5 Linguaggio C89 ANSI

Lo standard ANSI del linguaggio di programmazione C è stato progettato per promuovere la la portabilità dei programmi C fra una varietà di sistemi informatici. Per realizzare questo, la norma copre tre aree principali: l'ambiente in cui il programma compila ed esegue, la semantica e la sintassi del linguaggio e il contenuto e la semantica di un insieme di routine di libreria e file di intestazione.

5.1 Sintassi linguaggio

5.1.1 Operatori

- Assegnamento "=" che corrisponde ad dire *var* := *qualcosa*, ovvero definisce a cosa corrisponde il valore della variabile con quello che sta a destra dell'uguale, per questo l'assegnamento è unidirezionale, infatti $(A = B) \neq (B = A)$
- Operatori aritmetici (bisogna stare attenti al tipo della variabile perchè in base a quello si svolgono operazioni differenti):
  - Somma "+":
  - Differenza "−":
  - Prodotto "∗":
  - Divisione "/":
  - Resto divisione "%":
- Operatori logici, si seguono le tabelle di verità dei diversi operatori:
  - Prodotto logico (AND) "&&"
  - Somma logica (OR) "||"
  - Negazione (NOT) "!"
- Operatori relazionali:
  - Maggiore stretto ">" // Maggiore o uguale "≥"
  - Minore stretto ">" // Minore o uguale "≤"
  - Confronto "==" per ditinguerlo da assegnamento ("=0")
  - Diverso "! =" (non uguale)

Ripasso condizioni logiche nel while e if (legge di De Morgan):

- (condiz_1 || condiz_2) == !(!condiz_1 && !condiz_2)
- (condiz_1 && condiz_2) == !(!condiz_1 || !condiz_2)

5.2 Compilare ed eseguire

L'estensione del file è fondamentale perchè comunica al compilatore il probabile contenuto del file, ad asempio se salviamo un file come nomefile.c permettiamo all'editor di riconoscere il linguaggio di programmazione C ed evidenziare le keyword.
In qesto corso per gli studenti che utilizzano un compilatore windows sarà necessario installare una virtual machine di linux (ad esempio Ubunto for windows) che permette di eseguire i programmi da riga di comando di linux.

Per compilare un file si scrive nel prompt dei comandi di ubuntu:
gcc -Wall -std=c89 -pedantic -o nomefileeseguibile nomefilesorgente Analizziamo questo comando (l'ordine dei parametri non è importante e si possono scambiare fra loro):

- -Wall segnala tutti i warning
- -std=c89 controlla che il codice rispetti lo standard ANSI c89 del linguaggio C
- -pedantic segnala le violazioni segnalate
- -o nomefileeseguibile è il file eseguibile (in windows ha estensione file.exe e su linux non ha estensione)

- -nomefilesorgente è il file sogente .c

Si possono presetare due problemi:

- Error: non si crea l'eseguibile, abbiamo sbagliato sintassi (indicano anche riga codice) se si dimentica ; errore è nella riga dopo
- Warning: condizioni strane che potrebbero indicate la presenza di un problema, ma viene creato comunque un eseguibile possibile eerore di semantica (non ultima versione gcc)

Per l'esecuzione è necessario specificare al compilatore dove si trova il file eseguibile (directory del file).
Esistono dei tool utilissimi per i neoprogrammatori (sudo apt install nomeprogramma) che sono:

- Indent: formatta e indenta corretamente file.c
- Diff: confronta due file evidenziando i caratteri differenti

5.2.1 Spiegazione sintassi codice

```c
/* (1) inclusione librerie */
#include <stdio.h>

# define TOM 99.99

/* (2) int main */
int main(int argc, char*argv[])
{
    /* (3) indentazione*/
    /* (4) dichiarazione variabili */
    int intero1, intero2 , intero3;
    float decimale, tot;
    char carattere;

    /* (5) input */
    scanf("inserisci numero: %d \t %d", &intero1, &intero2);
    scanf("inserisci numero:\n %f", &decimale);

    /* (6) assegnamento (elaborazione) */
    tot = intero1 + intero2 + decimale;

    /* (7) output */
    printf("il totale è:\t", tot);

    return 0;
}
```

Spiegazione delle parti del programma:

1. Librerie: la libreria standard del C è un insieme di librerie che forniscono al programmatore funzioni tipizzate. Insieme alla libreria vengono inoltre forniti gli Header file, file di testo che permettono al programmatore di utilizzare lo specifico insieme di funzioni della libreria ad esse associate.
2. Int main: la funzione main è il punto di inizio per l'esecuzione di un programma. Essa è generalmente la prima funzione eseguita durante l'avvio di un programma.
3. indentazione: è l'inserimento di una certa quantità di spazio vuoto all'inizio di una riga di testo (tab = 4 spazi), è una convenzione utilizzata per esprimere al meglio la struttura di un codice sorgente. In alcuni linguaggi come C e C++ è una convenzione, mentre in altri come Python fa parte della sintassi.
4. Dichiarazione variabili: è importante per stabilire quanto spazio deve riservare il calcolatore per un certo dato e per sapere la tipologia di dato che deve essere pronto a ricevere.
5. Input: la sintassi è 'scanf("%tipo_variabile", &nome_variabile);', si possono leggere anche più di una variabile alla volta. I tipi della variabile possono essere molteplici e vengono spiegati nella foto 16.

6. Assegnamento ed elaborazione: vongono svolte operazioni di assegnamento, confronto, ecc usando gli operatori consentiti.

7. Output: la sintassi è 'printf("testo in output: %tipo_dato", nome_variabile);' e possiamo usare i caratteri ASCII (13) non stampabili nell'immagine 17.

TIPO	N. BIT	VALORI RAPPRESENTABILI
char	8	Da -128 a +127 (o caratteri ASCII corrispondenti)
unsigned char	8	Da 0 a 255 (o caratteri ASCII corrispondenti)
signed char	8	Gli stessi del tipo base "char"
int	16	Da -32768 a +32767
unsigned int	16	Da 0 a 65535
signed int	16	Gli stessi del tipo base "int"
short int	16	Gli stessi del tipo base "int"
unsigned short int	16	Gli stessi del tipo "unsigned int"
signed short int	16	Gli stessi del tipo base "int"
long int	32	Da -2147483648 a +2147483647
signed long int	32	Gli stessi del tipo "long int"
unsigned long int	32	Da 0 a 4294967295
float	32	Circa da $-3{,}4\times10^{38}$ a $+3{,}4\times10^{38}$
double	64	Circa da $-1{,}8\times10^{308}$ a $+1{,}8\times10^{308}$
long double	80	Circa da $-1{,}2\times10^{4932}$ a $+1{,}2\times10^{4932}$

Figure 15: Tipi delle variabili in C

%d	per il tipo int
%ld	per il tipo long int
%f	per il tipo float
%lf	per il tipo double
%c	per il tipo char
%s	per stringhe di caratteri

Figure 16: Tipi di dato in C

Carattere speciale	Descrizione
\n	Newline o anche ritorno a capo
\r	Carriage return o ritorno a inizio rigo
\t	Tab
\v	Vertical tab
\b	Backspace
\f	Form feed
\a	Alert o beep
\'	Singolo apice
\"	Singole virgolette
\?	Punto interrogativo
\\	Backslash

Figure 17: Caratteri speciali in C

5.3 Funzioni

5.3.1 Sottoprogrammi

Sottoprogrammi sono composti da funzioni e sottoprocedure (termine piu generale).
Ruolo dei sottoprogrammi è spostare parte dell'elaborazione in dei sottoprogrammi scomponendo un problema in tanti piccoli task svolti ciscuno da un singolo sottoprogramma.
Ad esempio funzione che dice se numero è primo o meno, retituisce '1' se primo '0' se non primo. Ad esempio anche 'scanf()', 'print()' e 'gets()' sono sottoprogrammi.

Sottoprogramma non viasualizza, calcola ed elabora sulle variabili.
Sottoprogramma non chiede direttamente all'untente, riceve informazioni dal main (A meno che non si acquisice valori matrice).
Come blackbox (immagine gg) che può retituire informazione come risultato dell'elaborazione (massimo 1) o può solo gestire informazioni senza return (minimo 0). Se produce più di un informazione, gestire un input in n parti diverse per non ripetere ciclo n volte (prossima lezione).

