

Unità Didattica 5

Linguaggio C

Stringhe. Accesso a file ASCII.
Strutture.

Stringhe

- Una stringa è una sequenza di caratteri (es: "ciao", "sd!n29dnqa");
- In C non esiste una variabile di tipo stringa;
- Una variabile stringa viene simulata mediante un vettore di `char` con l'ultimo carattere = `'\0'` (carattere terminatore (è un carattere solo));
- Es: la stringa "ciao" sarà rappresentata in memoria dalla sequenza "ciao\0" e contenuta in un vettore di `char` di dimensione 5 (= dimensione stringa + 1);
- Le stringhe costanti sono racchiuse fra apici "";
- Es: `char str[] = "ciao";`
- Il vettore di caratteri può essere trattato dalle funzioni che gestiscono le stringhe come un'unica variabile scalare contenente una stringa;

Stringhe: allocazioni statiche e dinamiche

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

void main(void)
{
    char str1[10]; //Allocazione statica di una stringa di al
                  //più 9 caratteri
    char *str2;

    str2 = (char *)malloc(sizeof(char) * 20);
                  //Allocazione dinamica di una stringa di al
                  //massimo 19 caratteri

    scanf("%s", str1); //Inserimento di stringhe dallo standard input
    scanf("%s", str2);

}
```

Composizione di stringhe

```
sprintf(stringa destinaz., "formato della stringa da comporre",  
       variabili dei valori componenti);
```

Es:

```
char nome[] = "Mario";  
char cognome[] = "Rossi";  
int eta = 45;  
char *titolare;  
  
titolare = (char *)malloc(sizeof(char)* 50);  
  
sprintf(titolare, "%s %s %d", nome, cognome, eta);  
        //In titolare sarà scritto  
        //"Mario Rossi 45"
```

Decomposizione di stringhe

```
sscanf(stringa sorg., "formato della stringa da decomporre",  
      indirizzi delle variabili destinazione dei valori  
      componenti);
```

Es:

```
char *titolare = "Maria Bianchi 52";  
char nome[10], cognome[20];  
int eta;  
  
sscanf(titolare, "%s %s %d", nome, cognome, &eta);  
  
//Nelle variabili nome, cognome e eta  
//verrà scritto: nome = "Maria"  
                cognome = "Bianchi"  
                eta = 52
```

Lunghezza di una stringa

```
(int) strlen(nome stringa);
```

- Restituisce come costante intera la lunghezza della stringa;

Es:

```
char *titolare = "Maria Bianchi 52";  
int lungh;
```

```
lungh = strlen(titolare);
```

```
printf("%d", lungh); //Viene visualizzata la lunghezza di  
                    //titolare eccetto '\0'  
                    //In questo caso lungh == 16
```

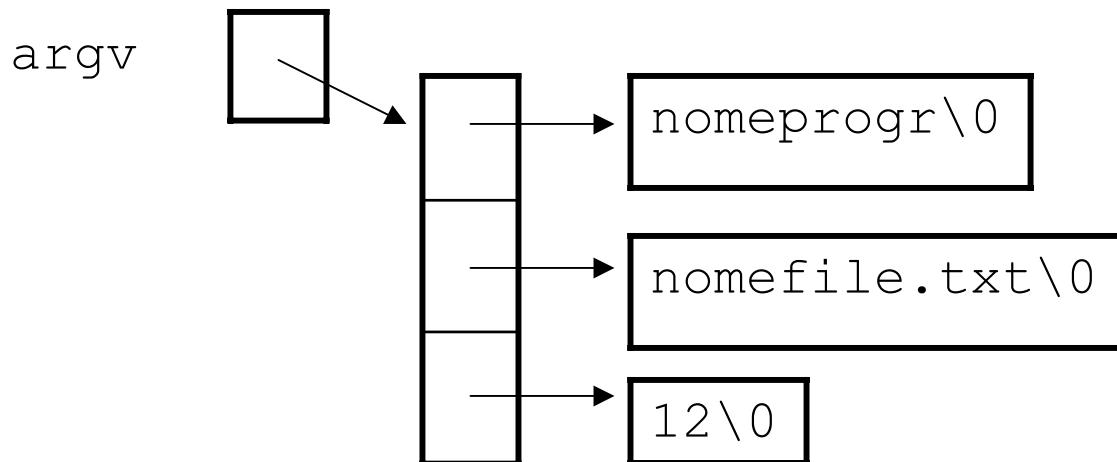
Passaggio di parametri al main

- Al momento della attivazione di un programma, dalla linea dei comandi, possono essere passati dei parametri al main;
- Dalla linea di comando vengono passate informazioni di tipo stringa;
- Le informazioni scritte sulla linea di comando vengono trasferite al programma in due variabili:
 - `char **argv` //vettore di stringhe
(ovvero `char *argv[]`)
 - `int argc` //dimensione del vettore `argv`

Passaggio di parametri al main

