Operatori sui bit.

Il C mette a disposizione degli operatori che lavorano su numeri di tipo intero (char, int, long int) manipolando il dato a livello di singolo bit. Non si applicano ad operandi floating point o a dati di tipo strutturato.

&	AND (bit a bit)
|	OR "
^	OR ESCLUSIVO (XOR)
~	NOT (complemento ad uno)
>>	SHIFT (SCORRIMENTO) a DESTRA
<<	SHIFT (SCORRIMENTO) a SINISTRA

L'operatore AND **&** effettua un and bit a bit tra due operandi, ovvero setta ad 1 un bit in una certa posizione quando **entrambi** i bit in quella posizione nei due operandi valgono 1, altrimenti lo setta a 0.

unsigned char op1, op2, op3;

op1=63;	0 0 1 1 1 1 1 1
op2=240; ;	1 1 1 1 0 0 0 0
op3 = op1 **&** op2;	0 0 1 1 0 0 0 0
op3 assume valore 48	

L'operatore OR **|** effettua un or bit a bit tra due operandi, ovvero setta ad 1 un bit in una certa posizione quando **almeno uno** dei bit in quella posizione dei due operandi vale 1 altrimenti lo setta a 0.

unsigned char op1, op2, op3;

op1=60;	0 0 1 1 1 1 0 0
op2=240; ;	1 1 1 1 0 0 0 0
op3 = op1 | op2;	1 1 1 1 1 1 0 0
op3 assume valore 252	

L'operatore XOR ^ effettua un or esclusivo bit a bit tra due operandi, ovvero setta ad 1 un bit in una certa posizione quando i bit in quella posizione nei due operandi sono diversi, altrimenti lo setta a 0.

unsigned char op1, op2, op3;

op1=60;	0 0 1 1 1 1 0 0
op2=240; ;	1 1 1 1 0 0 0 0
op3 = op1 ^ op2;	1 1 0 0 1 1 0 0
op3 assume valore 204	

Questo operatore presenta una tabella di verità (bit a bit) siffatta:

bit op1	bit op2	bit op1^op2
0	0	0
1	0	1
0	1	1
1	1	0

L'operatore NOT ~ è un operatore unario che inverte lo stato di ogni bit, cioè setta ad 1 i bit che valgono 0, e viceversa setta a 0 i bit che valgono 1.

op1=60;	0 0 1 1 1 1 0 0
op3 = ~op1;	1 1 0 0 0 0 1 1
op3 assume valore 195	

Gli operatori di scorrimento (SHIFT) sono operatori unari che realizzano lo spostamento a sinistra o a destra dei bit di una variabile di tipo intero, **mettendo a 0 i bit che entrano**, cioè **non** rientrano da sinistra i bit usciti da destra. La forma generale dell'istruzione di SHIFT A DESTRA è del tipo:

variabile_intera >> numero_posizioni

Significa che i bit di *variabile_intera* vengono tutti spostati a destra di *numero_posizioni*.

op1=61;	0 0 1 1 1 1 0 1
op3 = op1>>1;	***0*** 0 0 1 1 1 1 0
op3 assume valore 30	(in neretto il bit entrante)

Si noti come l'operatore corrisponda ad una divisione intera per multipli di 2. Shiftare a destra di 1 significa dividere per 2, shiftare di 2 è dividere per 4.

op3 = op1>>3;	***0 0 0*** 0 0 1 1 1
op3 assume valore 7	(in neretto i bits entranti)

La forma generale dell'istruzione di SHIFT A SINISTRA è del tipo:

variabile_intera **<<** ***numero_posizioni***

Significa che i bit di *variabile_intera* vengono tutti spostati a sinistra di *numero_posizioni*. I bits entranti da destra sono posti a 0.

op1=61;	0 0 1 1 1 1 0 1
op3 = op1<<1;	0 1 1 1 1 0 1 ***0***
op3 assume valore 122	(in neretto il bit entrante)

Notare come questo operatore corrisponda al risultato di una moltiplicazione intera per un multiplo di due finchè a sinistra non esce qualche bit ad 1. Shiftare a sinistra di 1 significa moltiplicare per 2 (se non escono bit ad 1).

op3 = op1<<3;	1 1 1 0 1 ***0 0 0***
op3 assume valore 232	(in neretto i bits entranti)

Operatori Compositi sui bit.

Il C mette a disposizione anche operatori che coniugano un'operazione bit a bit all'operazione di assegnamento.

AND bit a bit e Assegnamento	x &= y	equivale a	x = x&y
OR bit a bit e Assegnamento	x |= y		x = x|y
XOR bit a bit e Assegnamento	x ^= y		x = x^y
Shift a Destra e Assegnamento	x >>= y		x = x>>y
Shift a Sinistra e Assegnamento	x <<= y		x = x<<y

Alcune Funzioni della libreria C standard.