Sintassi funzione base:

```c
/* Sintassi funzione base */
tipo_funzione nome_funzione (argomenti scambiati){
        codice sottoprogramma 1;
        codice sottoprogramma 2;
        codice sottoprogramma 3;
        return dato_restituito;
}
```

Tipi di funzione in base a tipo di dato retituito (anche nuovo tipo con 'typedef struct'), possono essere:

- int
- float
- char
- void (se non entra nulla in input) es: $'void nome_funzione(){elaborazione}'$ o anche $'nome_funzione(){elaborazione}'$

Esempio di funzione 'void' per input valore:

```c
/* Sintassi funzione void */
void menu(){
        printf("----------------------\n");
        printf("inserisci valore\n");
        printf("0, termina\n\n");
        return; /* inutile perche funzione void */
}

int scelta_menu(){
        int sell;
        do{
                /* visualizza menù */
                menu();
                scanf("%d",&sell);
        }while(sell<0 || sell>4);         /* o anche (!(sell>=0 && sell<=4)) */
        return sell;
}

int primo(int val){ /* '1' primo - '0' non primo */
        int i, ris, meta;
```

```
        if(val>0){                /* controllo positivita numero */
                ris = 1;
                if(val%2 == 0){
                        ris = 0;
                }
                else{
                        meta = val/2;                /* tanto 20 non è divisibile per 'i>10' sicuro */
                        for(i=3; i<meta && ris != 0; i += 2){
                                if(val%i == 0)
                                        ris = 0;
                        }
                }
        }
        else
                ris = -1;
        return ris;
}
```

return torna alla funzione main e interrompendo flusso codice della funzione void, non metterne due di return. Al limite mettere 'if(cond) return 0; return1;'
Convenzione disposizione delle parti del codice:

```
/* Distribuzione delle parti di programma */
#include <librerie>
#define DEFINE
/* prototipo */
void mem()
int scelta_menu();
int primo();
int main(int argc; cher*argv[]){

        /* programma del main */

}
```

Passaggio dei parametri con i sottoprogrammi, riassunto generale con parametri necessari:

```
int lunghezza(char[]);
int index_max(int[],int);
int visualizz_matt(int[][NCOL], int , int);
```

È importante sottolineare che le varibili in C hanno uno scope, ovvero sono definite solo all'interno della funzione in cui sono state dichiarate, le varibili dichiarate nella funzione main sono dette variabili globali.

Esempi:

- Funzione che riceve in ingresso una stringa e retituisce lunghezza della stringa (dimesione stringa):

```
/*  Funzione che riceve in ingresso una stringa e retituisce lunghezza della stringa (dimesione stringa) */
int len_stringa(char s[]){
        int dim;
        for(dim=0; s[dim]!='\0'; dim++);
        return dim;
}
```

- Funzione che riceve in ingresso un array e retituisce indice dell'elelmento di valore massimo (si può usare anche 'strlen()' includendo la libreria 'string.h':):

```c
/* retituisce indice dell'elemento dell'array di valore massimo (se 2 pari tengo il più basso di indice) */
int index_max(int v[], int dim){
        int i, i_max;
        i_max = 0;
        for(i=0; i<dim; i++)
                if(v[i]>v[i_max])
                        i_max = i;
        return i_max;
}
```

- Array bidimensionale di valori interi e paramentro strettamente necessrio visuslizza il contenuto:

```c
/* array bidimensionale di valori interi e paramentro
strettamente necessrio visuslizza il contenuto */
void visualizz_mat(int m[][nc]; int nr, int nc) /* [ necessario specificzre tutte le dimensioni dello spazio per linerizzazzi
        int, i j;
        for(i=0; i<nr; i++){
                for(j=0; j<nc; j++)
                        printf("%d";m[i][j]);
        printf(/n);
        }
}
```

- Ingresso stringa e carattere e restituisce ultima posizione in cui quel carattere compare nella string ('-' se non compare):

```c
/* ingresso stringa  e carattere  e restituisce ultima posizioni in cui
quel carattere compare nella string ('-1' se non compare) */
int compare(char stringa[], char caratt){
        int i, pos;
        pos = -1;
        for(i=0; stringa[i]!='\0'; i++){
                if(stringa[i]==caratt)
                        pos=i;
        }
        return pos;
}
```

- Ingresso stringa e carattere e restituisce prima posizione in cui quel carattere compare nella string ('-' se non compare):

```c
/* ingresso stringa  e carattere  e restituisce ultima posizioni in cui
quel carattere compare nella string ('-1' se non compare) */
int compare(char stringa[], char caratt){
        int i, pos;
        pos = -1;
        for(i=0; stringa[i]!='\0' && pos!=-1; i++){
                if(stringa[i]==caratt)
                        pos=i;
        }
        return pos;
}
```

- Esercizio completo 1:

```c
/* trovare carattere in una stringa tramite funzione */
#define LMAX 35
#define TR 'trovato'
#define NT 'non trovato'
int cerca(char[],char)
int main(int argc, char*argv[]){
        char voc[LMAX+1], sel;
        int ris;
        scanf("%s %c", voc, &sel);
        ris = cerca(voc, sel);
        if(ris!=-1)
                printf("%s\n",TR);
        else
                printf("%s\n",NTR);
        return 0;
}
```

- Esercizio completo 2:

```c
/* chiede all'untente numero di valori da getstire e avvalemndosi del sottoprogramma max_array visualizza il massimo */
#include <stdio.h>
#define MAX 50
max_array(int [], int);
int main(int argc, char*argv[}){
        int valori[50], n_val, i;
        do
                scanf("%d", &n_val);
        while(!(n_val>=1 && val<=50));
        for(i=0;i<n_val; i++)
                scanf("%d", &valori[i]);
        pos = max_array(valori, n_val);
        max = valori[pos];
        printf("%d", max);
        return 0;
}
```

5.3.2 Puntatori

Con la sintassi '&nome-variabile' si ottiene la locazione nella memoria nel quale è salvato quel tale valore. Il programma passa, ad esempio, valori array[], locazione memoria del primo output e locazione di memoria del secondo output. La funzione associerà a quel indirizzo di locazione nella memoria il valore risultato dell'elaborazione. tipi di indirizzi:

- 'int* nome-varibile' è l'indirizzo di una generica varibile intera.
- 'float* nome-varibile' è l'indirizzo di una generica varibile reale.
- 'char* nome-varibile' è l'indirizzo di una generica varibile carattere.

Esempio sintassi di funzioni con output mandato tramite puntatore:

```c
/* sottoprogramma che riceve n dati in ingresso e trasmette (non restituisce con return) m>=2 caratteri */
/* ricevuto in ingresso una stringa, trasmette al chimante numero vocali e nuemero consonanti  */
void conta_voc_cons(char s[], int* n_vocali, int* n_consonanti){
        int i;
        int n_voc, n_cons;
        n_voc = 0;
        n_cons = 0;
```

```
        for(i=0; i!='\0'; i++){
                if(s[i]=='e'|| s[i]=='e'|| s[i]=='e'|| s[i]=='e'|| s[i]=='e')
                        n_voc++;
                else
                        n_cons++;
        }
        *n_vocali = n_voc;
        *n_consonanti = n_cons;
}
/* per chiamare questa funzione: */
char voc[LMAX+1];
int num_vocali, num_consonanti;
gets(voc);
conta_voc_cons(voc, &num_vocali, &num_consonanti)
printf("%d\n%d\n",num_vocali, num_consonanti);
```

Per non fare confusione fra l'utilizzo di '*' e '&':

- per trasmettere l'indirizzo di una variabile '*' (quando scrivo la funzione)
- Per chiamare la funzione di utilizza '&'
- asterisco '*': funzione che restituisce il valore della varibile in un indirizzo.
- e commerciale '&': funzione che retituisce indirizzo in cui è allocata una variabile.

Record di attivazione:
Viene creato uno stack (pila) che contine la memoria generata dalla funzione. Ci sono alcune celle che sono dedicate alla variabili passate alla funzione che vengono salvate in locale, ma è molto importnate sapere l'indirizzo di ritorno. Ad esempio quando viene eseguita funzione main c'è solo quelli, chiamata la funzione scanf si crea il record di attivazione del sottoprogramma che poi viene tolta andando all'indirizzo di ritorno.

Passaggio di struct nelle funzioni:

```
/* Distribuzione del codice */
#include
#define
typedef                 ;
/* prototipi        ;        */
int main(){
        /* codice del main */
}
int funzione(){
        /* istruzione della funzione */
}
```

Sintassi del passaggio di una struttura dati di tipo 'struct':

```
/* Sintassi base del passaggio per indirizzo di una struct */
typedef struct data_s{
        int giorno, max;
        int anno;
}data_t;

data_t a, b;

```

```
ris = f1(a,b);
ris = f2(&a,&b);
```

Esercizio con passaggio di struttre a funzioni per indirizzo:

```
/* sottoprogramma che ricevuti in ingresso due date 'da' e 'a' restituisce il numero di giorni tra le due date */
#include <stdio.h>
typedef struct data_s{
        int giorno, max;
        int anno;
}data_t;
int diff_date(data_t *da, data_t *a){
        int diff, m;
        if(*da.anno == *a.anno)                /* *a.anno corripsonde a->anno */
                if(*da.mese == *a.mese)
                        diff = *a.giorno - *da.mese;


        return differenza;
}
int giornidelmese(int mese; int anno){
}

```

Una notazione alternativa per scrivere i puntatori di strutture dati complesse (tipo 'struct') possimao scrivere al posto di '*a.anno' come 'a– anno'.

