Esercitazioni di Fondamenti di Informatica - Lez. 7
28/11/2019

Esercizi sull'allocazione dinamica della memoria in C

una interessante prova per verificare quanti byte vengono allocati per i TIPI è la seguente.

Si nota che le allocazioni di STR UCT non sono esattamene la somma dei suoi componenti poichè il compilatore alloca indirizzi per gruppi di 4 byte

```c
#include <stdio.h>

int main()
{
    char c;
    int i;
    unsigned u;
    float f;
    double d;
          void * p;

          printf("sizeof(char) %d\n", sizeof(c));
          printf("sizeof(int) %d\n", sizeof(i));
          printf("sizeof(unsigned) %d\n", sizeof(u));
          printf("sizeof(float) %d\n", sizeof(f));
          printf("sizeof(double) %d\n", sizeof(d));
          printf("sizeof(p) %d\n", sizeof(p));

          return 0;
      }
```

---

esempio di allocazione di memoria per ottimizzare la dimensione degli array: MAX lo conosciamo solo runtime

```c
#define MAX 1000
int main () {
        int  a[MAX];


                printf("Quanti valori?");
                scanf("%d", &n);
        for(i=0;i<n; i++)    {
                printf("Elemento %d: ", i+1);
                scanf("%d",  &a[i]) ;
        }
```

esempio di malloc per allocare solo quello che serve runtime

```
int *a, n;


a=(int *) malloc(sizeof(int));
printf("Quanti valori?" );
scanf("%d", &n);
for(i=0;i<n; i++)  {
        printf("Elemento %d: ", i+1);
        scanf("%d",  &a[i] ) ;
}
```

---

1. **Cosa stampa il seguente programma?**

```
#include<stdio.h>
#define MAXLENGTH 30
typedef struct {
       char nome[MAXLENGTH];
       char cognome[MAXLENGTH];
} Persona;

void stampa_persona(Persona p) {        //1 passaggio per valore
       printf("%s %s\n", p.nome, p.cognome);
}

void modifica_persona(Persona r) {     //1 modifica il valore localmente
       r.nome[0] = '\0';
}

//void modifica_persona(Persona *r){  //2 modifica per indirizzo passo il puntatore
//r->nome[0] = '\0';
//}

int main() {
       Persona p = { "Mario", "Rossi" };
       stampa_persona(p);
       modifica_persona(p);
//     modifica_persona(&p);            // 2 passaggio per indirizzo
       stampa_persona(p);            // 1 stampa la p senza modifiche

}
```

- La struttura p viene passata alla funzione modifica_persona per **valore**
- le modifiche vengono effettuale sulla variabile **locale** interna alla funzione
- In questo modo, la chiamata a modifica persona non ha **nessun effetto** sulla variabile p contenuta nel main.
- eliminando i commenti dalle righe con //2 otteniamo la versione che usa il passaggio per indirizzo(puntatore)

2. Cosa stampa il seguente programma?

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAXLENGTH 30

typedef struct{
	char * nome;
	char * cognome;
} Persona;

void alloca_persona(Persona * p, char nome[], char cognome[]){

	/* alloco in modo dinamico gli array che contengono  nome e cognome */
//  nome  avrà misura 30* size di char
```

0 8	9 15	16 31	..						

```
// cognome 30 char

	p->nome =(char *) malloc(MAXLENGTH * sizeof(char));
	p->cognome = (char *) malloc(MAXLENGTH * sizeof(char));
	strcpy(p->nome,nome);
	strcpy(p->cognome,cognome);
}

void stampa_persona(Persona p){

	printf("%s %s\n",p.nome,p.cognome);
}

void modifica_persona(Persona p){
	p.nome[0] = '\0';
	/* N.B. in questo caso la struttura contiene dei puntatori
	ad una zona di memoria dinamica. Quando la funzione
	viene richiamata si effettua una copia di ogni puntatore per i rispettivi attributi,
	ma non dell'area di memoria allocata dinamicamente a cui essi puntano */
}

int main(){

	Persona p;
	alloca_persona(&p,"Mario","Rossi");
	stampa_persona(p);
	modifica_persona(p);
	stampa_persona(p);
}
```

in questo caso il codice stamperá prima "Mario Rossi" e dopo " Rossi". Infatti, passando per riferimento l'indirizzo della variabile p del main alla funzione modifica persona, questa é in grado di modificare il contenuto di p appartenente al record di attivazione del main.

3. Che cosa stampa il seguente codice? Rimane della memoria non deallocata al termine del main?

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