Di seguito elenchiamo le più importanti funzioni della libreria standard dell'ANSI C. Informazioni più dettagliate possono essere ottenute dalle chiamate agli help in linea o ai man nei sistemi Unix.

Manipolazione dei buffer

#include <memory.h>

void *memchr (void *s, int c, size_t n) - Cerca un carattere **c** nei primi **n** caratteri di un buffer puntato da **s**. Restituisce un puntatore al primo carattere c trovato, o NULL se non lo trova.

int memcmp (void *s1, void *s2, size_t n) - Paragona i primi **n** byte di due buffers s1 ed s2. Restituisce un risultato minore di zero se s1<s2, uguale a zero se s1=s2, maggiore di zero se s1>s2.

void *memcpy (void *dest, void *src, size_t n) - Copia n byte di un buffer in un altro. Causa problemi se le due aree sono sovrapposte.

void *memmove (void *dest, void *src, size_t n) - - Copia n byte di un buffer in un altro. Funziona correttamente anche se le due aree sono sovrapposte. Restituisce un puntatore alla destinazione.

void *memset (void *s, int c, size_t n) - Setta tutti i bytes di un buffer ad un dato carattere. Restituisce un puntatore al buffer s.

Classificazione di caratteri

#include <ctype.h>
int isalnum(int c) - Vero se "c" e' alfanumerico, '0'...'9' 'a'...'z' 'A'...'Z'.
int isalpha(int c) - Vero se "c" e' una lettera dell'alfabeto.
int iscntrl(int c) - Vero se "c" e' un carattere di controllo, i primi 31.
int isdigit(int c) - Vero se "c" e' un numero decimale.
int islower(int c) - Vero se "c" e' una lettera minuscola.
int isprint(int c) - Vero se "c" e' un carattere stampabile.
int ispunct (int c) - Vero se "c" e' un carattere di punteggiatura.
int isspace(int c) - Vero se "c" e' un carattere spazio cioè tab o blanck o newline o carriage return..
int isupper(int c) - Vero se "c" e' una lettera maiuscola.
int isxdigit(int c) - Vero se "c" e' un numero esadecimale.
int toascii(int c) - Converte "c" in ASCII, settando a zero il bit piu' significativo.
tolower(int c) - Converte "c" in minuscolo.
int toupper(int c) - Converte "c" in maiuscolo.

Conversione dei dati

#include <stdlib.h>
double atof(char *string) -Converte una stringa in un valore floating point.
int atoi(char *string) - Converte una stringa in un valore integer.
int atol(char *string) - Converte una stringa in un valore long integer.
char *itoa(int value, char *string, int radix) - Converte un valore integer in una stringa utilizzando il "radix" come base per la conversione (es 10 per conversione di numeri in base 10).
Le seguenti funzioni effeftuano un maggiore controllo sull'errore:
char *ltoa(long value, char *string, int radix) - Converte un valore long integer in una stringa in un dato "radix".
double strtod(char *string, char *endptr) - Converte una stringa in un valore in floating point.
long strtol(char *string, char *endptr, int radix) - Converte una stringa in un valore long integer utilizzando un dato "radix".
unsigned long strtoul(char *string, char *endptr, int radix) - Converte una stringa in un valore long senza segno.

Funzioni Matematiche

solo alcune delle funzioni matematiche:

#include <math.h>

int abs (int n) - valore assoluto di un intero.
double acos(double x) - arcocoseno di x.
double asin(double x) - arcoseno di x.
double atan(double x) - arcotangente
double ceil(double x) - il piu piccolo intero maggiore o uguale a x.
double cos(double x) - coseno di angolo x in raduanti
double sin(double x) - seno di angolo x in raduanti
double exp(double x - esponenziale
double fabs (double x) - valore assoluto di un double
double floor(double x) - il piu grande intero minore o uguale a x.
labs(long n) - valore assoluto di un long
double log(double x) - logaritmo naturale
double log10 (double x) - logaritmo in base 10
double pow (double x, double y) - potenza, x elevato alla y.
void srand(unsigned seed) - inizializza il generatore di numeri casuali con un certo valore.
int rand (void) - genera un numero casuale tra 0 e 32.
int random(int max_num) - genera un numero casuale tra 0 e max_num.
void randomize(void) - inizializza il generatore di numeri casuali.
double sqrt(double x) - radice quadrata
double tan(double x) - tangent di un angolo in radianti

Allocazione di Memoria

#include <malloc.h>
void *calloc(size_t num elems, size_t elem_size) - Alloca un vettore ed inizializza tutti gli elementi a zero.
void free(void *mem address) - Libera un blocco di memoria.
void *malloc(size_t num bytes) - Alloca un blocco di memoria.

