Complementi di C

Luca Mazzei

Si raccolgono in questa dispensa alcuni complementi di programmazione C utili per affrontare il corso di Statistica Computazionale.

1 Leggere l'ora di sistema

Per leggere l'ora dell'orologio dell'elaboratore su cui si sta eseguendo il programma occorre usare una serie di funzioni contenute nella libreria `time.h`:

`time_t`	Tipo predefinito dalla libreria compatibile con `int` indica, in genere, il tempo trascorso in secondi dalla mezzanotte del 1 Gennaio 1970
`time_t time ( time_t * timer );`	Regola l'orologio di sistema all'ora contenuta nel parametro `timer`; il parametro può anche valere `NULL`. Restituisce l'ora del sistema; in caso di errore `-1`.
`const struct tm`	`struct` predefinita dalla libreria che contiene una data spezzata nelle sue componenti. Contiene i seguenti campi di tipo `int`

Nome Campo	Significato	Intervallo
`tm_sec`	Secondi dopo i minuti	0-61
`tm_min`	Minuti dopo le ore	0-59
`tm_hour`	Ore dalla mezzanotte	0-23
`tm_mday`	Giorno del mese	1-31
`tm_mon`	Mese (gennaio=0)	0-11
`tm_year`	Anno (dal 1900)	
`tm_wday`	Giorno della Settimana (Domenica=0)	0-6
`tm_yday`	Giorni dal 1 Gennaio	0-365
`tm_isdst`	Ora legale (0 non in vigore, 1 in vigore)	

`struct tm * localtime ( const time_t * timer );`	Restituisce una struct di tipo `tm` sulla base del valore del parametro
`char * asctime ( const struct tm * timeptr );`	Restituisce una stringa con la data contenuta nel parametro completa del carattere a capo (`\n`).
`char * ctime ( const time_t * timer );`	Restituisce una stringa con la data contenuta nel parametro completa del carattere a capo (`\n`).

L'esempio che segue mostra l'uso di questa libreria[1]:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
char *oraCompleta()
{
  time_t t;
  t = time(&t);
  return (asctime(localtime(&t)));
}
char *oraEssenziale()
{
```

[1] Nel codice si è usata la funzione `sprintf()` che si comporta come la funzione `printf()` eccetto per il primo argomento che deve essere una stringa opportunamente dimensionata dove inserire il risultato (in pratica invece di visualizzare il risultato lo scrive nella stringa che è il primo argomento).

Si è anche utilizzato il modificatore `%0xd` che permette di scrivere i numeri ad `x` cifre riempiendo le cifre non presenti a sinistra con la cifra `0`.

```
  time_t oraSis;
  struct tm * t;
  char *st;

  oraSis = time ( NULL );
  t = localtime ( &oraSis );
  st = (char *) calloc(21,sizeof(char));
  sprintf(st,"%02d/%02d/%04d %02d:%02d:%02d",
  t->tm_mday,t->tm_mon+1, t->tm_year+1900,
  t->tm_hour, t->tm_min, t->tm_sec);
  return st;
}
int main()
{
  printf("%s", oraCompleta());
  printf("%s\n", oraEssenziale());
  system("pause");
  return 0;
}
```

2 Generazione di numeri casuali

La generazione dei numeri casuali avviene in C/C++ con le funzioni contenute nella libreria `stdlib.h`; le funzioni da usare sono le seguenti:

`int rand ( void );`	genera un numero casuale fra 0 e `RAND_MAX` (questo valore dipende dal compilatore usato ed in genere è piuttosto piccolo)
`void srand ( unsigned int` *seed* `);`	Inizializza il generatore di numeri casuali al valore dato da *seed*.

Il programma che segue genera 10 numeri interi distribuiti in maniera casuale:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
  int i;
  for(i=0; i<10;i++)
    printf("%d ", rand());
  printf("\n RAND_MAX vale %d\n",RAND_MAX);
  system("pause");
```

```
  return 0;
}
```

