Liste e Ripasso

Sesto Laboratorio

Strutture e Liste

Rivediamo le Strutture

```
typedef struct anagrafico{
  char nome[20];
  int anni;
  char indirizzo[30];
} dati;

cittadino.anni = 3;

dati popolazione[1000];
```

ESERCIZIO

- Scrivere un programma che
  - definisca la struttura persona contententi i campi: nome, altezza, peso;
  - crei un array di 3 “persone”;
  - crei una funzione che riempie l’array chiedendo all’utente di inserire da tastiera nome, altezza e peso per le tre persone;
  - crei una funzione che stampa in output il contenuto dell’array;

SOLUZIONE.

```
typedef struct persona{
  char nome[20];
  int altezza;
  int peso;
} dati;

#define N 3

void inserisci(dati a[], int lung);
void stampa(dati a[], int lung);
```

SOLUZIONE..

```
int main(){
  dati persone[N];
  inserisci(persone, N);
  stampa(persone, N);
}
```

SOLUZIONE…

```
void inserisci(dati a[], int lung){
  int i = 0;
  for( ; i < lung; i++){
     printf("Inserire nome, altezza e
  peso");
     scanf("%s", a[i].nome);
     scanf(" %d", &a[i].altezza);
     scanf(" %d", &a[i].peso);
  }
}
```

SOLUZIONE....

```
void stampa(dati a[], int lung){
  int i = 0;
  for( ; i < lung; i++){
      printf("nome = %s ", a[i].nome);
      printf("altezza = %d ", a[i].altezza);
      printf("peso = %d ", a[i].peso);
  }
}
```

ESERCIZIO

- Descrivere la struttura triangolo, che usa la struttura punto: descrive un punto nel piano cartesiano.
- Costruire una funzione che calcola la distanza euclidea fra due punti.
- Scrivere poi una funzione che calcola il perimentro di un triangolo
- Scrivere un main di esempio che illustri le funzioni.

SOLUZIONE.

```
typedef struct {
  float x;
  float y;
} punto;


typedef struct {
  punto t;
  punto bl;
  punto br;
} triangolo;
```

SOLUZIONE..

```
float distanzaEuclidea(punto a, punto b)
{
  return sqrt(pow((a.x - b.x), 2) +
  pow((a.y - b.y), 2));
}


float perimetro(triangolo t) {
  return distanzaEuclidea(t.t, t.bl) +
  distanzaEuclidea(t.t, t.br) +
      distanzaEuclidea(t.br, t.bl);
}
```

SOLUZIONE...

```
main() {
  punto a = {0, 0};
  punto b = {1, 1};
  printf("distanza fra a e b: %f \n",
  distanzaEuclidea(a, b));

  punto t = {1, 2};
  punto bl = {0, 0};
  punto br = {2, 0};
  triangolo tr = {t, bl, br};
  printf("il perimetro del triangolo e`: %f
  \n", perimetro(tr));
}
```

PROBLEMA?!?!

- E se io volessi memorizzare un numero indefinito di elementi?
- Con un array statico questo non è possibile...
- Sappiamo che dobbiamo definire la sua dimensione all'inizio e non possiamo cambiarla...
- Quindi, o definiamo un array con un numero grandissimo di elementi, ma non possiamo avere la certezza che questi bastino e rischiamo di occupare tanta memoria in più...
- ... oppure dobbiamo usare un sistema dinamico per la memorizzazione dei dati

IDEA

- E’ necessario avere un metodo che permetta l’allocazione dinamica della memoria per inserire una quantità di elementi non nota quando si scrive il programma;

- Alcune soluzioni a questo problema sono
  - array creati dinamicamente;
  - liste di elementi.

FUNZIONE MALLOC

- Dal manuale di sistema (`man 3 malloc`):

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

malloc allocates size bytes and returns a pointer to the allocated memory. The memory is not cleared. If size is 0, then malloc returns either NULL, or a unique pointer value that can later be successfully passed to free.