Funzioni su stringhe

#include <string.h>
int strcmp(char *string1, char *string2) - Confronta string1 e string2 per determinare l'ordine alfabetico.
char *strcpy(char *string1, char *string2) - Copia string2 in stringl.
int strlen(char *string) - Determina la lunghezza di una stringa.
char *strncat(char *string1, char *string2, size_t n) - Aggiunge "n" caratteri di string2 in string1.
int strncmp(char *string1, char *string2, size_t n) - Confronta i primi "n" caratteri di due stringhe.
char *strncpy(char *string1, char *string2, size_t n) - Copia i primi "n" caratteri di string2 in string1.

Funzioni per I/O su stringhe

int sprintf(char *string, char *format_string, args)
Scrive output formattato su una stringa, comportamento analogo alla printf, ma scrive su una stringa.

int sscanf(char *buffer, char *format_string, args)
Legge input formattato da una "string", comportamento analogo alla scanf, ma prende l'input da una stringa

Funzioni per cattura del tempo

```
#include <time.h>
```

time_t **time**(time_t *t);
restituisce il tempo trascorso a partire dal 1 gennaio del 1979, misurato in secondi. Se il puntatore passato come argomento e' diverso da NULL, scrive lo stesso valore nella locazione di memoria puntata dal puntatore.
In caso di errore restituisce -1 e setta errno in modo appropriato.

int **gettimeofday**(struct timeval *tv, struct timezone *tz);
l'argomento di tipo timezone e' obsoleto, e viene istanziato a NULL.
La struttura di tipo timeval e' così definita:

```
struct timeval {
        long tv_sec;        /* seconds */
        long tv_usec;  /* microseconds */
};
```

La funzione scrive nella struttura puntata da tv il valore del clock corrente, ed e' quindi piu' precisa della time che mette a disposizione solo un valore in secondi. Restituisce 0 se tutto OK, altrimenti -1 se si verifica qualche errore.

clock_t **clock**(void);
restituisce il tempo di CPU utilizzato dal processo chiamante, a partire dalla prima volta che è stata chiamata la clock(). Il tempo restituito è la somma del tempo utilizzato dal processo chiamante e dai processi di sistema nell'esecuzione delle chiamate di sistema effettuate dal processo chiamante. Il valore restituito è convenzionalmente il numero di clock eseguiti. Per ottenere il tempo in secondi deve essere diviso per la costante CLOCKS_PER_SEC.

```
clock_t    start, end;
start = clock();
    istruzioni del programma ...
end = clock();
printf("tempo consumato in secondi: %lf\n",
                        (double)(end-start) / (double)CLOCKS_PER_SEC  );
```

I/O ad Accesso Casuale su file.

Un file può essere acceduto in lettura o scrittura anche in una maniera diversa da quella sequenziale, cioè è possibile spostarsi in un file aperto ovvero spostare il cosiddetto puntatore alla posizione corrente nel file. Questo puntatore è un intero e indica il prossimo byte da leggere o scrivere. Il primo byte del file è indicato da 0, il secondo da 1 e così via.
E possibile spostare la posizione corrente utilizzando una primitiva C:

int **fseek**(FILE *stream, long offset, int ptrname);
fseek stabilisce la nuova posizione del prossimo byte da leggere o scrivere. La nuova posizione viene indicata da due parametri: una base indicata da **ptrname** che puo' essere o l'inizio del file (SEEK_SET) o la posizione corrente (SEEK_CUR) o la fine del file (SEEK_END), ed uno scostamento cioè la distanza dalla base espressa in numero di byte, che è il parametro offset.
La funzione restituisce -1 in caso di errore, 0 se tutto OK.

```
/* lettura del byte in posizione 135 nel file */
int c;
if ( ! fp=fopen("prova.dat","rt") )
       exit(1);
if (  (fseek(fp, (long)135, SEEK_SET)) <0 )
         exit(2);              /* errore */
c = getc(fp);
```

/* ESEMPIO semplice semplice: il fattoriale */

```
#include <stdio.h>

unsigned long int fattoriale(unsigned int n)
{
	unsigned fatt=1;
	while(n>0)
		fatt *= n--;
	return(fatt);
}

void main(void)
{
	unsigned int n;
	scanf("%d",&n);
	printf("%lu\n",fattoriale(n));
}
```

/* ESEMPIO classico di uso di puntatori: string copy */

```
#include <stdio.h>

char *strcpy(char *dest, char *source)
{
      char *ptrdest=dest;
      while( *dest++  =  *source++  );
      return(ptrdest);
}

void main(void)
{
      char destinazione[1000];
      char *sorgente= "Heisemberg";

      strcpy(destinazione, sorgente);

      printf("%s\n",sorgente);
      printf("%s\n",destinazione);
}
```