Se viene richiamato più volte di seguito genererà sempre la stessa sequenza; per evitare questo problema occorre inizializzare ogni volta il generatore con la funzione `srand()`. utilizzando l'ora di sistema (funzione `time()` con parametro `NULL`) non è necessario chiedere all'utente un valore:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main()
{
  int i;
  srand(time(NULL));
  for(i=0; i<10;i++)
     printf("%d ", rand());
  printf("\n RAND_MAX vale %d\n",RAND_MAX);
  system("pause"); return 0;
}
```

Volendo generare numeri compresi fra 0 (incluso) ed 1 escluso occorre dividere per `RAND_MAX` i valori generati. Attenzione `RAND_MAX` è una costante di tipo `int` e quindi la divisione per il valore prodotto da `rand()` darà sempre zero. Il programma che segue mostra una maniera per risolvere il problema:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main()
{
 int i;
 srand(time(NULL));
 for(i=0; i<10;i++)
  printf("%f ", rand()/(1.0*RAND_MAX));
 system("pause");
 return 0;
}
```

Le funzioni contenute nella libreria GSL permettono di generare numeri casuali di qualità migliore.

3 Allocazione dinamica della memoria

Nel linguaggio C/C++ esiste la possibilità di riservare delle zone consecutive di memoria nella parte di Heap del programma mediante delle apposite funzioni. In questo modo è possibile gestire strutture dati dinamiche (liste o alberi) o creare dei vettori dinamici. Le principali funzioni C sono le seguenti e sono contenute nella libreria `stdlib.h`:

`void* calloc (size_t num, size_t size );`	Restituisce un blocco di memoria di atto a contenere `num` elementi di dimensione `size` o il valore `NULL` se ci sono stati dei problemi. La memoria allocata è riempita con il valore 0.
`void* malloc (size_t size );`	Restituisce un blocco di memoria di dimensione `size` o il valore `NULL` se ci sono stati dei problemi.
`void* realloc(void* ptr, size_t size );`	Rialloca la zona di memoria puntata da `ptr` ad una nuova dimensione indicate da `size`. Se la nuova dimensione è più grande di quella della zona di memoria puntata da `ptr` mantiene il contenuto della vecchia zona, se il valore è inferiore mantiene il contenuto fino alla dimensione riallocata. Se `size` vale 0 la memoria è deallocata ed è restituito un puntatore `NULL`. Questo valore è restituito anche quando vi sono problemi ad allocare la nuova memoria
`void free (void* ptr);`	Libera la zona di memoria indicata da `ptr`. Non restituisce nulla.

Si noti l'uso del puntatore `void` (`void *`), puntatore di tipo generico che va trasformato, con una opportuna operazione di cast nel tipo voluto, quando viene assegnato ad un valore di tipo puntatore. Il tipo `size_t` è un tipo definito dalla libreria ed è compatibile con `int` (di solito è `unsigned int`). L'esempio che segue mostra la creazione di un vettore dinamico riempito con valori casuali e la sua riallocazione ad una nuova dimensione (provare ad eseguirlo e a verificare le uscite):

```
#include <stdlib.h>
#include <stdio.h>
void stampa(int *v, int l)
{
  int i;
  for(i=0;i<l;i++)
    printf("%d ",v[i]);
  printf("\n");
}
int main()
{
 int *v, dim, i;
 dim=8;
 v =(int *) calloc(dim, sizeof(int));
 stampa(v,dim);
 for(i=0;i<dim;i++)
   v[i] = rand();
 stampa(v,dim);
 dim=4;
 v =(int *) realloc((int *)v, dim*sizeof(int));
 stampa(v,dim);
 dim=10;
 v =(int *) realloc((int *)v, dim*sizeof(int));
 stampa(v,dim);
 free(v);
 system("pause");
 return 0;
}
```

4 Passaggio di puntatori a funzioni

Come premessa occorre ricordare che ai programmi in esecuzione vengono riservate delle zona di memoria per conservare il codice, le variabili globali e lo stack per gestire il richiamo delle funzioni; in C/C++ è possibile accedere a queste zone di memoria.

