Introduzione al C
Lez. 4

Allocazione
Dinamica della memoria

Allocazione dinamica memoria

Finora abbiamo fatto uso esclusivamente di oggetti la cui dimensione è una costante specificata nel codice:

```
char  buffer[ 500 ];
```

In molte circostanze è impossibile prevedere la quantità di memoria usata a tempo di esecuzione:

- Input/Output: lettura di file di dimensione arbitraria
- Gestione strutture dati che crescono e decrescono durante l'esecuzione del programma

...

In tali casi occorre un meccanismo che consenta di gestire dinamicamente l'allocazione della memoria

Allocazione dinamica memoria

La libreria standard del C mette a disposizione diverse funzioni come **`malloc, calloc`** e **`free`** per allocare/deallocare blocchi di memoria la cui dimensione è specificata in input.

Le dichiarazioni si trovano nel file `stdlib.h`, includetelo con la riga:

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

Malloc

Definita in `stdlib.h`. Includetelo!

```
#include <stdlib.h>

                void * malloc(size_t size)
```

- dove `size` specifica la lunghezza in byte del blocco allocato. `size_t` è un tipo definito in `stdlib.h`. (In genere è un `unsigned int`.)

Malloc

Definita in `stdlib.h`. Includetelo!

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

```
void * malloc(size_t size)
```

- dove `size` specifica la lunghezza in byte del blocco allocato. `size_t` è un tipo definito in `stdlib.h`. (In genere è un `unsigned int`.)

- La funzione restituisce un `void*` (puntatore generico) contenente l'indirizzo del primo byte del blocco al primo byte del blocco. Un puntatore void * può essere convertito a qualsiasi tipo di puntatore

```
int *p = malloc( 40 )
```

Malloc

Definita in `stdlib.h`. Includetelo!

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

```
void * malloc(size_t size)
```

- dove `size` specifica la lunghezza in byte del blocco allocato. `size_t` è un tipo definito in `stdlib.h`. (In genere è un `unsigned int`.)

- La funzione restituisce un `void*` (puntatore generico) contenente l'indirizzo del primo byte del blocco al primo byte del blocco. Un puntatore void * può essere convertito a qualsiasi tipo di puntatore

```
int *p = malloc( 40 )
```

- Se non è possibile allocare memoria (ad es. è esaurita), la funzione restituisce il puntatore `NULL`.

Malloc
Esempio

```
#include <stdlib.h>

...

int i, n, *p;
scanf(“%d”, &n);
p = malloc(n * sizeof(int)); //alloca spazio per n interi

if (p == NULL) { // controlliamo
      printf(“Non posso allocare %d interi\n”,n);
      exit(1); // esce con un errore
}
```

Malloc
Esempio

```
#include <stdlib.h>

...

int i, n, *p;
scanf(“%d”, &n);
p = malloc(n * sizeof(int)); //alloca spazio per n interi

if (p == NULL) { // controlliamo
      printf(“Non posso allocare %d interi\n”,n);
      exit(1); // esce con un errore
}
… // se sono arrivato qui posso usare il blocco allocato...

for(i = 0; i < n; i++) // qui mettiamo i primi n interi
   p[i] = i;
```

Calloc

Definita in `stdlib.h`. Includetelo!

```
void * calloc(size_t nmemb, size_t size)
```

- simile alla malloc ma tutti i bytes del blocco contengono 0.

- `nmemb` indica il numero di elementi nell'array mentre `size` specifica la lunghezza in bytes di ciascun elemento.

Liberare la memoria

Quando un blocco di memoria non serve più è importante deallocarlo e renderlo nuovamente disponibile utilizzando la funzione

```
void free(void *)
```

L'argomento di free deve essere allocato precedentemente (o `NULL`) altrimenti il comportamento è indefinito.

Esercizio 1

Input: Leggere da tastiera una sequenza di interi. Il primo di questi indica quanti interi compongono la sequenza, il resto sono immessi in un array allocato dinamicamente con malloc

Output: `m x\n`
dove m è la media degli interi e x è il numero di interi uguali a `m`.

Esempio

4
7
7
4
10

```
7 2
```

Esercizio 2

Implementare una funzione

```
void* my_calloc(size_t size);
```

che simuli il comportamento della calloc eseguendo i seguenti tre passi:

- alloca un blocco di dimensione size bytes utilizzando la funzione malloc

- azzera il contenuto di ogni byte del blocco

- ritorna un puntatore void* al blocco

Esercizio 3 (passaggio per riferimento)

Modificare la funzione dell'esercizio 1 lez.3

```
void minmax(int a[], int len, int *min, int *max);
```

In modo tale che, al ritorno dalla funzione, le celle puntate da min / max contengano l'indice del minimo /massimo elemento dell'array a