Laboratorio II

Corso A

Lezione 11

- Puntatori a funzioni
- Genericità attraverso l’uso del tipo `void*`
- Manipolazione e gestione di stringhe

Puntatori a funzioni

- Le funzioni in C non sono delle variabili
  - Possiamo ottenere dei puntatori a funzioni definite nel programma o in librerie
- I puntatori possono poi essere passati come parametro o utilizzati per invocare la funzione.

```c
#include <stdio.h>

int sum(int, int);
int mul(int,int);

int main(){

  int (*f)(int,int);

  f=sum;
  printf("3+6=%d\n", (*f)(3,6));

  f=mul;
  printf("3*6=%d\n", (*f)(3,6));
}

int sum(int x, int y){
  return x+y;
}

int mul(int x, int y){
  return x*y;
}
```

Puntatori a funzioni

Possiamo avere degli array di puntatori a funzioni.

```c
#include <stdio.h>

int sum(int, int);
int mul(int,int);

int main(){

  int (*f[2])(int,int)={sum,mul};

  printf("3+6=%d\n", (*f[0])(3,6));

  printf("3*6=%d\n", (*f[1])(3,6));
}

int sum(int x, int y){
  return x+y;
}

int mul(int x, int y){
  return x*y;
}
```

Funzione qsort: uso di `void*`

```
int compInt(const void* a, const void * b){
  int* ac=(int*)a;
  int* bc=(int*)b;
  return *ac-*bc;

}
```

In `stdlib.h`

```
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

- `base` - array
- `num` - numero di elementi
- `size` - dimensione di 1 elemento
- `compar` - puntatore a funzione per confronto

Esempio di genericità usando `void*`

```
› ./main
0 2 9 17 43 ›
```

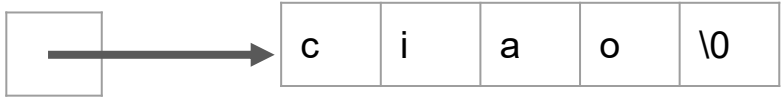
```
int compInt(const void*, const void *);

int main(){

  int a[5]={17,9,2,43,0};

  qsort(a,5,sizeof(int),compInt);

  for(int i=0;i<5;i++){
    printf("%d ",a[i]);
  }
  return 0;
}
```

Stringhe

Array di caratteri, con ‘\0’ alla fine (carattere null, con valore 0)

char s[]=”ciao”; ->

char* s=”ciao”; ->




```c
#include <stdio.h>

int main(){

  char s1[]="ciao";

  s1[2]='A';

  printf("s1=%s\n",s1);

}
```

```
> ./main
s1=ciAo
```

```c
#include <stdio.h>

int main(){

  char* s1="ciao";

  s1[2]='A';

  printf("s1=%s\n",s1);

}
```

```
> ./main
signal: segmentation fault (core dumped)
```

Stringhe

Lavorare con stringhe è analogo a lavorare con gli array.

E.g. ottenere una copia di una stringa

```c
#include <stdio.h>

int main(){

  char s1[]="ciao";
  char* s2=s1;

  s2[2]='A';

  printf("s1=%s\n",s1);
  printf("s2=%s\n",s1);

}
```

```
> ./main
s1=ciAo
s2=ciAo
```

```
> ./main
s1=ciao
s2=ciAo
```

```c
int main(){

  char s1[]="ciao";
  char s2[5];

  int i=0;
  while(s1[i]!='\0'){
    s2[i]=s1[i];
    i++;
  }
  s2[i]='\0';

  s2[2]='A';

  printf("s1=%s\n",s1);
  printf("s2=%s\n",s2);

}
```

Operazioni su stringhe

Libreria string.h include funzioni per lavorare con stringhe (qui esempi, la lista non è completa)

- `size_t strlen(const char * str)` - restituisce la lunghezza di una stringa
- `char* strcpy(char * dest, const char * src)` - copia la stringa `src` in `dest`. Restituisce `dest`.
- `int strcmp(const char * str1, const char * str2)` - confronto lessicografico, restituisce -1, 0, +1
- `char* strcat(char * dest, const char * src)` - concatena una copia di `src` alla `dest`. Restituisce `dest`.
  - Le varianti `strncpy, strncmp, strncat` - lavorano sui primi `n` caratteri

- `char* strtok(char * str , const char * delim)`

Operazioni su stringhe

`char* strtok(char * str , const char * delim)` - restituisce il pointer al prossimo token

```c
#include <stdio.h>
#include <string.h>

int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);

  tok=strtok(s,",");
  printf("%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL)
    printf("%s\n",tok);