- Includere la libreria

```
#include<malloc.h>
```

FUNZIONE FREE

- Dal manuale di sistema:

```
void free(void *ptr);
```

free() frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc(). Otherwise, or if free(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed.

FUNZIONE REALLOC

- Dal manuale di sistema:

```
void *realloc(void *ptr, size_t size);
```

realloc changes the size of the memory block pointed to by ptr to size bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If ptr is NULL, then the call is equivalent to malloc(size), for all values of size; if size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr). Unless ptr is NULL, it must have been returned by an earlier call to malloc(), calloc() or realloc(). If the area pointed to was moved, a free(ptr) is done.

OPERATORE SIZEOF

- Operatore unario (non una funzione) che restituisce il numero di byte del tipo dell'argomento (nome di tipo, variabile o espressione)
- Utile, quando abbinato a malloc o realloc

- Esempio:

```
char c;
printf("%d", sizeof(int));
printf("%d", sizeof(c));
```

ARRAY DINAMICI

- Esempio di allocazione di un array:

```
int dim = 5; // e' una variabile…
int* v;

int memoria_da_occupare = sizeof(int) * dim;

v = malloc(memoria_da_occupare);
```

ESERCIZIO

- Scrivere un programma che costruisce un array di interi della dimensione richiesta all'utente; lo si popoli e se ne stampino i valori;
- Per il popolamento e la stampa del vettore definire una apposita funzione;
- Attenzione ai dettagli (dimensione strettamente positiva, free, …)

SOLUZIONE.

```
#include<stdio.h>
#include<malloc.h>
. . .
main() {
   int dim;
   int* v;

   do {
         printf("Inserisci la dimensione dell'array: ");
         scanf("%d", &dim);
   } while (dim <= 0);

   int memoria_da_occupare = sizeof(int) * dim;
   v = malloc(memoria_da_occupare);
   if (v == NULL) {printf("Errore irreversibile!\n"); return;}

   popolaVettore(v, dim);
   stampaVettore(v, dim);

   free(v);
}
```

SOLUZIONE..

```
void stampaVettore(int* v, int dim) {
  printf("\nint* v = { \n  ");
  int i;
  for(i = 0; i < dim; i++) {
     printf("%d, ", v[i]);
  }
  printf("\n}\n");
}
```

SOLUZIONE...

```
void popolaVettore(int* v, int dim) {
  int i;
  for(i = 0; i < dim; i++) {
     printf("v[%d] = ", i);
     scanf("%d", v+i); // oppure &v[i]
  }
}
```

ESERCIZIO

- Scrivere un programma che continua a chiedere interi fino a quando non viene inserito il valore 0 e, dopo li stampa tutti.
- I dati dovranno essere salvati all'interno di un array dinamico

SOLUZIONE.

```
. . .
main() {

  int dim = 0;
  int* v = NULL;

  dim = riempiVettore(v);
  stampaVettore(v, dim);

  free(v);
}
```

