Laboratorio di Algoritmi e Strutture Dati

Funzioni standard del C:

- Funzioni di I/O
- Allocazione dinamica della memoria
- Funzioni su stringhe di caratteri

Testo di riferimento

B.W. Kernighan & D.M. Ritchie “**The C Programming Language**”, Prentice Hall 1988 (2nd Ed.).

oppure, in versione italiana:

B.W. Kernighan & D.M. Ritchie “**Il linguaggio C: principi di programmazione e manuale di riferimento**”, Pearson 2004.

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  int x = 14;
  float y = 3.1415;


  printf("x vale %d", x);  /* stampa il valore intero della variabile x */
  printf("y vale %f \n", y);   /* stampa il valore floating-point di y + newline */
}
```

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x = ‘a’;
  char y[5] = "abcd";

  printf("%c\n", x);  /* stampa il carattere contenuto nella variabile x */
  printf("%s\n", y);   /* stampa la stringa contenuta nell’array y */
}
```

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x = ‘a’;
  char y[5] = "abcd";

  printf("%c\n", x);  /* stampa il carattere contenuto nella variabile x */
  printf("%s\n", y);   /* stampa la stringa contenuta nell’array y */
}
```

Perché array **y** è dichiarato di lunghezza 5 ma contiene solo 4 caratteri?

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x = ‘a’;
  char y[5] = "abcd\0";

  printf("%c\n", x);  /* stampa il carattere contenuto nella variabile x */
  printf("%s\n", y);   /* stampa la stringa contenuta nell’array y */
}
```

L’ultimo elemento dell’array deve contenere il carattere di terminazione stringa ‘\0’.
Le funzioni standard del C assumono che le stringhe siano **sempre terminate**.

Il tipo di dati stringa di caratteri

Le stringhe sono array di elementi di tipo `char`, il cui ultimo elemento è 0 o, equivalentemente, il carattere di terminazione di stringa '\0'.

```
char s[] = "pippo";
```

è equivalente a

```
char s[6] = "pippo";
```

è equivalente a

```
char s[] = {'p','i','p','p','o','\0'};
```

è equivalente a (NB: non del tutto equivalente, vedremo in che senso più avanti):

```
char *s = "pippo";
```

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x = ‘a’;
  char y[5] = "abcd";

  printf("Carattere %c e stringa %s \n", x, y);
      /* stampa il carattere contenuto nella variabile x e, di seguito, la stringa
          contenuta in y */
}
```

La stringa di formato

Alcuni dei più comuni codici per la stringa di formato:

%d Stampa l'argomento corrispondente come un numero decimale con segno, e.g., 3490. L'argomento deve essere un int.

%f Stampa l'argomento corrispondente come un numero floating-point, e.g., 3.14159. L'argomento deve essere un float.

%c Stampa l'argomento corrispondente come un carattere, e.g., 'B'. L'argomento deve essere un char.

%s Stampa l'argomento corrispondente come una stringa, e.g., "Salve mondo?". L'argomento deve essere un char* o un char[].

%% Nessun argomento viene convertito, viene stampato un semplice segno di percento. È il modo di stampare '%' con **printf()**.

%p Stampa un tipo puntatore in formato esadecimale, i.e., l'indirizzo di memoria a cui il puntatore punta. (**Non** il valore memorizzato nell'indirizzo, ma il numero corrispondente alla cella di memoria.)

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  FILE *fd = fopen("myfile.txt", "w");
  char y[5] = "abcd";
  if (fd) {
     fprintf(fd,"%s\n", y);   /* inserisce nel file la stringa contenuta in y */
     fclose(fd);
   }
}
```

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream,
```

Variabile di tipo puntatore a (descrittore di) file.

```
#include <stdio.h>

int main(void) {
  FILE *fd = fopen("myfile.txt", "w");
  char y[5] = "abcd";
  if (fd) {
     fprintf(fd,"%s\n", y);   /* inserisce nel file la stringa contenuta in y */
     fclose(fd);
   }
}
```

Funzioni standard di output

```
#include <stdio.h>

int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  FILE *fd = fopen("myfile.txt", "w");
  char y[5] = "abcd";
  if (fd) {
     fprintf(fd,"%s\n", y);   /* inserisce nel file la stringa contenuta in y */
     fclose(fd);
   }
}
```

Funzioni di apertura e chiusura di file.

Funzioni standard di output