```
c:\>nomeprogr nomefile.txt 12
```

```
void main(int argc, char **argv)
{
}
}
```



Accesso a file ASCII

- L'accesso ad un file avviene mediante l'apertura del file stesso attraverso la funzione `fopen` della libreria `stdio.h`:

```
(FILE *)fopen(nome_file, "modalità di accesso");
```

- Restituisce un puntatore ad un tipo predefinito (`FILE`) per una variabile che rappresenterà il puntatore al file;
- Le modalità di accesso per i file ASCII sono espresse da stringhe:
 - `"r"`: lettura;
 - `"w"`: scrittura (se esistono vecchi contenuti vengono scartati);
 - `"a"`: aggiunta o append (scrittura in coda a eventuali vecchi contenuti);

Es:

```
FILE *fp;  
fp = fopen("miofile.txt", "r");
```

- Al termine delle operazioni il file deve essere chiuso con la funzione:

```
fclose(puntatore a file);
```

Lettura da file

```
fscanf(puntatore al file, "formato dati",  
       indirizzi delle variabili destinazione);
```

```
Es:      miofile.txt      ->      |Mario Gino Vittorio  
                                           |Andrea Maria Giovanna
```

```
FILE *fp;  
char nome[10][50];  
int i = 0;  
fp = fopen("miofile.txt", "r");  
  
while (!feof(fp))  
{  
    fscanf(fp, "%s", nome[i]);  
    i++;  
}  
fclose(fp);
```

Scrittura in un file

```
fprintf(puntatore al file, "formato dati",  
variabili il cui contenuto e' da scrivere sul file);
```

Es:

```
char *nome[] = {"Mario", "Gino", "Vittorio"};  
FILE *fp;  
int i;  
  
fp = fopen("miofile.txt", "w");  
i = 0;  
while (i < 3)  
{  
    fprintf(fp, "%s\n", nome[i]);  
    i++;  
  
}  
fclose(fp);
```

Strutture (Record)

- Strutture dati contenenti elementi di tipo diverso;
- In altri linguaggi vengono chiamate record;
- Consentono di trattare come un unico oggetto variabili complesse e correlate fra di loro;
- La definizione di una nuova struttura determina la creazione di un nuovo tipo di dato (*tipo derivato*);
- In C un record (struttura) si ottiene dichiarando una variabile del nuovo tipo creato;
- Tutte le proprietà e le regole sintattiche associate alle variabili scalari e vettoriali dei tipi standard si estendono anche alle variabili scalari e vettoriali dei tipi derivati.

Accesso ai campi di una struttura

```
struct studente
{
    char nome[30];
    char cognome[30];
    int voto;
};

void main()
{
    struct studente stud_ing;

    scanf("%s", stud_ing.nome);
    scanf("%s", stud_ing.cognome);
    scanf("%d", &(stud_ing.voto));
    ...
    printf("%s", stud_ing.nome);
    printf("%s", stud_ing.cognome);
    printf("%d", stud_ing.voto);
    ...
}
```

Direttiva typedef

- La direttiva `typedef` consente di creare degli alias (sinonimi) per qualunque tipo di dato;

```
typedef tipo tiponuevo;
```

- Esempi:

```
typedef float temperatura;  
temperatura y; //Variabile di tipo "temperatura"
```

```
typedef struct studente s_stud;  
stud stud_ing; //Variabile di tipo "s_stud"
```

Passaggio di una struttura a funzione

```
main()
{
    struct studente stud_ing;
    ...
    funz(stud_ing);
}

void funz(struct studente stud)
{
    ...
    printf("%s\n", stud.nome);
    printf("%s\n", stud.cognome);
    printf("%d\n", stud.voto);
}
```

Vettori di Struttura

- Sono vettori i cui elementi sono strutture di un determinato tipo;

- Es:

```
struct studente
{
    char nome[30];
    char cognome[30];
    int voto;
};
struct studente stud_ing[100];
```

```
//allocazione statica del vettore "stud_ing" di
//100 "struct studente"
```

Dichiarazione di un vettore di strutture con uso di typedef

- **Es:**

```
//Creazione del sinonimo "s_stud" per  
//"struct studente"
```

```
typedef      struct studente      s_stud;
```

```
//Allocazione statica del vettore "stud_ing"  
//di tipo "s_stud", sinonimo di "struct studente"
```

```
s_stud stud_ing[100];
```

Accesso ai campi in un vettore di strutture

```
//Acquisizione dei dati
for (i = 0; i < 100; i++)
{
    scanf("%s", stud_ing[i].nome);
    scanf("%s", stud_ing[i].cognome);
    scanf("%d", &(stud_ing[i].voto));
}

//Visualizzazione dei dati
for (i = 0; i < 100; i++)
{
    printf("%s", stud_ing[i].nome);
    printf("%s", stud_ing[i].cognome);
    printf("%d", stud_ing[i].voto);
}
```

Allocazione dinamica di un vettore di strutture