Metodi alternativi per il passaggio di parametri a funsioni:

- Array:
  function(int v[ ], int dim)
  function(int *v, int dim)
- Matrice: function(int m[ ][NCOL], int nr, int nc)
  function(int *m, int nr, int nc, int ncdmax) // poi gestitre tutto con puntatori per centrare la cella giusta
- Struttura:
  function(mio-tipo-t *s)

5.3.3 Acquisizione da riga di comando

Serve per creare terminali di lavoro con poca iterazione necessaria.
Per compilare e creare l'eseguibile si scrive 'gcc -o nome-programma nome-programma.c'. 'argv[ ]' è un vettore di stringhe (si vede da 'char *argv[ ]') composto da '/.nome-programma ciao 55 volte'.
Essendo un array non si ha i terminatore '\0', quindi si deve fornire anche il numero di elementi 'int argc' (cardinalità). Per convertire i vari 'argv[2], argv[3], ...':

- 'atoi' : converte stringa di 'argv[]' in integer.
- 'atof' : converte stringa di 'argv[]' in float.

Esempio 1 di acquisizione da riga di comando (non c'è 'scanf()'):

```
/* main per funzione  fattoriale_r*/
int main(argc, char *argv[]){
        int val, ris;
        char *numero;
        if(argc==2){
                numero = argv[1];
```

```
                val = atoi(numero);
                ris = fattoriale_r(val);
                printf("%d\n", ris);
        }
        else
                printf("!!!        parametri errati 2!!!\n");
        }

}
```

Esempio 2 di acquisizione da riga di comando:

```
/* main per funzione  */
int main(argc, char *argv[]){
        int val, ris;
        char *seq;
        if(argc==3){
                seq = argv[1];
                c_str = argv[2];
                c = cstr[0];
                ris = verifica_presenza(seq, c);
                printf("%d\n", ris);
        }
        else
                printf("!!!        parametri errati 2!!!\n");
        }
}
```

5.3.4 Contatori

Contatori: sono variabili come i, j che servono per tenere il conto del numero di iterazoni svolte, siccome sono operazioni utilizzate frequentemente, si può possono scrivere nella forma:

- Operatore di incremento: $num = num + 1; \implies num++;$
- Operatore di decremento: $num = num - 1; \implies num--;$

Gli operatori di incremento e decremento si possono scrivere anche nella forma ++num; e –num, ma noi non useremo il postincremento.
Si possono abbreviare anche le operazioni in questo modo:

- Somma: $tot = tot + cont \implies tot+ = cont$
- Differenza: $tot = tot - cont \implies tot- = cont$
- Prodotto: $ris = ris * val \implies ris* = val$
- Divisione: $ris = ris/val \implies ris/ = val$

5.3.5 Casting

L'operazione di casting ha la sintassi '(tipo_variabile)nome_variabile_castare'. È importante sottolineare che tronca la visualizzazione della variabile, non fa cambiare il valore di essa, infatti bisogna assegnarlo per salvarlo in un'altra variabile.
La'operazione ha priorità maggiore ripetto agli operatori classici, ad esempio scrivendo 'avg=(float)tot/num' viene svolto prima casting di 'tot' a reale e poi viene svolta la divisione fra reali, mentre per svolgere prima la divisione si scrive 'avg=(float)(tot/num)'.
Notiamo che spesso questa operazione è ridondante perchè se assegnamo ad un intero (int) un valore di tipo reale (float), esso viene troncato perchè non ci sono abbastanza bit per rappresentarlo.

```c
/* Cambiare tipo di una variabile */
int main(int argc, char*argv[])
{
        float val;
        int dif;
        scanf("%f",&val);
        dif=(int)val;         /*cast esplicito*/
        print("%d\n", dif)
}
```

5.4 Costrutti in C

```c
/* Elenco di tutta la sintassi da imparare a memoria */
#include <_nome_>   #define SYMB VAL
typedef _definizione_ _nomet_;
typedef struct _nomes_ _definizione_ _nomet_;
int main(int argc, char * argv[])   return   /* */   ;   ,
_tipor_ _nomesottop_(_tipoin_ _nomep_, _tipoin_ _nomep_);
sizeof   (_tipo_)   ()   []   {}
void   int _nomev_;   char _nomev_;   float _nomev_;
FILE * _nomev_;   _nomet_ _nomev_;   ''   '\"   '\0'
=   +   -   *   /   %   ++   --   +=   -=   *=   /=
&&   ||   !   ==   !=   >   >=   <   <=   ->   .   &   *
if(_espr_) _istr_   if(_espr_) { _istr_ _istr_ }
if(_espr_) _istr_ else _istr_
if(_espr_) _istr_ else if (_espr_) _istr_ else _istr_
if(_espr_){ if(_espr_) } else _istr_
switch(_espr_){ case _caso_: _istr_ case _caso_: _istr_
default: _istr_ }   break
while(_espr_) _istr_   while(_espr_){ _istr_ _istr_}
do _istr_ while(_espr_);   do{ _istr_ _istr_} while(_espr_);
for(_espr_;_espr_;_espr_) _istr_
scanf   printf   gets   fopen   fclose   feof   EOF   fscanf
fprintf   fgets   fread   fwrite   malloc   free   atoi
strlen   strcmp   strcpy
```

5.4.1 Costrutto if

```c
/* Sintassi if base */
if(espressione)
    istruzione_Vero;
```

Per scrivere 3 espressioni nell'if bisogna utilizzare le parentesi graffe per riunire costrutti che vanno svolti dopo una condizione, senza graffe solo l'espressione successiva viene eseguita (l'indentazione è solo un aspetto visivo).

```c
/* Sintassi if con più istruzioni*/
if(espressione){
    istruzione1_Vero;
    istruzione2_Vero;
    istruzione3_Vero;
}
```

Programma valore assoluto con cotrutto if:

```c
/* Valore assoluto */
int main(int argc, char*argv[]){
    float val;
    scanf("%f", &val);
    if(val < 0}
        val=-val;
    printf("\d", val);
}
```

Programma valore assoluto con cotrutto if else:

```c
/* Valore assoluto salvando dato iniziale */
int main(int argc, char*argv[]){
    float val;
    scanf("%f", &val);
    if(num < 0}
        abs=-val;
    else if(val > 0) /*ulteriore condizione ridondante, anche solo: else operazione*/
        abs=val;
    else
        operazione
    printf("\d", abs);
}
```

Se una istruzione si trova sia nell'if che nell'else significa che si svolge sempre, questa istruzione è ridondante ed è consigliabile metterla prima dell'if in modo da scriverla una volta sola.

Programma che acquisisce valore input, se negativo restituisce '-', se positivo '+' e se nulla uno spazio ' ' (if-else-annidato):

```c
/* Retituisce simbolo in base al segno del valore */
#define POS '+'
#define NEG '-'
#define NUL ' '
int main(int argc, char*argv[]){
    int val;
    char ris;
    scanf("%d", &val);
    if(val>0)
        ris = POS;
    else
        if(val<0)
            ris = NEG;
        else
            ris = NUL;
    printf("%c",ris);
}
```

In alcuni linguaggi come C, C++ e Python si può scrive nella stessa riga anche 'if else' senza indentare la condizione rispetto al 'if' principale:

```c
/* Sintassi if ad elenco */
if(espressione)
    istruzione1_Vero;
else if(espressione)
    istruzione2_vero;
else if(espressione)
    istruzione3_vero;
```

```
else intruzione4_finale;
```

È importante sottolineare che else è legato al precedente if che non ha già un altro else legato (non possono esseci due else per lo stesso if, nel caso si deve creare un if annidato che fa parte del'else del primo if delimitato da graffe).

All'interno dell'if si valutano le condizioni, ma di fatto sono espressioni che retituiscono un booleano '0'(F falso) o '1' (T vero); queste espressioni possono essere combinate tramite operatori logici e il risultato si controlla con le tavole della verità degli operatori logici.

Ad esempio possiamo scrivere 'if(val==0)' per svolgere delle operazioni solo nel caso in cui è vera l'espressione val==0, che è uguale a 'if(!val)'.
La condizione opposta sarebbe 'if(val!=)', che è uguale a 'if(val)'.
Per confrontare con una costante possiamo scrivere 'if(5==val)', così se ci dimentichiamo '==' e mettiamo solo '=' il compilatore segnala errore di sintassi.

5.4.2 Switch

Costrutto di selezione particolare che selezione in base al valore di una variabile il tipo di istruzioni da eseguire. Possiamo usarlo quando abbiamo una variabile di tipo intero o che sottende un intero (quindi anche con char).