```
#include <stdio.h>

FILE *fopen(const char *name, char *mode);
int fclose(FILE *stream);
```

```
#include <stdio.h>

int main(void) {
  FILE *fd = fopen("myfile.txt","w");
  char y[5] = "abcd";
  if (fd) {
     fprintf(fd,"%s\n", y);   /* inserisce nel file la stringa contenuta in y */
     fclose(fd);
   }
}
```

Modalità di apertura file

I possibili valori per il **mode** in fopen("myfile.txt", **mode**) sono:

r	Apre il file per la lettura (read-only).
w	Apre il file per la scrittura (write-only). Il file viene creato se non esiste.
r+	Apre il file per lettura e scrittura. Il file deve già esistere.
w+	Apre il file per scrittura e lettura. Il file viene creato se non esiste.
a	Apre il file per aggiornamento (append). È come aprire il file per la scrittura, ma ci si posiziona alla fine del file per aggiungere dati alla fine. Il file viene creato se non esiste.
a+	O Apre il file per lettura e aggiornamento. Il file viene creato se non esiste.

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

Indirizzo della variabile intera x.

```
#include <stdio.h>

int main(void) {
  char x;
  printf("Inserire un carattere:");
  scanf("%c", &x);
   /* riceve un carattere da tastiera e lo inserisce nella variabile x */
}
```

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

ATTENZIONE!!!

```
#include <stdio.h>

int main(void) {
  char x;


  scanf("Inserire un carattere: %c", &x);
      /* Legge da tastiera la stringa "Inserire un carattere:"  seguita da un ultimo
          carattere che viene inserito nella variabile x */
}
```

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x[10];


  printf("Inserire una stringa di 9 caratteri:");
  scanf("%s", x); /* Legge da tastiera la stringa che viene inserita nell'array x */
}
```

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x[10];

  printf("Inserire una stringa di 9 caratteri:");
  scanf("%s", x); /* Legge da tastiera la stringa che viene inserita nell'array x */
}
```

Cosa succede se si inseriscono più di 10 caratteri?

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x[10];

  printf("Inserire una stringa di 9 caratteri:");
  scanf("%s", x); /* Legge da tastiera la stringa che viene inserita nell'array x */
}
```

Errore di indirizzamento (**segmentation fault**)!
Non vi è alcun controllo sui limiti degli array!

Funzioni standard di input

```
#include <stdio.h>

int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

```
#include <stdio.h>

int main(void) {
  char x[10];

  printf("Inserire una stringa di 9 caratteri:");
  scanf("%9c", x);
    /* Legge una stringa i cui primi 9 caratteri vengono inseriti nell'array x */
}
```

Una possibile soluzione

Funzioni standard di input

```
#include <stdio.h>
int getc(FILE *stream);
int fgetc(FILE *stream);
int getchar(void);
```

Una soluzione più robusta.

```
#include <stdio.h>
void my_get_str(char *str, int len) {
 int i;
 for(i=0; (i<len && (c = getchar()) !='\n' && c != EOF; i++)   /* legge max  len caratteri */
        str[i]=c;
  str[i] = '\0';  /* inserimento carattere di terminazione stringa */
}

int main(void) {
  char x[10];
  printf("Inserire una stringa di massimo 9 caratteri:");
  my_get_str(x,9);
}
```

Funzioni standard di input

```
#include <stdio.h>
int getc(FILE *stream);
int fgetc(FILE *stream);
```

```
#include <stdio.h>
void my_fget_str(FILE *fd, char *str, int len) {
 int i;
 for(i=0; (i<len && (c = fgetc(fd)) !='\n' && c != EOF; i++) /*legge max  len caratteri in str*/
        str[i]=c;
  str[i] = '\0';  /* inserimento carattere di terminazione stringa */
}
int main(void) {
  char x[10], c;
  FILE *fd = fopen("myfile.txt", "r");
  if (fd) {
     my_fget_str(fd,x,9);
     fclose(fd);
   } }
```

Strutture

Per aggregare variabili di tipo diverso sotto un unico nome il C fornisce le **strutture**.

```
struct punto {
    int x;
    int y;
};
```

```
struct rettangolo_colorato_pesato {
    int colore;
    double peso;
     struct punto bottomleft, topright;
 };
```

```
struct punto sommapunti(struct punto a, struct punto b)
{
  a.x += b.x;
  a.y += b.y;
  return a;
}
```

Strutture

Per aggregare variabili di tipo diverso sotto un unico nome il C fornisce le **strutture**.

```
struct punto {
    int x;
    int y;
};
```

```
struct rettangolo_colorato_pesato {
    int colore;
    double peso;
     struct punto bottomleft, topright;
```

Operatore "." (punto) per accesso a campo di struct.