In modo molto semplificato quando il programma incontra il richiamo di una funzione si comporta nel seguente modo:

1. **punto di richiamo della funzione**:
   - salvataggio su stack indirizzo della prossima istruzione da eseguire
   - salvataggio del contesto (situazione della memoria)
   - salto prima istruzione della funzione
2. **prima istruzione della funzione (punto di entrata)**:
   - gestione parametri
3. **ultima istruzione della funzione (punto di uscita)**:
   - ripristino contesto e gestione valore restituito dalla funzione
   - salto a istruzione successiva a quella in cui è stata richiamata la funzione (l'indirizzo era stato salvato sullo stack al punto 1)

In C/C++ l'indirizzo del punto di entrata di una funzione è legato al nome della funzione ed è di tipo puntatore a funzione. Sintatticamente si ha:

dichiarazione del puntatore	`tipo restituito (* nomeVar) (el. parametri);`
assegnamento	`nomeVar = nomeFunzione;`
uso (*si notino le parentesi*)	`(* nomeVar)(parametri);`

L'esempio che segue mostra l'uso dei puntatori a funzione:

```
#include <stdio.h>
int somma(int a, int b) {return(a+b);}
int differenza(int a, int b) {return(a-b);}
int main()
{
  int (* pf) (int, int); //dichiarazione variabile puntatore
  int v, w;
  pf = somma; //assegna il valore
  v = (* pf)(5,6); //uso
  pf = differenza; //assegna il valore
  w = (* pf)(7, 8); //uso
  printf("%d\t%d\n",v,w);
  return(0);
}
```

I puntatori a funzione trovano l'applicazione più adeguata nel loro passaggio ad un'altra funzione in modo da rendere l'uso della stessa il più generico possibile. Ciò avviene secondo la seguente sintassi:

parametro attuale:	si passa il nome della funzione
parametro formale definizione:	`nomeFunz(... tipo(* NomePar)(parametri)...)`
parametro formale uso:	`(* NomePar)(parametri);`

Vediamo un semplice esempio:

```
#include <stdio.h>
int somma(int a, int b) {return(a+b);}
int diff(int a, int b) {return(a-b);}
int leggiEfai(int (* pf)(int, int)) //definizione parametro formale
{
  int a,b;
  scanf("%d%d",&a,&b);
  return((*pf)(a,b));   //uso del parametro formale
}
int main()
{
```

```
  printf("%d\n",leggiEfai(somma)); //si passa il nome della funzione
  printf("%d\n",leggiEfai(diff));  //si passa il nome della funzione
  return 0;
}
```

Si supponga di dover ordinare un vettore di interi in modo decrescente mediante l'algoritmo di bubble sort:

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
void BubbleSort(int *vt, int Nel)
{
 int i, ord=FALSE, temp;
 while (!ord)
 {
  ord = TRUE;
  for(i=0;i<(Nel - 1);i++)
   {
    if(vt[i] < vt[i+1]) //DETERMINANTE PER IL VERSO DELL?ORDINAMENTO
     { temp = vt[i];
       vt[i] = vt[i+1];
       vt[i+1] = temp;
       ord = FALSE; }
   }
 }
}
void stampa(int *vt, int l)
{
  int i;
  for(i=0;i<l; i++)
    printf("%d\t",vt[i]);
  printf("\n\n");
}
int main()
{
  int vt[5]={4,-76,23,-9,-456}; int i;
  stampa(vt, 5);
  BubbleSort(vt, 5);
  stampa(vt,5);
  system("pause");
```

```
  return(0);
}
```

Nell'algoritmo l'unica parte che è determinante per il verso dell'ordinamento è la condizione evidenziata; quindi modificando la condizione si potrebbe ottenere un ordinamento decrescente. Se alla funzione `BubbleSort()` potessimo passare una funzione che abbia questo scopo il problema sarebbe risolto senza dover scrivere un'ulteriore funzione:

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
/*funzioni per ordinare ritornano 1 o 0*/
int CmpCre(int a, int b) {return(a<b);} //crescente
int CmpDec(int a, int b) {return(a>b);} //decrescente
void BubbleSort(int *vt, int Nel, int (* fp)(int,int))
{
  int i, ord=FALSE, temp;
  while (!ord)
  {
    ord = TRUE;
    for(i=0;i<(Nel - 1);i++)
     {
       if((* fp)(vt[i],vt[i+1]))
         {
           temp = vt[i];
           vt[i] = vt[i+1];
           vt[i+1] = temp;
           ord = FALSE;
         }
     }
    }
}
void stampa(int *vt, int l)
{
  int i;
  for(i=0;i<l; i++)
    printf("%d\t",vt[i]);
  printf("\n\n");
}
int main()
```

```
{
  int vt[5]={4,-76,23,-9,-456}; int i;
  stampa(vt, 5);
  BubbleSort(vt, 5,CmpCre);/*ord. crescente*/
  stampa(vt,5);
  BubbleSort(vt, 5,CmpDec);/*ord. decrescente*/
  stampa(vt,5);
  system("pause");
  return(0);
}
```

Le due modifiche permettono di generalizzare la funzione.