Sintassi del contrutto 'switch()':

```
switch(var){
        case 1: istruzione_1;        /* con char:        case 'a': .... */
                        break;        /* necessario per uscire (altrimenti esegue tutto) */
        case 2: istruzione_2;

        default: intruzione_n;        /* facoltativo */
}

```

5.4.3 Ciclo While e Do-while

Costrutto che permette di itereare delle operazioni fin tanto che (in inglese while) una condizione è vera, quando è falsa si esce e continua il flusso lineare del codice. Usando il While la condizione viene posta prima di svolgere le operazioni, mentre usando il Do-while la condizione di uscita viene posta dopo aver svolto le operazioni.
Nel costrutto While è necessario che vengano svolte delle istruzioni che modificano la variabile nela condizione del ciclo, in modo da evitare i loop infinti.

```
/* Sintassi while */
while (espressione)
    istruzione_Vero;
```

Il costrutto while pone la condizione dopo aver svolto una volta tutte le istruzioni, è utili anda esempio per controlla re che il valore inserito in input in una variabile sia accettabile.

```
/* Sintassi while */
Do
    istruzione;
while (espressione);
```

Calcola valore massimo (While):

```c
/* programma che acquisisce 20 valori interi e acquisisce valore massimo */
#include <stdio.h>
#define N_CICLO 20
int main(int argc,char*argv[]){
        int cont=1,num,max;
        scanf("%d\n",&max);
        while(cont<N_CICLO){
                scanf("%d\n",&num);
                if(max<num)
                        max=num;
                cont++;
        }
        printf("%d\n",max);
        return 0;
}
```

Ricevere dati in input fin che non si inserisce numero specifico (While):

```c
/* programma che acquisisce primo valore intero 'fine' seguito da una sequenza di valori interi
a priori di lunghezza ignota che si ritiene termianta quando l'utente inserisce tale valore
massimo minimo e media dei valori, se subito valore fine visualizza max min e medi 27*/
#include<stdio.h>
int main(int argc,char*argv[]){
        int stop, val, max, min, tot;
        float avg, count;
        scanf("%d\n%d",&stop,&val);
        max=val;
        min=val;
        count=0;
        while(stop!=val){
                scanf("%d",&val);
                count++;
                tot+=val;
                if(max<val)
                        max=val;
                else if(min>val)
                        min=val;
        }
        if(min!=stop)
                avg=(float)tot/count;
        else
                avg=stop;
        printf("massimo: %d\nminimo: %d\nmedia: %f\n", max, min, avg);
        return 0;
}
```

Acquisire numero positivo e calcolarne numero cifre (Do-while):

```c
/* chiedere all'utente valore fin che non è strettamente positivo, poi calcolare numero di cifre da cui è composto */
#include<stdio.h>
#define BASE 10
int main(int argc,char*argv[]){
        int val,cifre,count;
        do{
        scanf("%d",&val);
        }while(val<=0); // input numero fin che non è positivo
        count=val; // così salvo valore iniziale
        cifre=0;
```

```
        do{
                count/=BASE; // divido numero per 10
                cifre++; // incremento il contatore delle cifre
        }while(count>0);// fin che il numero non diventa minore di zero (num>(int)0.999=0)
        printf("%d\n",cifre);
        return 0;
}
```

5.4.4 Costrutto for

Il costrutto for è extra e facoltativo, ma utile per fare iterazioni a conteggio, in particolare è utile per getisre dati salvati nin array.
Il for é un ciclo a condizione iniziale, come while con contatore che fa finire il ciclo dopo un numero definito a priori di iterazioni.

La sintassi (pseudocodice) del ciclo for è la seguente:

```
/* Sintassi pseudocodice ciclo for */
#define numero delle iterazioni
int contatore;
for(inizio contatore; condizione di uscita; incremento contatore)
        operazioni ed elaboraizoni
```

La sintassi (base) del ciclo for è la seguente:

```
/* Sintassi base ciclo for */
#define ITERAZIONI numero_iterazioni
int contatore;
for(contatore=0; contatore<ITERAZIONI; contatore++)
        operazioni ed elaboraizoni
```

La sintassi (completa) del ciclo for è la seguente:

```
/* Sintassi condizioni multiple ciclo for */
#define ITERAZIONI numero_iterazioni
int contatore;
for(count1=0, count2=0; condiz1 && condiz2 || condiz3; count1++, count2--){
        operazione1;
        operazione2;
        operazione3;
}
```

Lo scopo del for è scrivere in maniera più compatta un particolare ciclo while che fa scorrere l'array. Non è da usare sempre il for al posto del while, al limite aggiungere condizione di uscita dal ciclo for nella sezione delle condizioni oltre al fatto di non scorrere array otre la sua dimensione.
Non si devono dichiarare variabili locali che valgono solo nel for, meglio usare solo variabili globali. Considerare che le varibili globali usate nel for non si azzerano e il valore non cambia.

Esercizi con il ciclo for:
Conversione da decimale a binario (configuarazione con tutti i bit):

```
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo,
programma calcola e visualizza la sua rappresentazione nel sistema binario (tutti i bit) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
```

```c
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num, binario[BIT], resto, i;
        do
                scanf("%d",&num);
        while(!(num>=MIN && num<=MAX)); /* anche (num<MIN || num>MAX)*/
        resto = num;
        for(i=0; i<BIT; i++){
                binario[i] = resto % BASE;
                resto = resto / BASE;
        }
        for(i=BIT-1; i>=0; i--){
                printf("%d",binario[i]);
        }
        printf("\n");
        return 0;
}
```

Conversione da decimale a binario (configuarazione con solo i bit necessari):

```c
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo,
programma calcola e visualizza la sua rappresentazione nel sistema binario (minimo bit) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num, binario[BIT], resto, i;
        do
                scanf("%d",&num);
        while(!(num>=MIN && num<=MAX)); /* anche (num<MIN || num>MAX)*/
        resto = num;
        i = BIT-1;
        while(resto>0){ /* non usare for perchè ciclo a condizione e non conteggio */
                binario[i] = resto % BASE;
                resto = resto / BASE;
                i--;
        }
        for(i++; i<BIT; i++){
                printf("%d",binario[i]);
        }
        printf("\n");
        return 0;
}
```

Conversione da decimale a binario di due numeri (configuarazione a bit necessari del numero maggiore):

```c
/* programma che acquisito valore natirale 1<=num<=1023 e fin che non è tale lo richiedo, programma calcola e
visualizza  a sua rappresentazione nel sistema binario2 usando lo stesso numero di cifre (minimo indispensabile) */
#include <stdio.h>
#define BASE 2
#define MIN 1
#define MAX 1023
#define BIT 10 /* formula configuazioni bit BIT=log_BASE(MAX-MIN)=10 */
int main(int argc, char*argv[]){
        int num1, num2 , binario1[BIT], binario2[BIT], i, supp, k;
        do
                scanf("%d",&num1);
        while(!(num1>=MIN && num1<=MAX)); /* anche (num<MIN || num>MAX)*/
```

```
        do
                scanf("%d",&num2);
        while(!(num2>=MIN && num2<=MAX));
        if(num2>num1){
                supp = num1;
                num1 = num2;
                num2 = supp;
        }
/* numero in binario 1 */
        i = BIT-1;
        while(num1>0){
                binario1[i] = num1 % BASE;
                num1 = num1 / BASE;
                binario2[i] = num2 % BASE;
                num2 = num2 / BASE;
                i--;
        }
        k = i;
/* output */
        for(i++; i<BIT; i++){
                if(num1>=num2)
                        printf("%d",binario1[i]);
                else
                        printf("%d",binario2[i]);
        }
        printf("\n");
        for(k++; k<BIT; k++){
                if(!(num1>=num2))
                        printf("%d",binario1[k]);
                else
                        printf("%d",binario2[k]);
        }
        printf("\n");
        return 0;
}
```

5.4.5 Ricorsione

Metodo particolare di risolvere un problema. Scrivere un sottoprogramma che, o direttmente o indirettamente, richiama se stesso.

```
// Ricorsione diretta
___f(...){
                istruzioni;
}

___g(...){
                istruzioni;
}

// Ricorsione indiretta
___f(...){
        ___g(...){
                istruzioni;
        }
}

___g(...){
        ___f(...){
```

```
                istruzioni;
        }
}
```

Sottoprogramma di esempio (fattoriale) del metodo ricorsivo:

```
// ricorsione fattoriale:       metodo 1
int fattoriale(int num){
        int ris;
        if(num==0 || num==1)
                ris = 1;
        else
                ris = num * fattoriale(num-1);
        return ris;
}
```

// ricorsione fattoriale: metodo 2 (compatto)

```
int fattoriale(int num){
        if(num==0 || num==1)
                return 1;
        return num * fattoriale(num-1);
}
```

Esempio di sottoprogramma ricorsivo:

1. 'strlen()':

```
                /* sottoprogramma ricorsivo che ricevuta in ingreso una stringa,
                conta e restituisce al chiamante la lunghezza della stringa */
                strlen_r(char stringa[]){
                        if(stringa[0]=='\0')
                                return 0;
                        return 1+strlen_r((&stringa[0])+1);
                }
```

2. Contare ripetizioni carattere:

```
                        /* sottoprogramma che ricevuto in ingresso stringa e carattere conta quante volte è contenuto */
                        conta_ripetizioni(char stringa[], char carattere){
                                int count;
                                if(stringa[0]=='\0')
                                        count = 0;
                                if(s[0]==carattere)
                                        count = 1;
                                else
                                        count = 0;
                                count += conta_ripetizioni(&stringa[1], c);
                                return count;
                        }
```

3. Controllare se compare carattere in una stringa:

```
                        /* sottoprogramma che ricevuto in ingresso stringa e carattere verifica se compare o meno */
                        verifica_presenza(char stringa[], char carattere){
                                int count;
```

```
                if(stringa[0]=='\0')
                        return 0;
                else if(s[0]==carattere)
                        return 1;
                else
                        return verifica_presenza(&stringa[1], c);
        }
```

5.5 Strutture dati

5.5.1 Array monodimensionali

gli array monodimensionali sono anche detti vettori, ad esempio $\vec{v}[5] = (10, 4, 3, 2, 1)$
Per ora abbiamo gestito situazioni e problemi dove non era necessario salvare i valori di cicli indefiniti, i dati in se li abbiamo cancellati e salvavamo i risultati parziali delle iterazione in delle variabili.

Se dobbiamo salvare 50 valori non è comodo utilizzare 50 variabili, quindi introduciamo una nuova struttura dati detta array che può avere cardinalità definita dal programmatore. La variabile è un array[1][1]

Definition 5.1 (Array)**.** *è qualle struttra dati (variabile strutturata) che ci permette di definire una cardinalita con un unico nome che stabilisco a priori che conterra un numero definito di valori.*

Un array contiene dati omogenei, tutti interi o float ecc, e questa quantità è costante e nota a priori. Nella fase di compilazione deve essere nota a priori, nello standard ANSI non usiamo array dinamici (faremo alloracizone dinamica malloc).

In linguaggio C si parte da elemento 0, quindi se abbaimo un array di dimensione 10 il primo valore sarà in posizione 0 e l'ultimo in posizione 9. Sintassi dichiarazione e utilizzo degli array:

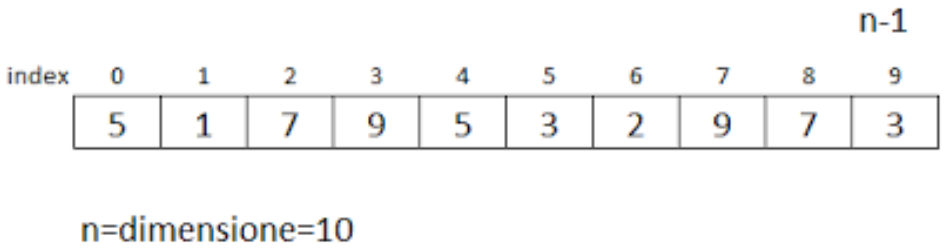

Figure 18: Array monodimesionali

```
/* Sintassi array monodimensionale */
#define NUM_ELEMENTI numero elementi
tipo_array nome_array[NUM_ELEMENTI];
int numeri[NUM_ELEMENTI];
float soldi[30];
```

Per accedere agli elementi si utilizza ciclo e variabili contatatori (es: i, j, h, k).
Ciclo while che fa scorrere elementi in input, sintassi per popolazione di array monodimensionale:

```
/* Sintassi popolazione array monodimensionale */
int numeri[num_elementi]
while(i<num_elementi){
        scanf("%d",&numeri[i])
}
```

Esercizio di esempio, calcolo della media dei valori inseriti e printa valori dell'array superiori alla media:

```c
/* Programma che acquisice 50 valori interi e visualizza valori supriori al valore medio */
#include <stdio.h>
#define CELLE 5
int main(int argc,char*argv[]){
        int i, valore[CELLE],tot;
        float avg;
        i = 0;
        tot = 0;
        while(i<CELLE){
                scanf("%d",&valore[i]);
                tot += valore[i];
                i++;
        }
        avg = (float)tot / CELLE;
        i = 0;
        while(i<CELLE){
        if(valore[i]>avg)
                printf("%d\t",valore[i]);
        i++;
        }
        printf("\n");
        return 0;
}
```

5.5.2 Array multidimensionali

Struttura dati oraganizzata su due dimensioni (es: immagine, tabella). Se si devono rappresentare una lista di 50

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Figure 19: Array multidimensionali in C

numeri si usano array monodimensionali, mentre array multidimensionali si usano per le matrici; qui riporto la sintassi base di un array multidimensionale:

```c
/* Sintassi array multidimensionale base */
tipo_array nome_array[num_righe][num_colonne];
int mat[N_RIGHE][N_COLONNE];
```

Si usano le varibili indice 'i' e 'j' che rappresentano rispettivamente il nuemro di colonne e il numero di righe. Dichiarazione e popolazione di un array multidimensionale:

```c
/* Sintassi popolazione array multidimensionale */
#define N_RIGHE 8
#define N_COLONNE 5
int mat[N_RIGHE][N_COLONNE], i, j;
for(i=0;i<N_RIGHE;i++){
        for(j=0;j<N_COLONNE;j++){
                scanf("%d",mat[i][j]);
        }
}
```

Esercizio con array bidimensionale:

```c
/* programma che acquisisce dati array buismensionele quadrato di dimensione 5
programma calcola e visulaiiza 1 se la matrice è identità */
#include <stdio.h>
#define DIM 5
int main(int argc,char*argv[]){
        int matrice[DIM][DIM], i, j, ris;
        for(i=0;i<DIM;i++){ /* Input matrice */
                for(j=0;j<DIM;j++){
                        printf("riga %d, colonna %d: ",i+1,j+1);
                        scanf("%d",&matrice[i][j]);
                }
        }
        ris = 1;
        for(i=0;i<DIM && ris != 0;i++){ /* Iscorrimento righe */
                for(j=0;j<DIM && ris != 0;j++){ /* scorrimento colonne */
                        /* controllo  se diganoale è 1 e resto 0*/
                        if((i==j && matrice[i][j]!=1) || (i!=j && matrice[i][j]!=0))
                                ris = 0;
                }
        }
        printf("%d\n",ris);
        return 0;
}
```

.

Linearizzazione della memoria:
la memoria è monodimensionale (vettore-colonna), di conseguenza bisogna portare in forma monodimensionale un array bidimensionale. Di per sé si potrebbe gestire una matrice multidimensioanle come un vettore, ma risulterebbe complicato nellla lettura da parte di un altro utente. Esercizio con gli array multidimensioneali, verificare se un

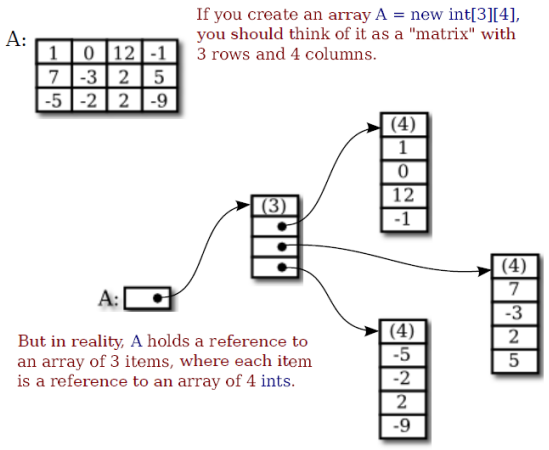

Figure 20: Linearizzazione della memoria

quadrato è magico (vedere se la somma delle righe, colonne e diagonali è uguale)

```c
/* verificare s è quadrato magico (somma righe / colonna / digonali è uguale) output si '1' // no '0' */
#include <stdio.h>
#define DIM 3
int main(int argc, char*argv[]){
        int matrice[DIM][DIM], i, j, mag, sum1, sum2, sum3;
        mag = 1;
```

```
        sum1 = 0;
        sum2 = 0;
        sum3 = 0;
        for(i=0;i<DIM;i++){ /* Input matrice */
                for(j=0;j<DIM;j++){
                        printf("riga %d, colonna %d: ",i+1,j+1);
                        scanf("%d",&matrice[i][j]);
                }
        }
        for(i=0;i<DIM;i++) /* scorrimento digonale principale */
                        sum1 += matrice[i][i];
        for(i=0;i<DIM;i++) /* scorrimento digonale secondaria */
                        sum2 += matrice[i][DIM-i-1];
        if(sum1!=sum2)
                mag = 0;
        for(i=0;i<DIM && mag!=0;i++){ /* scorrimento esterno righe/colonne */
                sum1 = 0;
                sum3 = 0;
                for(j=0;j<DIM && mag!=0;j++){
                        sum1 += matrice[i][j];
                        sum3 += matrice[j][i];
                }
                if((sum1!=sum2) || (sum3!=sum2))
                                mag = 0;
        }
        printf("%d %d\n",mag,sum1);
        return 0;
}
```

5.5.3 Structure

Salvare dati in struttura dati come array ma senza vincolo del tipo, struct possono contenere diverse tipologie di variabili, array mono e multidimensionali, il tutto gestito come unica identità e campi di tipologia diversa (La typedef va messa prima del main).