```
struct punto sommapunti(struct punto a, struct punto b)
{
  a.x += b.x;
  a.y += b.y;
  return a;
}
```

Allocazione dinamica della memoria

```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int dimelem);
void free(void *ptr);
```

```
…
int *interi;
char *string;


interi = (int *) malloc(n * sizeof(int)); /* allocazione di spazio per n interi */
stringa = (char *) calloc(n, sizeof(char)); /* allocazione di n caratteri */
…
free(interi);
free(stringa);
```

Allocazione dinamica della memoria

```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int d
void free(void *ptr);
```

Tipo puntatore a (indirizzo di) dato intero.

Tipo puntatore a (indirizzo di) dato carattere.

```
...
int *interi;
char *string;


interi = (int *) malloc(n * sizeof(int)); /* allocazione di spazio per n interi */
stringa = (char *) malloc(n * sizeof(char)); /* allocazione di n caratteri */
...
free(interi);
free(stringa);
```

Allocazione dinamica della memoria

```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int dimelem);
void free(void *ptr);
```

```
…
int *interi;
char *string;


interi = (int *) malloc(n * sizeof(int));          [ .. = malloc(n * sizeof (*interi)]
stringa = (char *) malloc(n * sizeof(char));   [.. = malloc(n * sizeof (*stringa)]
…
free(interi);
free(stringa);
```

Operazione di ***casting*** per convertire a un altro tipo.

Allocazione dinamica della memoria

```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int dimelem);
void free(void *ptr);
```

Operatore: restituisce l'occupazione in byte del tipo o variabile specificati.

```
  …
  int *interi;
  char *string;

  interi = (int *) malloc(n * sizeof(int));          [ .. = malloc(n * sizeof (*interi)]
  stringa = (char *) malloc(n * sizeof(char));   [.. = malloc(n * sizeof (*stringa)]
…
  free(interi);
  free(stringa);
```

Allocazione dinamica della memoria

```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int dimelem);
void free(void *ptr);
```

```
…
int *interi;
char *stringa;
int k=8, i=5;

interi = (int *) malloc(n * sizeof(int));
stringa = (char *) calloc(n, sizeof(char));
*(interi+i) = k;    /* assegna valore di k al (i+1)-esimo intero a patire
                       dall'indirizzo interi */
stringa[i] = 'b';   /* assegna carattere 'b' al (i+1)-esimo carattere a patire
                       dall'indirizzo stringa */
…
```

Allocazione dinamica della memoria

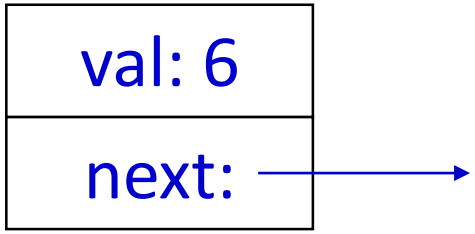
```
#include <stdlib.h>

void *malloc(int dim);
void *calloc(int nelem, int dimelem);
void free(void *ptr);
```

```
int *interi;
int i;

interi = (int *) malloc(n * sizeof(int));
for (i=0; int < n; i++)
     *(interi+i) = i;     /* interi[i] = i; */
printf("La sequenza di interi è: ");
for (i=0; int < n; i++)
     printf("%d ",*(interi+i));   /* printf("%d ", interi[i]); */
 printf("\n");
```

Allocazione dinamica di strutture

Per aggregare variabili di tipo diverso sotto un unico nome il C fornisce le **strutture**.

```
struct lista {
    int val;
    struct lista *next;
};
```

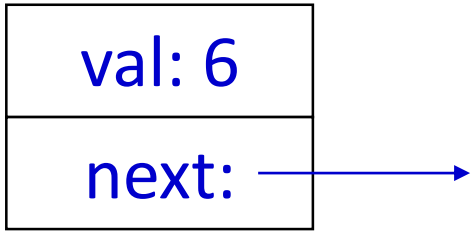



```
struct lista *mylist, *nodo;
…
nodo = (struct lista *) malloc(sizeof(struct lista));
nodo->val = 6;
nodo->next = NULL;
mylist = nodo;
…
```

Allocazione dinamica di strutture

Per aggregare variabili di tipo diverso sotto un unico nome il C fornisce le **strutture**.