4.1 Funzioni di libreria che hanno come argomento puntatori a funzione

Adesso si applicano le idee viste a due funzioni di libreria del C/C++ `qsort()` e `bserach()` che, rispettivamente, ordinano un vettore secondo l'algoritmo Quick Sort, ricercano un elemento in un vettore ordinato con l'algoritmo di ricerca binaria.

4.1.1 `qsort()`

```
void qsort(void *base, size_t nel, size_t dim,
       int(* comp)(const void*,const void*));
```

I parametri hanno il seguente significato:

`void *base`	Puntatore al primo elemento del vettore da ordinare
`size_t nel`	Numero elementi del vettore
`size_t dim`	Dimensione di un elemento del vettore
`int(* comp)(const void*,const void*)`	Funzione da usare per il confronto deve restituire un valore intero: • valore <0 se primo argomento minore secondo argomento • valore =0 se primo argomento uguale secondo argomento • valore >0 se primo argomento maggiore secondo argomento

L'esempio applica la funzione di libreria per ordinare un vettore di strutture:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{int a,b;}EL;
int CmpAsc(const void *a, const void *b)
{
  return (((EL *) a)->b - ((EL *) b)->b);
}
int main()
{
  EL vt[5]={{4,89},{-76, 34},{23,12}, {1,-9},{3,-456}};
  int i;
  for(i=0;i<5; i++)
    printf("%d\t%d\n",vt[i].a,vt[i].b);
  printf("\n\n");
  qsort(vt, 5, sizeof(EL), CmpAsc);
  for(i=0;i<5; i++)
    printf("%d\t%d\n",vt[i].a,vt[i].b);
  system("pause");
  return(0);
}
```

4.1.2 `bsearch()`

void * bsearch (const void * key, const void * base, size_t num,
size_t size,
int (* comp) (const void *, const void *)); La funzione restituisce un puntatore ad un elemento che corrisponde al criterio di ricerca, `NULL` se non trova corrispondenza. I parametri hanno il seguente significato:

`const void * key`	Puntatore all'oggetto da cercare
`const void * base`	Puntatore al primo elemento del vettore
`size_t num`	Numero elementi del vettore
`size_t size`	Dimensione di un elemento del vettore

`int (* comp) (const void*, const void*)`	Funzione da usare per il confronto deve restituire un valore intero: • valore <0 se primo argomento minore secondo argomento • valore =0 se primo argomento uguale secondo argomento • valore >0 se primo argomento maggiore secondo argomento

Infine un esempio di uso:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{int a,b;}EL;
/*
si cerca una corrispondenza solo sul
secondo campo (solo rispetto ad esso
e' ordinato il vettore
*/
int CmpAsc(const void *a, const void *b)
{
  return (((EL *) a)->b - ((EL *) b)->b);
}
int main()
{
  EL vt[5]={{4,89},{-76, 34},{23,12},{1,-9},{3,-456}}, *rit, cerca;
  int i;
  qsort(vt, 5, sizeof(EL), CmpAsc);
  cerca.a=9; cerca.b=-693;
  rit = (EL *)bsearch(&cerca,vt,5,sizeof(EL),CmpAsc);
  if (rit == NULL)
    printf("non ci sta\n");
  else
    printf(" %d\t%d\n",rit->a,rit->b);
  system("pause");
```

```
   return(0);
}
```

Ovviamente ordinare un vettore per poi fare delle ricerche con il metodo della ricerca dicotomica non è un modo efficiente di risolvere il problema.

