int B1, int C1,
             int A2, int B2, int C2,
             float *X, float *Y);

main()
{
  int    A1,B1,C1,A2,B2,C2;
  float  X,Y;

  scanf("%d%d%d%\n",&A1,&B1,&C1);
  scanf("%d%d%d%\n",&A2,&B2,&C2);

  sistema(A1,B1,C1,A2,B2,C2,&X,&Y);
  printf("%f%f\n",X,Y);
}
```

```
void sistema (int A1, int B1, int C1,
          int A2, int B2, int C2,
          float *X, float *Y)
{
  int    XN,YN,D;
  XN = (C1*B2 - C2*B1);
  D =  (A1*B2 - A2*B1);
  YN = (A1*C2 - A2*C1);
  if (D == 0)
    {if (XN ==  0)
       printf("sistema indeterminato\n");
     else
       printf("sistema impossibile\n");
    }
  else
     { printf("Determinante%d",D);
       *X=(float) (XN) /D;
       *Y=(float) (YN) /D;
     }
}
```

Vettori come parametri di funzioni

Il C prevede che parametri di tipo vettore debbano essere passati **attraverso il loro indirizzo**:

Ad esempio:

```
#include <stdio.h>
#define MAXDIM 30

int getvet(int v[], int maxdim);

main()
{
  int k, vet[MAXDIM];
  int dim;
  dim=getvet(vet, MAXDIM);
...
```

Definizione della funzione:

```
int getvet(int *v, int maxdim);
{ int i;

  for(i=0;i<maxdim;i++)
    {printf("%d elemento:\t", i+1);
     scanf("%d", &v[i]); }
  return n;
}
```

☞ Il vettore e' modificato all'interno della funzione e le modifiche sopravvivono all'esecuzione della funzione poiché in C i vettori sono trasferiti ***per indirizzo***.

Esempio: ordinamento di un vettore

leggere, ordinare e stampare un vettore (vedi pagina 3):

```
#include <stdio.h>
#define dim 10

/* dichiarazioni delle funzioni */
void leggi(int V[], int n);

main()
{int V[dim], i,j, max, tmp, quanti;
  leggi(V,dim);
  ordina(V,dim);
  stampa(V,dim);
}

/* definizioni delle funzioni */
void leggi(int V[], int n);
{
  for (i=0; i<n; i++)
    { printf("valore n. %d: ",i);
      scanf("%d", &V[i]);
     }
}

void ordina(int V[], int n);
{
  for(i=0; i<n; i++)
    {quanti=dim-i;
    max=quanti-1;
    for( j=0; j<quanti; j++)
       { if (V[j]>V[max])
            max=j;
       }
    if (max<quanti-1)
       { /*scambio */
```

```
            tmp=V[quanti-1];
            V[quanti-1]=V[max];
            V[max]=tmp;
            }
        }
    }
```

```
void stampa(int V[],int n);
{
  for(i=0; i<n; i++)
    printf("Valore di V[%d]=%d\n", i,
  V[i]);
}
```

Struttura di un Programma C

Se un programma fa uso di funzioni/procedure, la sua struttura viene estesa come segue:

```
#include <stdio.h>
# ...
/* variabili e tipi globali al programma:
   visibilità nell'intero programma */
tipovar   nomevar,  ...;

/* dichiarazioni  funzioni */
int F1 ( parametri );
...
int FN ( parametri );

/*main*/
main (void)
{   /* variabili  locali al main:
  visibilità nel solo main */

/*codice del main:  si invocano le Fi */
} /* fine main */

/* definizioni funzioni */
int F1 ( ... )
{  /* parte dichiarativa */
/*codice di F1*/... }
```

☞ Le definizioni di funzioni non possono essere innestate in altre funzioni o blocchi.

Variabili locali:

Nella parte dichiarativa di un sottoprogramma (procedura o funzione) possono essere dichiarati costanti, tipi, variabili (detti ***locali*** o ***automatiche***).

```
#include <stdio.h>
char  saltabianchi (void);
main(void)
{char C;
 C = saltabianchi();
 printf("\n%c",C);        /* stampa
*/
}
 char  saltabianchi (void)
 {char Car; /* Car e' locale a
saltabianchi */
   do
      {scanf("%c", &Car);}
   while (Car==' ');
   return Car;
 }
```

- ☞ Alla variabile Car si puo' far riferimento solo nel corpo della funzione saltabianchi (***campo di azione***).
- ☞ Il ***tempo di vita*** di Car e' il tempo di esecuzione della funzione saltabianchi.
- ☞ I parametri formali vengono trattati come variabili locali.

Variabili Locali

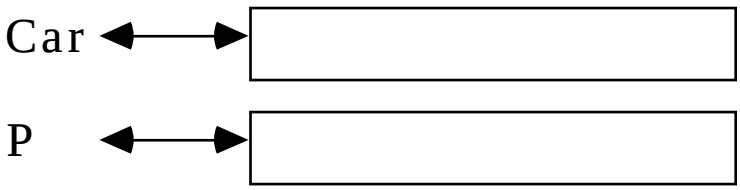
- Quando una funzione viene chiamata, viene creata una associazione tra l'identificatore di ogni variabile locale (e parametro formale) ed una cella di memoria allocata automaticamente.

Esempio:

```
int f(char Car)
{
int P;

main()
{ char C;

f(C);
}
```



- Alla fine dell'attivazione tale associazione viene distrutta. Se la procedura viene attivata di nuovo, viene creata una nuova associazione.

- Non c'e` correlazione tra i valori che Car assume durante le varie attivazioni della funzione f.

Variabili esterne (o globali)

- Nell'ambito del blocco di un sottoprogramma (del main) si puo` far riferimento anche ad identificatori **globali**, nella ***parte dichiarazioni globali*** del programma.

- Il ***tempo di vita*** delle variabili esterne e` pari al tempo di esecuzione del programma (***variabili statiche***).

Esempio:

```
#include <stdio.h>
void  saltabianchi (void);

char C; /* def. variabile esterna */

main()
{
  saltabianchi();
  printf("\n%c",C);      /* stampa C */
}

void  saltabianchi (void)
  {
   do
      {scanf("%c", &C);}
   while (C==' ');
  }
```

Variabili Esterne

Nell'esempio:

- C e` una ***variabile esterna*** sia al main che a saltabianchi.

- la definizione di C e` visibile sia dalla funzione main che dalla procedura saltabianchi. Entrambe queste unita` possono far riferimento alla variabile C.

- E` la ***stessa*** variabile. Ogni modifica a C prodotta dalla funzione, viene "vista" anche dal main.

› **possibilita' di effetti collaterali**

Variabili esterne & passaggio dei parametri

Le variabili esterne rappresentano un modo alternativo ai parametri per far ***interagire*** tra loro le varie unita‘ di programma.

Vantaggi:

- Evitano lunghe liste di parametri (da copiare se passati per valore).

- Permettono di restituire in modo diretto i risultati al chiamante.

Svantaggi:

- I programmi risultano meno leggibili (rischio di errori).

- Interazione meno esplicita tra le diverse unita` di programma (effetti collaterali).

- Generalita`, riusabilita` e portabilita` diminuiscono.

Visibilita‘ degli Identificatori

Identificatori: nomi per indicare costanti, variabili, tipi, funzioni definiti dall’utente.

Dato un programma P, costituito da diverse unita` di programma, eventualmente scomposte in blocchi, si distingue tra:

- ***Ambiente globale***
  e` costituito dalle dichiarazioni e definizioni che compaiono nella parte di dichiarazioni globali di P.

- ***Ambiente locale a una funzione:***
  e` l'insieme delle dichiarazioni e definizioni che compaiono nella parte dichiarazioni della funzione, piu` i suoi parametri formali.

- ***Ambiente di un blocco***
  e` l'insieme delle dichiarazioni e definizioni che compaiono all'interno del blocco.

Effetti collaterali

Si chiama effetto collaterale (***side effect***) provocato dall'attivazione di una funzione la modifica di una qualunque tra le variabili **esterne** (caso di *parametri passati per indirizzo* o assegnamento a *variabili esterne*).

☞ Se presenti, le funzioni non sono piu` funzioni in senso matematico.

Esempio:

```
#include <stdio.h>
int B;
int f (int * A);

main()
{
 B=1;
 printf("%d\n",2*f(&B));  /* (1) */
 B=1;
 printf("%d\n",f(&B)+f(&B)); /* (2) */
}

int f (int * A)
{
 *A=2*(*A);
 return *A;
}
```

☞ Fornisce valori diversi, pur essendo attivata con lo stesso parametro attuale. L'istruzione (1) stampa 4 mentre l'istruzione (2) stampa 6.

Effetti Collaterali

Se nel corpo di una funzione si modifica una variabile non locale o un parametro passato per indirizzo:

```
int V;

float  f (int X)
{
 V=V*X; /* origine side effect */
 return (sqrt(X)+1)
}
```

non e' piu' detto, ad esempio, che:

V+f(X) == f(X) +V

Per evitare effetti collaterali in funzioni occorre:

- non avere parametri passati per indirizzo nelle intestazioni di funzioni;
- non introdurre assegnamenti a variabili esterne nel corpo di funzioni.

Funzioni esterne e file header

In C, ogni funzione o gruppo di funzioni riunito in un file (estensione ***.c***) e' compilabile separatamente ➛ il codice di un programma puo' essere distribuito su piu' files.

- Se in un file si utilizzano funzioni definite in un altro file, occorre ***includere i prototipi*** di tali funzioni.

- I prototipi delle funzioni definite in un file di nome ***file.c*** sono riportate in ***file header*** (di nome ***file.h***).

L'inclusione avviene con la direttiva:

#include ...

seguita dal nome del file header:

- racchiuso tra **< >** se si tratta di un ***header*** collocato in un direttorio standard (ad es. \tc\include).
- racchiuso tra **" "** se si tratta di un ***file header*** collocato in un direttorio qualsiasi (e` bene specificarne il nome assoluto).

Esempi:

```
#include   <stdio.h>

#include   "c:\paolo\utility.h"
```

☞ Se un programma e‘ organizzato su piu‘ file, la compilazione deve avvenire con un comando di ***Make*** che ricerca le dipendenze tra i file, li compila (se necessario), li collega e produce l’eseguibile (vedi *progetti* turbo C)

Funzioni predefinite: La *Standard Library* del C

Tutte le versioni ANSI C mettono a disposizione del programmatore un insieme di ***funzioni predefinite*** che costituiscono la ***Standard Library*** del C.

Tali funzioni sono racchiuse in vari file. I prototipi di tali funzioni sono riportate in ***file header*** (estensione ***.h***)

La compilazione di un programma che utilizzi tali funzioni richiede l'inclusione del file header della libreria che si vuole utilizzare:

Esempio:

#include <stdio.h>

Funzioni standard che riguardano:

- input/output *stdio.h*
- funzioni matematiche *math.h*
- gestione dinamica della memoria *stdlib.h*
- gestione di caratteri e stringhe *string.h*
- ricerca ed ordinamento *stdlib.h*
- interazione con il sistema operativo

Esempi di funzioni per la gestione di stringhe:

#include <string.h>

- Funzione **strlen:** restituisce la lunghezza di una stringa di caratteri

```
strlen(char s[ ])
{
  int j;
  for (j = 0; s[j] != '\0'; j++) ;
            /* scansione */
  return j;
}
```

- Funzione **strcmp:** confronta due stringhe di caratteri, s1 e s2, e restituisce un valore:

< 0	se	s1 < s2
0	se	s1 == s2
> 0	se	s1 > s2

```
strcmp(unsigned char s1[ ],
       unsigned char s2[ ])
{
  int j = 0;
  while (s1[j] && s2[j] &&
        s1[j] == s2[j])  j++;
  return (s1[j] – s2[j]);
}
```

- Funzione **strcpy:** copia una stringa (sorgente) in un'altra (destinazione) e restituisce l'***indirizzo*** della stringa destinazione

```
char *strcpy(char dest[ ],char src[ ])
{
 int j = 0;
 do
  dest[j] = src[j];
 while (src[j++])
 return dest;/* return &dest[0] */
}
```

Argomenti delle linee di comando: *argc, argv*

Anche la funzione ***main*** puo‘ avere parametri. I parametri rappresentano gli eventuali argomenti passati al programma, quando viene messo in esecuzione:

prog arg1 arg2 ... argN

I parametri formali di main, differentemente dalle altre funzioni, sono sempre due:

- argc
- argv

int argc:

e` un parametro di tipo ***intero***. Rappresenta il numero degli argomenti effettivamente passati al programma nella linee di comando con cui si invoca la sua esecuzione. Anche il nome stesso del programma (nell'esempio, *prog*) e` considerato un argomento, quindi argc vale sempre almeno 1.

char **argv:

vettore di stringhe, ciascuna delle quali contiene un diverso argomento. Gli argomenti sono memorizzati nel vettore nell'ordine con cui sono dati dall'utente.
Per convenzione, ***argv[0]*** contiene il ***nome del programma stesso***.

prog arg1 arg2 ... argN

- **Quindi:**

```
argv[0]="prog"
argv[1]="arg1"
argv[2]="arg2"
...
argv[N]="argN"
```

Per convenzione, ***argv[argc]*** vale ***NULL***.

Esempio:

Programma che stampa i suoi argomenti.

```
#include <stdio.h>
/* programma esempio.exe */
main(int argc, char *argv[])
{
  int i;

  for(i=0; i<argc; i++)
    printf("%s%s",argv[i],
              (i<argc-1)?"\t":"\n");
  return 0;
}
```

Invocazione:

esempio a b c zeta

Cosa stampa?