```
struct lista {
    int val;
    struct lista *next;
};
```



Operatore "->" (freccia) per accesso a campo di struct indirizzata tramite puntatore.

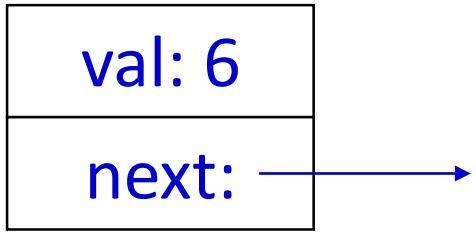
```
struct lista *mylist, *nodo;
...
nodo = (struct lista *) malloc(sizeof(struct lista));
nodo->val = 6;          /* equivalete a: *(nodo).val = 6; */
nodo->next = NULL;
mylist = nodo;
...
```

typedef: tipi definiti dall'utente

La parola chiave **typedef** permette di un ***alias*** per tipi definiti dall'utente tramite costrutti di composizione di tipi.

typedef type alias;

```
struct lista {
    int val;
    struct lista *next;
};
```



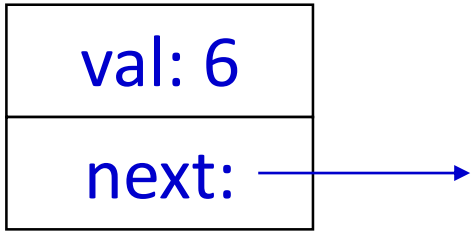
typedef struct lista LISTA;

```
typedef struct lista LISTA;

LISTA *mylist;
 …
 nodo = (LISTA *) malloc(sizeof(LISTA));
 nodo->val = 6;
 nodo->next = NULL;
 …
```

typedef: tipi definiti dall'utente

La parola chiave **typedef** permette di creare un ***alias*** per tipi definiti dall'utente tramite costrutti di composizione di tipi.

```
typedef struct lista LISTA;

struct LISTA {
    int val;
    LISTA *next;
};
```



La libreria standard string.h

```
#include <string.h>

char *strcpy(char* dst, char* src);
int strlen(char* str);
int strcmp(char* str1, char* str2);
```

```
...
char *stringa;

stringa = (char *) malloc(n * sizeof(char));
*(stringa+i) = ‘k’;
 stringa[i+1] = ‘b’;
 strcpy(stringa,"hello");
 printf("la stringa è: %s\n", stringa);
```

strcpy: possibile implementazione

```
char *strcpy(char *dst, char *src)
{
   int i=0;
   do {
          dst[i] = src[i];
    } while (src[i++] != '\0')
   return (dst);
 }
```

strcpy: possibile implementazione II

```
char *strcpy(char *dst, char *src)
{
   char *ret=dst;
   while ( (*(dst++) = *(src++)) !=  '\0')
           ;
   return (ret);
 }
```

La libreria standard string.h

```
#include <string.h>

char *strcpy(char* dst, char* src);
int strlen(char* str);
int strcmp(char* str1, char* str2);
```

La funzione strlen restituisce la lunghezza della stringa, cioè il numero di caratteri fino al simbolo di terminazione \0 (escluso).
Il valore restituito può essere diverso da *n* (numero di byte allocati per stringa).

```
int val;
char *stringa;

stringa = (char *) malloc(n * sizeof(char));
strcpy(stringa,"hallo");
val = strlen(stringa);
printf("lunghezza %d\n", val);
```

strlen: possibile implementazione

```
int strlen(char *str)
{
   int len=0;
   while (str[len] != '\0')
            len++;
    return (len);
 }
```

La libreria standard string.h

```
#include <string.h>

char *strcpy(char* dst, char* src);
int strlen(char* str);
int strcmp(char* str1, char* str2);
```

```
int val;
char *str1;
char *str2;

str1 = (char *) malloc(n * sizeof(char));
strcpy(str1,"hello");
str2 = (char *) calloc(n, sizeof(char));
strcpy(str2,"hallo");
val = strcmp(str1,str2);
printf("lunghezza %d\n", val);
```

La funzione strcmp restituisce un valore:

- < 0 se str1 precede lessicograficamente str2
- > 0 se str1 segue lessicograficamente str2
- = 0 se str1 identica a str2

strcmp: possibile implementazione

```
int strcmp(char *str1, char *str2)
 {
   while ( *(str1) && *(str2) && *(str1++) == *(str2++) )
         ;
    return ( *(--str1) - *(--str2) );
  }
```

Generazione di numeri casuali

```
#include <stdlib.h> /* rand() e srand() */
#include <time.h>   /* time() */

void srand(unsigned int seed);
int rand();
time_t time(time_t *timer);
```

```
int main() {
   int val;

   srand(time(NULL));
   val = rand();
   printf("Numero casuale: %d",val);
   return(0);
}
```

Generazione di numeri casuali: esempio

```
#include <stdio.h> /* printf() */
#include <stdlib.h> /* rand() e srand() */
#include <time.h>   /* time() */

int random_num(int a, int b)
{    /* genera un numero casuale tre a e b */
  int ret = a+(rand()%(b-a+1));
  return (ret);
}

int main() {
   int val;

   srand(time(NULL)); /* inizializza il generatore casuale */
   val = random_num(10,20);
   printf("Numero casuale: %d",val);
   return(0);
}
```