Sintassi del costrutto struct:

```
/* Sintassi base struct */
struct nome_struttura_s{
        tipo1 nome1;
        tipo2 nome2;
}
```

Esempio di utilizzo:

```
/* Sintassi esempio 1 struct */
struct data_s{
        int giorno, mese;
        float soldi;
}
```

Esempio di struct composto da variabili e array di tipo diverso:

```
/* Sintassi esempio 2 struct */
struct studente_s{
        char cognome[MAX_LEN+1];
        char nome[MAX_LEN+1];
        int erasmus;
```

```
        bool laureato;
        float media;
}
```

Per definire un nuovo tipo di struct e poterlo richiamare per utilizzarlo più volte nel programma si deve definire il tipo di struttra dati con il comando '*typedef*' (non è necessario sia dichiare struttura e definire nuovo tipo):

```
/* Sintassi typedef struct */
typedef struct nome_struttura{
        tipo1 nome1;
        tipo2 nome2;
}nome_struttura_t;
```

Esempio di utilizzo del comendo '*typedef*':

```
/* Sintassi typedef esempio struct */
typedef struct data_s{
        int giorno;
        int mese;
        int anno;
}data_t;

data_t nascita_s;
nascita_s.giorno = 13; /* assegnamento mirato */
nascita_s.mese = 03;
scanf("%d",&nascita.anno);
```

Confronto fra date:

```
/* acquisire 50 date di nascita e vedere quanti sono nati nello stesso giorno */
#include <stdio.h>
#define NUM 50
typedef struct data_s{
        int giorno, mese, anno;
}data_t;

int main(int argc,char*argv[]){
        int i, nascita, comp;
        data_t persone_s[NUM], data_rif_s;
        for(i=0;i<NUM;i++)
                scanf("%d %d %d", &persone_s[i].giorno, &persone_s[i].mese, &persone_s[i].anno);
        scanf("%d %d %d",&data_rif_s.giorno, &data_rif_s.mese, &data_rif_s.anno);
        nascita = 0;
        comp = 0;
        for(i=0; i<NUM; i++){
                if(persone_s[i].mese==data_rif_s[i].mese)
                        if(persone_s[i].giorno==data_rif_s[i].giorno){
                                comp++;
                                if(persone_s[i].anno==data_rif_s[i].anno)
                                        nascita++;
                        }
        }
        printf("%d\t%d\n");
        return 0;
}
```

5.5.4 Strighe

Stringa è struttura dati che non fa accedere a singolo carattere di array, si risparmia nella complessità di gestione infatti è presente un terminatore non contenuto nella stringa.
Array di n caratteri corrisponde ad una stringa di $n+1$ caratteri perchè è presente un carattere che per definizione fa da carattere terminatore, in c è presente il carrattere non visibile '\0'.

Sintassi gestione stringhe:

```c
/* Sintassi stringhe base */
scanf("%s", &seq[0]);
printf("%s\n", &seq[0]);
```

In un array in c, scrivere '&seq[0]' corrisponde a 'seq', quindi è implicito che si operi sulla posizione 0 del vettore che corrisponde alla cella z(0, 0) della matrice.

Per lo scanf spazio ' ', tab ' ' e a capo '\n', quindi si devono sostituire spazi con underscore o trattini per evitare che si fermi la stringa perchè lo spazio viene visto come terminatore.

Per fare ciò è necessario includere la libreria 'string', che si scrive con '#include ¡string.h¿'
Un altro metodo alternativo allo scanf() è get();

```c
/* Sintassi stringhe con gets */
gets(seq);
gets(&seq[0]);
```

Con 'gets()' i tab e spazi sono considerati caratteri, metre il terminatore è solamente a capo 'È pericolosa perchè rischia di sovrascrivere memoria dopo quella dedidcata all'array, con gets si riceve infatti un warning ma lo ignoriamo perchè l'utente sono inserirà più caratteri rispetto al limite.

Sintassi con define:

```c
/* Esempio programma per vedere sintassi stringhe */
        #define LMAX 100
        int main(int argc, char *argv[]){
                char frase[LMAX+1], mask[LMAX+1];
                gets(frase);
                c = INIZIO + imax;
                i = 0;
                while(frase[i] != '\0'){ /* for(i=0; frase[i]!='\0'; i++) */
                        if(frase[i] == c)
                                mask[i] = SOST;
                        else
                                mask[i] = frase[i];
                        i++;
                }
                mask(i)=frase[i]; /* aggiungere terminatore */
        }
```

Contare numero lettere in una stringa:

```c
/* acquisisce sequenza al più 100 caratteri e terminata con carattere '*'
calcolare il carattere più frequente e visualissa frase in cui ha sostituito
il carattere più frequente con '*' - caratteri minuscoli, spazi e interpunzione
e non cè asterisco per primo e a parità di frequenza carattere
preso con max_frequente à l'ultimo nell'alafabeto*/
#define <stdio.h>
#define MAX 100
#define STOP '*'
#define INIZIO 'a'
```

```c
#define FINE 'z'
int main(int argc, char *argv[]){
        char frase[MAX];
        int dim, i, imax, pos, lettere[MAX];

        /* acquisizione */
        dim = 0;
        scanf("%c", &c);
        while(c != STOP){
                frase[DIM] = c;
                dim++;
                scanf("%c", &c);
        }

        /* contatore e inizializzazione*/
        for(i=0;i<ALFA; i++){
                lettere[i] = c;
        }

        /* analisi primo carattere */
        c = frase[0];
        pos = c - INIZIO;
        lettere[pos] = 1;
        imax = pos;

        /* analisi caratteri rimannenti */
        for(i=1; i<DIM; i++){
                if(frase[i]>=INIZIO && frase[i]<=FINE){
                        pos = frase[i] - INIZIO;
                        lettere[pos]++;
                        if(lettere[pos]>lettere[imax])
                                imax = pos;
                        else if(lettere[pos]==lettere[imax])
                                if(pos>imax)
                                        imax = pos;
                }
        }
        /* sostintuzione */
        c = INIZIO + imax;
        for(i=0; i<dim; i++0){
                if(frase[i] == c)
                        frase[i] = SOST;
                printf("%c", frase[i]);
        }
        printf("\n");
        return 0;
}
```

Stabilire se una stringa è palindroma confrontando elementi array[i] e array[DIM-i-1]:

```c
/* scrivere programma che acquisisce una stringa di al piu 100 carratteri, calcola e visualizza '1' so sono palindrome '0' se no */
#include<stdio.h>
#include<string.h>
#define MAX 100
#define PAL 1
#define NO_PAL 0
int main(int argc, char *argv[]){
        char stringa[MAX+1], ris;
        int i, leng;
        gets(stringa);
        ris = PAL;
```

```
        for(leng=0; stringa[leng]!='\0'; leng++);
        i = 0;
        while(ris==PAL && i<leng){
                if(stringa[i]!=stringa[leng-1-i])
                        ris = NO_PAL;
                i++;
        }
        printf("%d\n", ris);
        return 0;
}
```

Contare numero vocali e consonanti in una stringa:

```
/* programma che conta consonanti e vocali, utente inserisce solo carartteri minuscoli*/
#include<stdio.h>
#include<string.h>
#define MAX 100
#define INIZIO 'a'
#define FINE 'z'
int main(int argc, char *argv[]){
        char seq[MAX+1];
        int voc, con, i;
        voc = 0;
        con = 0;
        gets(seq);
        for(i=0; seq[i]!='\0';i++){
                if(seq[i]>=INIZIO && seq[i]<=FINE)
                        if(seq[i]=='a' || seq[i]=='e' || seq[i]=='i' || seq[i]=='o' ||seq[i]=='u')
                                voc++;
                        else
                                con++;
        }
        printf("voc: %d\t cons: %d\n", voc, con);
        return 0;
}
```

5.5.5 Allocazione dinamica

L'indirizzo che non guarda da nessuna parte ha contenuto 'NULL'. Il tipo della varibile che vado a chiamare è un indirizzo di un intero, che all'inizio del programmma sarà senza contenuto. Dopo la variabile la assegno a dove mi è stata data la memoria.

È necessario includere la libereria '¡stdlib.h¿' e la memoria che sarà occupata dall'allocazione dinamica non è lo stack, ma lib.