4.1.3 Algoritmo generalizzato di Bubble Sort

Si mostrerà come generalizzare l'algoritmo di Bubble Sort per un vettore di strutture di tipo generico, ordinando rispetto ad uno qualunque dei suoi campi. La funzione generalizzata ha il seguente codice:

```
void BubbleSort(void *vt, size_t Nel, size_t dim,
int (* cmp)(void *,void *))
{
  int i, ord=FALSE; void *temp;
  temp = calloc(1,dim);
  while (!ord)
    {
      ord = TRUE;
      for(i=0;i<(Nel - 1);i++)
        {
          if((* cmp) (((char *)vt+(i*dim)),((char *)vt+(i+1)*dim)))
            {
              memcpy(temp, ((char *)vt + i*dim), dim);
              memcpy(((char *)vt + i *dim), ((char *)vt + (i+1)*dim),dim);
              memcpy(((char *)vt + (i+1) * dim), temp, dim);
              ord = FALSE;
            }
       }
   }
  free(temp); /*rilascio zona di memoria*/
}
```

L'uso dei puntatori generici che obbliga ad utilizzare la funzione `memcpy()`[2] per copiare il contenuto delle zone di memoria invece degli assegnamenti (non

[2]La funzione

```
void* memcpy (void *destination, const void * source, size_t num );
```

contenuta nella libreria `string.h`, ha i seguenti parametri `destination`, indirizzo di memoria in cui copiare, `source`, indirizzo di memoria da cui copiare, `num`, numero di byte da copiare. I parametri `destination` e `source` debbono avere la stessa dimensione ed essere distinti. Copia byte per byte il contenuto della memoria

si sa all'interno della funzione il tipo dei dati nel vettore ma solo la dimensione attraverso il parametro `dim` ed il numero attraverso il parametro `Nel`). Si noti che il puntatore alla funzione per il confronto `(* cmp)` ha anch'esso dei puntatori generici per lo stesso motivo. Ricapitolando i parametri da passare sono:

`void *vt`	Vettore da ordinare (puntatore al primo elemento)
`size_t Nel`	Numero elementi del vettore
`size_t dim`	Dimensione di un elemento del vettore
`int (* cmp)(void *,void *)`	Funzione che gestisce il confronto ha come argomento due elementi del vettore e restituisce un intero utilizzato nell'istruzione di confronto per ordinare il vettore

Per lo scambio si usa un elemento temporaneo `temp` allocato dinamicamente all'inizio della funzione e poi rilasciato al suo termine. Il tutto avviene utilizzando l'aritmetica dei puntatori e la funzione `memcpy()`.

```
(* cmp) (((char *)vt+(i*dim)),(char *)vt+(i+1)*dim)
```

il parametro richiede l'uso del cast `(char *)` altrimenti non si è in grado di calcolare di quanto ci si deve spostare rispetto all'inizio del vettore `(i*dim)`. L'istruzione equivale a **vt[i]** (ma non si può fare che così) similmente per il secondo parametro. Il codice che segue mostra un esempio ordinando un vettore di strutture rispetto al campo `b`:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
typedef struct{int a,b;}EL;
/*funzioni per ordinare ritornano 1 o 0*/
int CmpAsc(void *a, void *b)
{
  return(((EL *)a)->b >((EL *)b)->b);
}
int CmpDesc(void *a, void *b)
{
  return(((EL *)a)->b < ((EL *)b)->b);
}
```