```
struct studente *stud_ing; //puntatore al tipo derivato
                           // "struct studente"

int dim;

scanf("%d", &dim); //Acquisizione da tastiera della
                  //dimensione della struttura dati
                  //da allocare

//Allocazione dinamica del vettore "stud_ing" di "dim"
//elementi di tipo "struct studente"
stud_ing =
    (struct studente*)malloc(sizeof(struct studente)*dim);
```

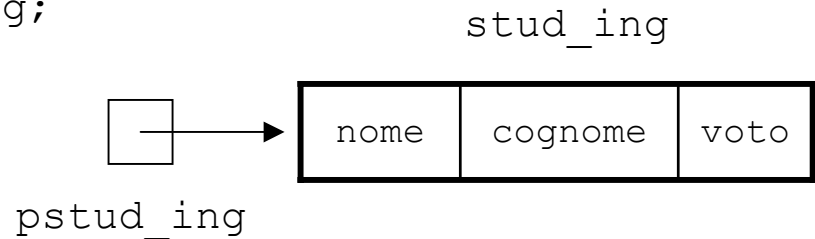
Puntatori a struttura

- Variabili che contengono l'indirizzo di una variabile strutturata (record);

```
//dichiarazione e allocazione statica della struttura  
//(record) "stud_ing"  
    struct studente stud_ing;
```

```
//dichiarazione di un puntatore ad una variabile di tipo  
//"struct studente"  
    struct studente *pstud_ing;
```

```
pstud_ing = &stud_ing;
```

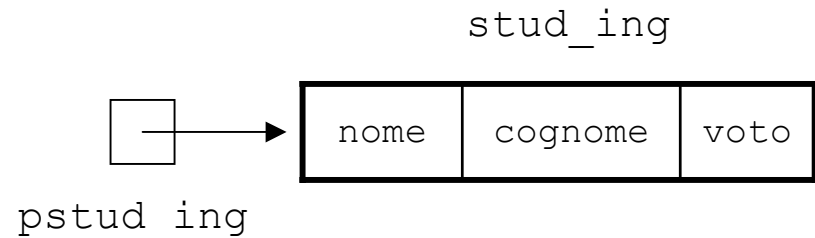


Accesso ai campi di una struttura tramite puntatori

- Valgono le stesse regole per l'accesso alle variabili scalari tramite puntatori;

```
struct studente stud_ing;  
struct studente *pstud_ing;
```

```
pstud_ing = &stud_ing;
```



```
//Accesso al campo "voto" della struttura "stud_ing"
```

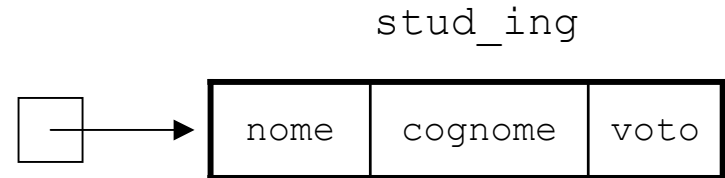
```
stud_ing.voto  $\Leftrightarrow$  (*pstud_ing).voto  $\Leftrightarrow$  pstud_ing->voto
```

Sintassi alternativa e usatissima per l'accesso ai campi di una struttura, riferita mediante un puntatore

Esempi di accesso ai campi di una struttura tramite puntatori

```
struct studente stud_ing;  
struct studente *pstud_ing;
```

```
pstud_ing = &stud_ing;
```



```
                                pstud_ing  
//Acquisizione dei valori dei campi cognome e voto  
scanf("%s %d", stud_ing.cognome, &stud_ing.voto);  
    //oppure  
scanf("%s %d", (*pstud_ing).cognome, &(*pstud_ing).voto);  
    //oppure  
scanf("%s %d", pstud_ing->cognome, &pstud_ing->voto);  
  
//Visualizzazione del contenuto del campo "cognome" e "voto"  
printf("%s %d\n", stud_ing.cognome, stud_ing.voto);  
    //oppure  
printf("%s %d\n", (*pstud_ing).cognome, (*pstud_ing).voto);  
    //oppure  
printf("%s %d\n", pstud_ing->cognome, pstud_ing->voto);
```

Passaggio di un vettore di strutture a funzione

```
void main(void)
{
    int i, dim;
    struct studente *pstud_ing;
    scanf("%d", &dim);
    pstud_ing = (struct studente*)malloc(sizeof(struct studente)*dim);
    for (i = 0; i < dim; i++)
    {
        scanf("%s", pstud_ing[i].nome);
        scanf("%s", pstud_ing[i].cognome);
        scanf("%d", &pstud_ing[i].voto);
    }
    visualizza_vect(pstud_ing, dim);
    ...
}

void visualizza_vect(struct studente *pstud, int dim)
{
    for (i = 0; i < dim; i++)
    {
        printf("%s %s %d\n", pstud[i].nome, pstud[i].cognome,
                pstud[i].voto);
    }
}
```