/* vet.c - ESEMPIO di uso vettori e funzioni random:
somma vettoriale di due vettori generati casualmente */

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

void alloca_vettore_double(double* *ppvet, unsigned int n);
double frand(double fmax );
void init_vect_random(double *v, unsigned int n);
void somma_vet( unsigned int n , double *v1, double *v2, double *v3);
void stampa_vet( unsigned int n , double *v);

void main(void)
{
	unsigned int n;
	double *v1, *v2, *v3,  seme;

	printf("dimensione vettori ? ");
	scanf("%d",&n);
	alloca_double(&v1,n);
	alloca_double(&v2,n);
	alloca_double(&v3,n);
	seme=1.0;
	srand(seme);
	init_vect_random(v1,n);
	stampa_vet(n,v1);
	init_vect_random(v2,n);
	stampa_vet(n,v2);
	somma_vet(n,v1,v2,v3);
	stampa_vet(n,v3);
}
```

```
void alloca_double(double* *ppvet, unsigned int n)
{
	*ppvet = (double*) calloc(n,sizeof(double));
	if( !(*ppvet) )  {
		fprintf(stderr,"calloc error\n");
		exit(0);
	}
}

double frand(double fmax )
{
return( fmax * ((double) rand()) / ( (double)RAND_MAX ) );
}

void init_vect_random(double *v, unsigned int n)
{
	unsigned int i;
	for(i=0;i<n;i++)
		v[i]=frand(1000000.0);
}

void somma_vet( unsigned int n , double *v1, double *v2, double *v3)
{
	unsigned int i;
	for(i=0;i<n;i++)
		v3[i]=v1[i]+v2[i];
}

void stampa_vet( unsigned int n , double *v)
{
	unsigned int i;
	for(i=0;i<n;i++)
		printf(" %10.3lf", v[i]);
	printf("\n");
}
```

```
/* tree.C - costruzione e visita di albero binario a partire da elementi /
#include <stdio.h>
#include <stdlib.h>

typedef struct s_NODO{
   double  x;
   s_NODO     *destra;
   s_NODO     *sinistra;
   }  NODO;

#define FMAX 10000.0        /* max valore generato da frand */

double frand ( double fmax );
void init_root (NODO* *ppnodo );
void visita_anticipata_albero ( NODO *pnodo );
void dealloca_albero ( NODO* *ppnodo );
int insert( double x , NODO* *ppnodo );

void main(void)
{
      unsigned int n, i, seme;         double x , seme;
      NODO *root;                      int ris;

      seme=1;        srand(seme);        /* inizializza. stessa seq. casuale */
      /* srand(  time(NULL) );             seq. casuale sempre diversa    */
      init_root(&root);
      printf("numero nodi ? ");
      scanf("%d",&n);
      for(i=0;i<n;i++)     {
            x = frand(FMAX);
            ris = insert( x , &root );
            if(ris==0)       printf("elemento %f gia' presente\n",x);
      }
      visita_anticipata_albero(root);
      dealloca_albero(&root);
}
```

```
void init_root(NODO* *ppnodo)
{
	*ppnodo=NULL;
}

void visita_anticipata_albero(NODO *pnodo)
{
	if(pnodo)  {
		visita_anticipata_albero(pnodo->sinistra);
		printf("%lf ",pnodo->x);
		visita_anticipata_albero(pnodo->destra);
	}
}

void dealloca_albero(NODO* *ppnodo)
{
	if(*ppnodo)
	{
		dealloca_albero( &((*ppnodo)->sinistra) );
		dealloca_albero( &((*ppnodo)->destra) );
		free(*ppnodo);
		*ppnodo=NULL;
	}
}

double frand(double fmax )
{
	return( fmax * ((double) rand()) / ( (double)RAND_MAX ) );
}
```

```
int insert( double x , NODO* *ppnodo )
{
	if(  ! (*ppnodo)  )
	{
		*ppnodo = (NODO*) calloc ( 1 , sizeof(NODO) );
		if( ! (*ppnodo) )
			{   fprintf(stderr,"calloc failure\n"); exit(0); }
		(*ppnodo)->x=x;
		(*ppnodo)->destra   = NULL;
		(*ppnodo)->sinistra = NULL;
	}
	else
	{
		if (x == (*ppnodo)->x )
			return(0); /* elemento gia' presente */
		else
		{
			if ( x < (*ppnodo)->x )
				return( insert( x , & ((*ppnodo)->sinistra) )  );
			else
				return( insert( x , & ((*ppnodo)->destra) )  );
		}
	}
}
```