```
void BubbleSort(void *vt, size_t Nel, size_t dim, int (* cmp)(void *,void *))
{
  int i, ord=FALSE; void *temp;
  temp = calloc(1,dim);
  while (!ord)
   {
     ord = TRUE;
     for(i=0;i<(Nel - 1);i++)
       {
          if((* cmp) (((char *)vt+(i*dim)),((char *)vt+(i+1)*dim)))
            {
              memcpy(temp, ((char *)vt + i*dim), dim);
              memcpy(((char *)vt + i *dim), ((char *)vt + (i+1)*dim),dim);
              memcpy(((char *)vt + (i+1) * dim), temp, dim);
              ord = FALSE;
            }
      }
   }
  free(temp); /*rilascio zona di memoria*/
}
int main()
{
  EL vt[5]={{4,89},{-76, 34},{23,12},{1,-9},{3,456}};
  int i;
  for(i=0;i<5; i++)
    printf("%d\t%d\n",vt[i].a,vt[i].b);
  printf("\n\n");
  BubbleSort(vt, 5, sizeof(EL), CmpAsc);
  for(i=0;i<5; i++)
    printf("%d\t%d\n",vt[i].a,vt[i].b);
  system("pause");
  return(0);
}
```

5 Nozioni per la traduzione dal Pascal al C

Si ricorda che in Pascal il carattere minuscolo è equivalente al quello maiuscolo (non è *case sensitive*) come il C. Quindi `BEGIN` equivale, ad esempio, a `BeGin`. I commenti in Pascal possono essere racchiusi fra `(*` e `*)` oppure fra parentesi graffe.

5.1 Tabella di equivalenza degli operatori

Pascal	C
<>	`!=`
`:=`	`=`
`=`	`==`

5.2 Il tipo `subrange` in C

Le variabili di tipo `0..maxint` o `1 .. maxint` possono essere dichiarate di tipo `unsigned`;

5.3 il tipo `const` del Pascal

Il tipo `const` del Pascal può essere tradotto in C in due modi:

- con `#define nomeCostante valore` dopo le istruzioni `#include` senza `;` finale.
- con `const tipo nomeVariabile=valore`

Quindi:

Pascal	C (modo1)	C (modo 2)
`const v=1E-9;`	`#define v 1E-9`	`const double v=1E-9;`

5.4 Il tipo `boolean` del Pascal

Il tipo `boolean` del Pascal si può essere tadurre come `int` con la convenzione che 0 è falso e diverso da 0 è vero; oppure con il tipo C++ `bool`.

5.5 Il ciclo `repeat ... until` in Pascal

In Pascal esiste il ciclo `repeat ... until(condizione)` che ripete un gruppo di istruzioni finché non si verifica una determinata condizione. Questo ciclo si rende in C utilizzando il ciclo `do ... while(condizione)`, negando

(ad esempio con le leggi di De Morgan) la condizione del ciclo `repeat`. In C i cicli hanno sempre una condizione di permanenza nel ciclo.

Pascal	**C**

`Repeat`
`  istr;`
`Until (x>0);`
	`do{`
	`  istr`
	`}while(x<=0);`

5.6 Il tipo `enum` del C

6 Uso del tipo enum

I programmi Pascal contengono delle variabili di tipo insieme che in pratica permettono di sostituire un valore con un nome. Ad esempio: `state: (iterating, withintol, maxitsreached);`

In C un modo analogo (non equivalente) è usare delle variabili di tipo `enum`.In questo modo è possibile definire un unico insieme di valori associati ad una variabile. Ad esempio `enum {iterating, withintol, maxitsreached } state;`

Ai nomi contenuti fra parentesi graffe è associato un valore numerico a partire da 0, quindi nell'esempio ad `iterating` è associato 0, ad `withintol` è associato 1 e a `maxitsreached` è associato 2. Sui nomi è associato un controllo di tipo, è possibile usare gli operatori aritmetici, ed assegnare direttamente dei numeri alla variabile (anche fuori dall'intervallo):

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ enum {iterating, withintol, maxitsreached } state;
  state = iterating;
  // state = itera; //errato itera non è uno dei possibili nomi
  state = withintol;
  printf("%d \n",state);
  state++;
  printf("%d \n",state);
  if (state == maxitsreached)
    printf("uguale\n");
  if (state == 2)
    printf("uguale\n");
  state = 4;
system("pause");return 0;
}
```

Un uso corretto e consistente del tipo prevede di usarlo solo attraverso i nomi assegnati in fase di definizione. L'esempio che segue mostra alcuni modi diversi di dichiarare ed usare un tipo `enum`.

```
#include <stdio.h>
typedef enum {v1, v2, v3} VL;
enum VE {vz1, vz2, vz3};
void funz ()
{ VL valori;
  enum VE val;
  enum {x1, x2, x3} vari; //a voi basta questo
  valori = v1;
  val = v3;
  printf("%d %d\n",val,sizeof(val));
}
int main()
{
  funz();
  return 0;
}
```

Ecco un uso di `enum` per definire anche in C il tipo `boolean` del Pascal:

```
#include <stdio.h>
#include <stdlib.h>
typedef enum {true, false} boolean;
int main()
{
  boolean success;
  if (3 > 4)
    success = false;
  else
    success = true;
  printf("%d\n",success);
system("pause"); return 0;
}
```

6.1 Il concetto di funzione in Pascal e C

In Pascal esiste il concetto di funzione e di procedura. In C il concetto di procedura può vedersi equivalente ad una funzione che non restituisce nulla (`void`):

Pascal	C

procedure x ()
begin
...
end;
	void x ()
	{
	...
	}

Le funzioni hanno in Pascal la caratteristica che il nome della funzione è anche usato per restituire il valore alla funzione chiamante. In C occorre definire una variabile locale alla funzione[3] e poi far restituire questa variabile mediante l'istruzione `return`.