Le funzioni per l'allocazione dinamica è la 'memory alloc' abbreviata con 'malloc()' :

```
/* Spiegazione della funzione malloc */
void * malloc(size_t)

/* la funzione 'sizeof()' restituisce una varibile di tipo 'size_t' */
sizeof(int) = "quanto occupa di memoria (varia in base ad architettura del calcolatore)"

var = mallc()
if(var!=NULL){        /* if(var) */
        /* operazioni */
        free(var)               /* void free(void*) */
}
else
```

```
        printf("errore di allocazione \n")

```

Se la memoria si esarisce la malloc restituisce 'NULL'. Quindi devo chiedere memoria, se risultato malloc è diverso da 'NULL' procedo con algoritmo, else probblema di memoria.

Devo fare la free solo se il cpntenuto della memoria non serve più, nel caso dopo la si mette a fine del main. La memoria restituitsa dal programmma allocata dionamicamentie non si cancella, quindi è il mezzo ideale per restituire al chiamante da funzione. Si libera e non èpiù accessibili solo dopo la 'free()'.

Esercizio di esempio con allcoazione dinamica:

```
/* Programma che ordina  */
int main(inr argc, char *argv[]){
        int *v;
        int n, i;
        do
                scanf("%d", &n);
        while(N<=1);
        v = (int*)malloc(n*sizeof(int));
        if(v!=NULL){
                for(i=0; i<n; i++)
                        scanf("%d", v+i);        /* &v[i] */
                bsort(v,n);
                for (i=0, i<n; i++)
                        printf("%d\t", *(v+i));        /* v[i] */
                printf("\n");
                free(v);
        }
        else
                printf("errorre di allocazione %d int", n);
        return 0;
}

```

Esercizio allocazione dinamica con sottoprogramma:

```
/* crea e restituisce una nuova stringa che crea e restituisce le consonanti della stringa di partenza.
Scriviamo iun programma da riga di comanfo che acquisita da riga di comando una stringa,
avvalendosi del sottoprogramma, visualizza le consonanti presenti */

int main(int argc, char *argv[]){
        char *seqin;
        char *cseq;
        if(argc==2){
                seqin = argv[1];
                cseq = consonanti(seqin);
        }
        if(cseq!=NULL){
                printf("%s\n", cseq);
                free(cseq);
        }
        else
                printf("argomenti non coerenti\n");
        return 0;
}

char * consonanti(char s[]){
        char *sc;
        int nc, i, j;
```

```
        nc = 0;
        for(i=0; s[i]!='\0'; i++)
                nc += is_cons(s[i]);
        sc = (char *)malloc((nc+1)*sizeof(char));
        if(sc!=NULL){                /* anche malloc e if uniti (NULL==0000000) */
                J = 0;
                for(i=0; s[i]!='\0'; i++){
                        if(is_cons(s[i])==1){
                                *(sc+j) = s[i];
                                j++;
                        }
                }
                *(sc+j) = '\0';
        }
        else
                printf("errorre di allocazione %d char", nc);
        return sc;
}

int is_cons(char caratt){
        if(isalpha(caratt)){
                if(caratt!= vocali)
                        return 1;
                else
                        return 0;
        else
                return -1;
}
```

5.5.6 Linked list

Struttura dati per memorizzare informazioni senza sapere all'inizio il numero di elementi, si possono poi modificare anche in corso le dimensioni.
Permette di allocare memoria dinamicamente tenendo traccia di dove sono i dati, ed eventualmente una volta non più utili utilizzando 'free()' si libera la memoria.
Si deve creare una struttura dati che contiene il valore di quell'elemento della lista e il puntatore all'elemento successivo.

```
/* struttura base di una lista */
typedef struct element{
        int val;
        int *point;
}nodo;
ilist_t *h NULL;
/* sottoprogrammi sulle liste */
ilist_t * append(ilist_t *, int ){        /* (tipo: testa di lista aggiornato; parametri: lista, valore da aggiungere) */4
ilist_t * insert_in_order()        /* aggiunge in ordine (così non si ordina) */
ilist_t * push()         /* aggiunge in coda */
}
```

Esercizio di esempio sulle liste concatenate:

```
/* chiedere a utenete valore intero sentinella, inserire valori che vuole fin che non inserisce il valore sentinella.
Visualizzare tutti i valori maggiori di quelli acquisiti fino a quel momento */
#include<stdio.h>
#include<stdlib.h>
typedef struct ilist_s{
        int val;
```

```
        struct list_s *next;
}ilist_t;

ilist_t *append(ilist_t *, int);         /* dichiarazione prototipo */

int main(int argc, char *argv[]){
        ilist_t *h = NULL;                /* dichiaro puntatore nullo perche lista vuota */
        ilist_T *elem;
        int stop, num, tot, cont;
        float avg;
        scanf("%d", &stop);
        scanf("%d", &num);
        while(num!=stop){
                h = append(h, num);
                scanf("%d", num);
        }
        tot = 0;
        cont = 0;
        elem = h;
        while(elem!=NULL){
                tot += elem->val;         /* sommo valore elemento */        /* 'elem->val' come  '*elem.val' */
                cont++;                /* mi sposto all'elemento successivo */
                elem = elem->next;                /* 'elem->next' come '*elem.next' */
        }
        if(cont>0){
                avg = (float)tot/cont;
                for(el=h; el!=NULL; el=el->next){               /* 'el!=null' come 'el' */
                        if(el->val>avg)
                                printf("%d\t", el->val);
                        else
                                printf("lista vuota");
                        freelist(h);        /* programma che libera la lista come 'free(testa_lista)' */
                }
        }
}
```

Esercizio 2 (versione 1):

```
/* sottoprogramma che riceve array e retituisce tutti e soli i numeri primi nell'array */
typedef struct elem{
        int val;
        ilist_t *elem;
} ilist_t;
ilist_t * append(ilist_t *testa, int num);
int is_prime(num);
ilist_t * solo_primi(int arr[], int dim){
        int i;
        ilist_t *h = NULL;
        for(i=0; i<dim; i++){
                if(is_prime(arr[i])!=0)
                        h = append(h, arr[i]);
}
return h;
}
```

Esercizio 2 (versione 1):

```
/* sottoprogramma che riceve array e retituisce tutti e soli i numeri primi nell'array */
typedef struct elem{
        int val;
```

```
        ilist_t *elem;
} ilist_t;
ilist_t * inserisci_ordinato(ilist_t *testa, int num);
ilist_t * find(ilist_t *testa, int num);
int is_prime(num);
ilist_t * solo_primi(int arr[], int dim){
        int i;
        ilist_t *h = NULL;
        for(i=0; i<dim; i++){
                if(is_prime(arr[i])!=0 && find(h, arr[i])==NULL)
                        h = append(h, arr[i]);
}
return h;
}
```

Esercizio 3:

```
/* programma che prende input valori interi che termina con '0', visualizza in ordine inverso */
#define STOP 0
typedef struct elem{
        int val;
        ilist_t *elem;
} ilist_t;
int main(int argc, char **argv){
        ilist_t *h = NULL;
        int val;
        scanf("%d", &val);
        while(val!=STOP){
                h = push(h, val);
                scanf("%d", &val);
        }
        for(p=h; p!=NULL; p=p->val)
                printf("%d", p->val);
        printf("\n");
        freelist(h);
        return 0;
}
```

Sttoprogrammi utili per le liste:

- 'push()' inserimento in testa alla lista
    1. richiedere memoria con 'malloc' e ricevo l'indirizzo
    2. assegno all'indirizzo il nuovo valore
    3. dico dove deve guardare il nuovo primo elemento (all'ex primo)
    4. sposto la testa e dico che guarda al nuovo primo elemento
    5. se la lista è vuota funziona comunque correttaeìmente

```
ilist_t * push(ilist_t *head, int num){
        ilist_t *n;
        n = (ilist_t*)malloc(sizeof(ilist_t));                /* punto 1 */
        if(n!=NULL){
                n->val = num;                /* punto 2 */
                n->next = head;                /* punto 3 */
                head = n;                        /* punto 4 */
        }else
                printf("push: errore di allocazione\n");
        return head;
}
```

- 'append()' inserimento alla fine
  1. richiedere memoria con 'malloc' e ricevo l'indirizzo
  2. assegnare valore da appendere alla struttura uscita dalla malloc
  3. assegnare puntatore ultimo elemento a null
  4. scorrere lista fino all'ultimo elemento e vedere che punta a 'NULL' (gestire anche caso lista vuote)
  5. assegnare puntatore ex ultimo elemento e risultato malloc
  6. se lista vuota 'append()' si riduce alla 'push()'

```
ilist_t * append(ilist_t *head, int num){
        ilist_t *n, *el;
        n = (ilist_t*)malloc(sizeof(ilist_t));                /* punto 1 */
        if(n!=NULL){
                n->val = num;                /* punto 2 */
                n->next = NULL;                /* punto 3 */
                if(head == NULL)
                        head = n                /* punto 4 (caso lista vuota) */
                else
                        for(el=head; el->next != NULL; el=el->next) /* fin che non si arriva alla fine della lita */
                                ;                /* punto 4 (caso lista non vuota) */
                el->next = n;                /* punto 5 */
        }else
                printf("append: errore di allocazione\n");
        return head;
}
```