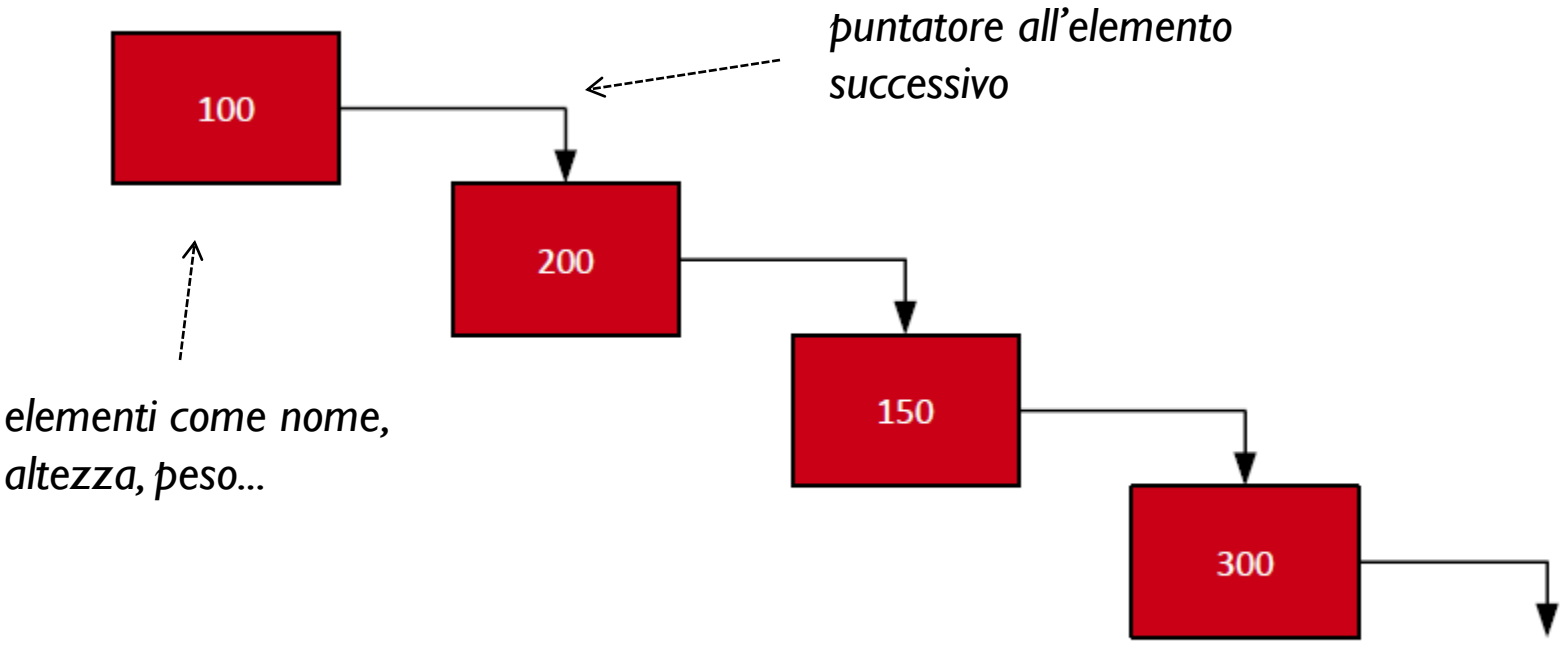
SOLUZIONE..

```
int riempiVettore(int* v){
   int dim = 0;
   int memoria_da_occupare = sizeof(int) * dim;
   int input;

   do {
         printf("Inserisci il valore da aggiungere: ");
         scanf("%d", &input);
         if (input != 0) {
                  dim++;
                  // dobbiamo riallocare il vettore
                  memoria_da_occupare = dim * sizeof(int);
                  v = (int*)realloc(v, memoria_da_occupare);
                  if (v == NULL) {printf("Errore irreversibile!\n");return;}
                  v[dim - 1] = input;
         }
   } while(input != 0);
   return dim;
}
```

LISTA

- Le liste sono una soluzione al problema



Definizione Lista

- Definizione del nodo

```
typedef struct nodo {
  int value;
  struct nodo * next;
} nodo
```

- Definizione della lista

```
typedef nodo * lista;
```

CREAZIONE LISTA

- Costruire alcuni elementi

```
nodo* n1 = (nodo*)malloc(sizeof(nodo));
nodo* n2 = (nodo*)malloc(sizeof(nodo));
nodo* n3 = (nodo*)malloc(sizeof(nodo));
```

- Popolarli

```
n1->value = 100;
n1->next = n2;
n2->value = 200;
n2->next = n3;
n3->value = 150;
n3->next = NULL;
```

- Creare la lista

```
lista l = n1;
```

Scorrimento e pulizia

- Stamparne i valori

```
nodo * n = n1;
do {
  printf("%d ", n->value);
  n = n->next;
} while(n != NULL);
```

- Pulire la memoria

```
free(n1);
free(n2);
free(n3);
```

ESERCIZIO

- Scrivere un programma che continua a chiedere interi fino a quando non viene inserito il valore 0 e, dopo li stampa tutti.
- I dati dovranno essere salvati all'interno di una lista