Pascal	C

Function (a: real): real
Var b: real;
Begin
b := a * 2;
f : = a;
End;
	float f (float a)
	{
	float b;
	b = a * 2;
	return(a);
	}

I parametri preceduti dalla parole chiave `var` sono parametri per indirizzo ed inoltre in Pascal una serie di parametri dello stesso tipo è separata da virgole.

```
Procedure Bisect
(
  function f(x: real): real;      (*passaggio di funzione*)
  x0, x1 : pospoints;            (*parametri per valore di tipo posints*)
  Var root, fatroot: real;       (* parametri per indirizzo di tipo real*)
  Var noofits: posints;          (*parametro per indirizzo di tipo posints*)
  Var converged: boolean        (*parametro per indirizzo di tipo boolean*)
);
```

Infine ecco la funzione di prova da usare nei programmi:

```
#include <stdio.h>
#include <math.h>
double f(double x)
{
  return(x-exp(1/x));
```

[3]Il nome della variabile deve essere diverso dal nome della funzione.

```
}
void stampa(double (*fx) (double))
{
  int i;
  double passo = 0.001,x;
  for(i=0;i<10; i++)
   {x = 1.0 + i*passo;
    printf("f(%f)=%f\n",x,fx(x));}
}
int main()
{
  stampa(f);
  return 0;
}
```

6.1.1 Esempio di traduzione di un algoritmo da Pascal a C

```
procedure Bisect
(
 function f(x:  real):  real;
  x0, x1, xtol:  real;
  maxits:  posints;
  var root, fatroot:  real;
  var noofits:  posints;
  var converged:  boolean);
 var
 x, fx,fo:  real;
 itcount:  0..maxint;
 state:  (iterating,
withintol,maxitsreached);
begin
 f0 :=f(x0);
 itcount := 0; state:= iterating;
 repeat
  itcount := itcount + 1;
  x :=(x0+x1)/2; fx:=f(x);
  if f0 * fx <= 0 then
   x1 := x
  else
   begin
    x0:=x; f0:=fx
   end;
  if (x1-x0) <= xtol then
   state :=withintol
  else
   if (itcount=maxits) then
    state:= maxitsreached;
  until state <> iterating;
 converged := state =withintol;
 root := (x0+x1)/2;
 fatroot:=f(root);
 noofits := itcount
end
```

```
void Bisect(
 double (* f)(double),
 double x0, double x1,
 double xtol,
 unsigned maxits,
 double * root, double *fatroot,
 unsigned * noofits,
 int *converged
)
{
 double x, fx,fo ;
 unsigned itcount ;
 enum {iterating,
withintol,maxitsreached} state;
 f0 = f(x0);
 itcount = 0;
 state = iterating;
 do{
  itcount = itcount + 1;
  x=(x0+x1)/2; fx=f(x);
  if (f0 * fx <= 0)
   x1 = x;
  else
  {
   x0 = x; f0 = fx;
  }
  if ((x1-x0) <= xtol)
   state = withintol;
  else
   if (itcount == maxits)
    state = maxitsreached;
 }while (state == iterating);
 *converged = state == withintol;
 *root = (x0+x1)/2;
 fatroot = f(*root);
 *noofits = itcount;
 }
}
```

```
Bisect(f,1,2,tol,50,x,fx,its,success)
```

`const`
`  tol = 5E-7;`
`type`
`  posints = 1...maxint;`
`var`
`  x,fx:  real;`
`  its:  posits;`
`  success:  boolean;`
	`const double tol = 5E-74;`[4]
	`typedef unsigned posints`[5]`;`
	`double x,fx;`
	`posits its;`
	`int`[6] `success;`

`function f(x:  real):  real;`
`begin`
` f:=x-exp(1/x)`
`end`
	`double f(double x)`
	`{`
	` return(x-exp(1/x));`
	`}`

[4] Da dichiarare al di fuori di ogni funzione. Si potrebbe usare anche `#define tol 5E-7`.

[5] Come detto si può definire direttamente come `unsigned` ma facendo come indicato si crea una distinzione logica fra il tipo `unsigned` ed il tipo ridefinito.

[6] Il tipo boolean potrebbe essere definito attraverso `enum` come visto in precedenza.