- 'list_length()' lunghezza di una lista

```
int list_lenght(ilist_t *head){
        ilist_t *el;
        int len;
        el = head;
        len = 0;
        while(el!=NULL){
                len++;
                el = el->next;
        }
        return len;
}
```

- 'find_val()' ricerca valore, si: indirizzo —— no: NULL

```
int find(ilist_t *head, int num){
        ilist_t *el, *ris;
        el = head;
        ris = NULL;
        while(el!=NULL && ris==NULL){
                el = el->next;
                if(el->val == num)
                        ris = el;
        }
        return ris;
}
```

- 'delete()' riceve testa lista, elemento lista e cancella elemento

1. Se devo eliminare primo elemento:
   (a) caso semplice
2. Se devo eliminare elemento in mezzo:
   (a) trovare elemento pre
   (b) trovare elemento post
   (c) fare 'free()' del puntatore all'elemento

```
ilist_t * delete(ilist_t *head, int num){
        ilist_t *pre, *del;
        del = head;
        while(head!=NULL && (head->val == n)){                /* punto 1 caso semplice */
                del = n;
                h = h->next;                  /* h=del->next */
                free(del);                   /* punto 3 */
        }
        if(h!=NULL){
                pre = h;
                while(pre->next){
                        if(pre->next->val == num){                        /* 'pre->next' punta all'elemento successivo
                                                                             quindi 'pre->next->val' punta
                                                                             succesivo al pre, quindi l'ele
                                del = pre->next;                        /* punto 1 */
                                pre->next = del->next;                /* punto 2:               pre->next=pre->next->next; *
                                free(del);                                        /* punto 3 */
                        }else
                                pre = pre->next;
                }
        }
        return head;
}
```

5.5.7 File

Esiste carattere invisibile che si mette nella defiene 'EOF' (End Of File).
Quando apro un file devo controllare che l'apertura vada a buon fine, sia se devo leggere che scrivere per nopn sovrascrivere file con stesso nome.
Ricordarsi di chiudere i file perchè il sistema operativo riesce a getsore solo un certo numero di file aperti, poi si comporta in maniera casuale.
Noi impareremo a gestire i file composti da una sequenza di caratteri ASCII, quando guardo un file ASCII si vedono i caratteri, quando si opera si lavora con i binari almeno non si perde efficienza nella conversione.

Sintassi base della lettura dei file:

```
FILE* fopen(char nome_file[], char modalita_apertura[]);
/* Restituisce 'NULL' se fallisce l'apertura */
/* Restituisce il riferimento al file  */

fclose(FILE*);
/* 'fclose' su 'NULL' retituisce errore */

int fscanf(FILE* , ______________ );

int fprintf(FILE* , ______________ );

int feof(FILE* ) /* retituisce !0 se ce 'EOF' else 0 */

```

```c
fgets(char[], int , FILE* )

fscanf(stdin, "%d", &val);                /* come scanf() */
fprintf(stdout, "%d\n", val);         /* come printf() */

fread(void* , size_t , int , FILE*)
fwrite(void* , size_t , int , FILE*)
/* r: dove sono i dati        \\ w: dove metto i dati */
/* size_t: dimesione del singolo dato */
/* int: dimensione del singolo dato */
/* FILE*: il riferimento del file */
```

Sintassi generica:

```c
/* read */
fscanf(fin, "%c", &c);
while(!feof(fin)){

        /* istruzioni */

fscanf(fin, "%c", &c);
}

/* write */
ris(fin, "%c", &c);
while(ris==1){

        /* istruzioni */

ris = fscanf(fin, "%c", &c);
}
```

Esercizio con lettura file:

```c
/* programma chiede inserire nome file testo ASCII e conta e visalizza
numero caratteri e carattere piu grande */
#define LMAX 40
int main(int argc, char *argv[]){
        FILE* fin;
        char alpha, cmax;
        int nchar, ris;
        char nome[LMAX+1];
        gets(nome);
        fin = fopen(nome, "r")                /* "r" read, "w" write, "rb" read binario*/
        if(fin){
                ris = fscanf(fin, "%c", &alpha);
                if(ris==1){                /* if(!feof(fin)) */
                        cmax = alpha;
                        nchar = 1;
                        while(fscanf(fin, "%c", &alpha)==1){        /* retiruisce 1 se non 'EOF' */
                                /* lavoro sul dato letto */
                                nchar++;
                                if(alpha>cmax)
                                        cmax = alpha;
                                }
                }
                else{
                        /* file c'e ma è vuoto */
                        nchar = 0;
                }
```

```
                fclose(fin);
                printf("%d\t[%c]\n", nchar, cmax);
        }
        else
                printf("problemi apertura file %s\n", nome);
        return 0;
}
```

Esercizio con lettura file:

```
/* programma chiede inserire nome file testo ASCII e conta e visalizza
numero caratteri e carattere piu grande */
#define LMAX 40
int main(int argc, char *argv[]){
        FILE* fin;
        char alpha, cmax;
        int nchar, ris;
        char nome[LMAX+1];
        gets(nome);
        fin = fopen(nome, "r")                /* "r" read, "w" write, "rb" read binario*/
        if(fin){
                ris = fscanf(fin, "%c", &alpha);
                if(ris==1){                /* if(!feof(fin)) */
                        cmax = alpha;
                        nchar = 1;
                        while(fscanf(fin, "%c", &alpha)==1){        /* retiruisce 1 se non 'EOF' */
                                /* lavoro sul dato letto */
                                nchar++;
                                if(alpha>cmax)
                                        cmax = alpha;
                                }
                }
                else{
                        /* file c'e ma è vuoto */
                        nchar = 0;
                }
                fclose(fin);
                printf("%d\t[%c]\n", nchar, cmax);
        }
        else
                printf("problemi apertura file %s\n", nome);
        return 0;
}
```

Esercizio sui file binario:

```
/* realizzare sottoprogramma che riceve nome file in binario e array di interi
sottoprogramma legge i dati e li mette nell'array e dice quanti dati ci ha messo */
int conta_dati(char nome_file[], int v[]){
        int nval;
        FIL* fin;
        if(fin = fopen(nomef, "rb")){
                fread(&nval, sizeof(int), 1, fin);
                fread(v, sizeof(int), nval, fin)
                fclose(fin);
        }
        else{
                printf("problemi di accesso al file %s\n", nome f);
                nval = 0;
        }
```

```
        return nval;
}
```

```
/* programma che acquisisce valori di lunghezza ignota che termina con valore
inferiore a 1 file salva numeri primi nel file 'primi.csv'*/
/* file.csv è comma separeted value \\ file.ssv è space seoareted value*/

#define SOGLIA 1
#define FNAME "primi.csv"
int main(int argc, char *argv[]){
        FILE* fo;
        int val;
        fo = fopen(FNAME, "w")                /* "r" read, "w" write, "rb" read binario*/
        if(fo){
                scanf("%d", &val);
                while(val>=SOGLIA){
                        if(is_prime(val)){
                                fprintf(fo, "%d\n", val);        /* al posto di '\n' mettere ',' */
                        scanf("%d", &val);
                        }
                }
                fclose(fo);
        }
        else
                printf("problemi apertura file %s\n", FNAME);
        return 0;
}
```

5.5.8 Varibili globali

Per ora abbiamo usato solo programmi con variabili locali, che sono visibili solo nella singola funzione.
Per il passaggio ad altre funzioni si usa o passaggio per valore o per indirizzo.

Definition 5.2 (Variabili globali)**.** *:*
Sono variabili con visibilità potenziata che vengono dichiarate fuori dal main e fuori da qualsiasi sottoprogramma (vengono dichiarate dopo define, typedef e prototipi ma prima del main).

5.6 Algoritmi non immediati

Rotazione in senso orario di una matrice quadrata (indici array multidimensionale):

```
/* inserire dimensione array bidimensionale quadrato di dim_max(M)=10
crea e visulaizza matrice ruotatta in senso orario*/
#include <stdio.h>
#define MAX 10
int main(int argc, char*argv[]){
        int dim, i, j, mat[MAX][MAX], mat_r[MAX][MAX];
        do
                scanf("%d", &dim);
        while (!(dim >= 2 && dim <= 10));
        for(i=0; i<dim; i++){
                for(j=0; j<dim; j++)
                        scanf("%d", &mat[i][j]);
        }
        for(i=0; i<dim; i++){
                for(j=0; j<dim; j++)
```