}
```

```
> ./main
12,13,14,15
12
13
14
15
```

strtok - lavora sulla stringa originale

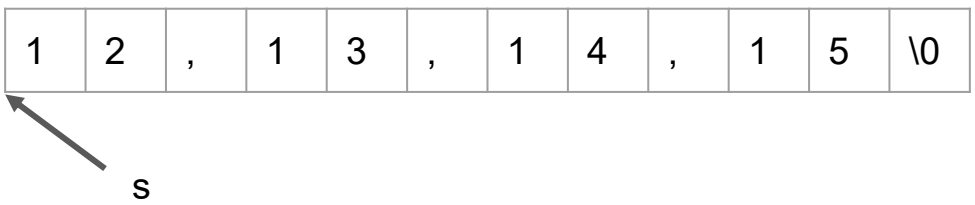
```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```

strtok - lavora sulla stringa originale

```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

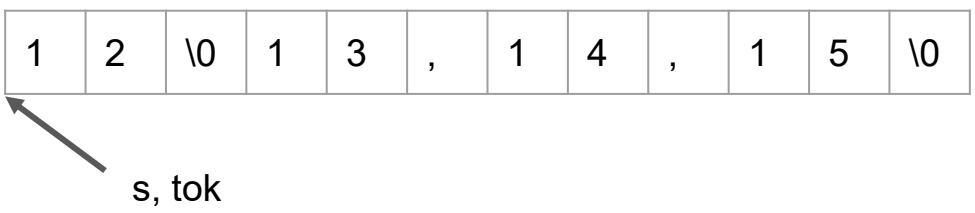
```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```



strtok - lavora sulla stringa originale

```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```



strtok - lavora sulla stringa originale

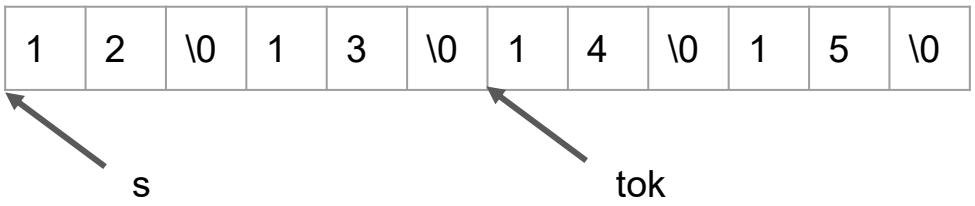
```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```

strtok - lavora sulla stringa originale

```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

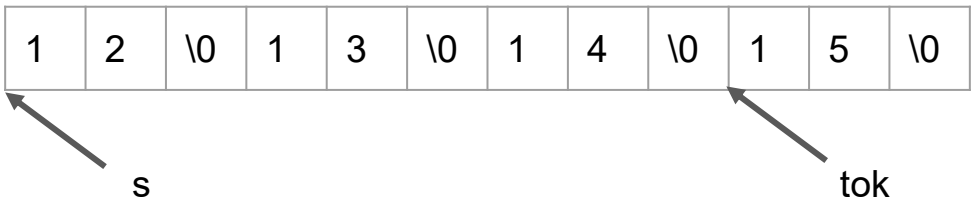
```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```



strtok - lavora sulla stringa originale

```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

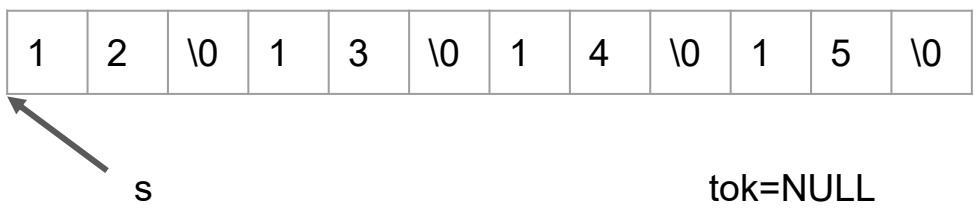
```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```



strtok - lavora sulla stringa originale

```c
int main(){

  char s[100], *tok;
  scanf("%[^\n]",s);
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

```
> ./main
12,13,14,15
Stringa:12,13,14,15
Token:12
Token:13
Token:14
Token:15
Stringa:12
>
```



strtok - lavora sulla stringa originale

```c
int main(){

  char *s="12,13,14,15", *tok;
  printf("Stringa:%s\n",s);

  tok=strtok(s,",");
  printf("Token:%s\n",tok);

  while((tok=strtok(NULL,","))!=NULL){
    printf("Token:%s\n",tok);
  }

  printf("Stringa:%s\n",s);
}
```

```
> ./main
Stringa:12,13,14,15
signal: segmentation fault (core dumped)
```

Operazioni su caratteri

Libreria `ctype.h` include funzioni per lavorare con caratteri (qui esempi)

`int isalpha(int)` !=0 se parametro è una lettera

`int isupper(int)` !=0 se parametro è una lettera maiuscola

`int islower(int)` !=0 se parametro è una lettera minuscola

`int isdigit(int)` !=0 se parametro è una cifra

`int isalnum(int)` !=0 se parametro è cifra o lettera

`int isspace(int)` !=0 se parametro è white space

`int toupper(int)` trasforma in lettera maiuscola se possibile, altrimenti restituisce lo stesso carattere

`int tolower(int)` trasforma in lettera minuscola se possibile, altrimenti restituisce lo stesso carattere

Leggere caratteri e stringhe

`int scanf ( const char * format, ... );` - `“%c”,“%s”, “%[^\n]”, “%[^EOF]”`

`int getchar ( void );` Legge un carattere da stdin

`char * fgets ( char * str, int num, FILE * stream );` Legge una riga di max num caratteri da ***stream***. Utilizzare `stdin` per leggere da standard input.

Da stringa ad altri tipi di dato

```
int sscanf ( const char * s, const char * format, ...);
```

Legge dalla stringa `s` invece che da `stdin`

```
int atoi(const char * str), long atol(const char * str),
```

```
float atof(const char * str), double atod(const char * str)
```

Trasformano una stringa in un `intero/long/float/double`

Domande?