```
#include<stdlib.h>


int main() {

	int* p;
	int* m;

	p = (int*)malloc(sizeof(int));  /* alloco lo spazio di un intero */
	*p = 10;
	m = p;
	*m = (*p)++;
	/*
		int temp = *p; *p += 1;
		*m = temp;
	*/

	printf("%d\n", *m); /* stampa 10 */

	*m = ++(*p);
	/*
	*p += 1;
	*m = *p;
	*/

	printf("%d\n", *m); /* stampa 11 */
	free(m);     }
```

la free() libera lo spazio indirizzato da m , siccome anche p punta alla stessa zona --> non rimane spazio allocato si usi il debug per fare la verifica

4. Cosa stampa questo codice? Rimane della memoria non deallocata al termine del main?

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

int main(){

    int i = 10, j = 20;
    int *p;
    p = (int*) malloc(sizeof(int)); /* alloco lo spazio di un intero*/
    *p = i;
    for(;*p<j;(*p)++){
      printf("%d\n",i);
    }
    p = &i;

    for(;*p<j;(*p)++){
      printf("%d\n",i); /* stampo i numeri da 10 a 19 */
    }
    free(p);
}
```

il programma termina con un errore perche' tenta di deallocare una zona non allocata in modo dinamico.
Infatti in riga 16 viene assegnato a p l'indirizzo della variabile i

➔ in quanto non gestito dinamicamente la memoria non viene deallocata in modo corretto.

5. Analizza il codice e verifica cosa stampa ?
verificare quale variabile corretta per deallocare la memoria free(?)

```
// ex5 lez 7
#include<stdio.h>
#include<stdlib.h>
#define MAX 30

int main() {

	char *p, *k;

	p = (char*)malloc(MAX * sizeof(char));
	/* alloco uno spazio di memoria grande 30 volte un char,
	in pratica un array di char contente 30 elementi
	char p[30];
	*/

	/* leggo una stringa di testo e la salvo nella zona di memoria puntata da p */
	scanf("%s", p);

	k = p;
	printf("%s\n", k);
	/* dato che k punta alla zona di memoria allocata
	dinamicamente, stampo la stringa letta in input */


	for (;*k != '\0';k++) {  /* uso il puntatore k per iterare lungo l'array */
		printf("%c", *k);
		/* stampo uno alla volta tutti i caratteri della
		   stringa letta in input */
	}

	printf("\n");
	free(?);
	/* libero la memoria

	*/
}
```

nota: free(k), genera un errore

6. Cosa stampa questo codice?

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

```c
#define MAX 5

int main() {

	int* m[MAX]; /* dichiaro un array di puntatori ad intero*/
	/* N.B. -> ogni cella dell'array e' un puntatore */

	int i, j;

	for (i = 0;i < MAX;i++) {

		m[i] = (int*)malloc(MAX * sizeof(int));

		/* Alloco un'area di memoria
			con l'equivalente di un array di interi di dimensione MAX e
			restituisco il puntatore ad una cella dell'array m. */

			/* alla fine del ciclo ottengo una matrice MAX x MAX*/

		for (j = 0;j < MAX;j++) {
			m[i][j] = (i + 1) * (j + 1);
		/* per ogni cella di indice(i,j) della matrice, calcolo (i+1) * (j+1).
			Costruisco in pratica una tavola pitagorica.
		*/
		}
	}

	for (i = 0;i < MAX;i++) {
		for (j = 0;j < MAX;j++) {
			printf("%d ", m[i][j]); /* stampo ogni cella della tavola*/
		}
		printf("\n");
		free(m[i]); /* devo deallocare ogni array creato dinamicamente*/
	}

}
```

7. Cosa stampa questo codice?

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

#define MAX 5

int main(){

	int **m; /* puntatore a puntatori */
	int i,j;
```

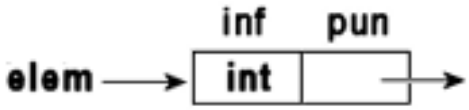
```
	/* N.B. A differenza dell'esercizio 6, la matrice e' completamente dinamica,
sia per il numero di righe che di colonne */

	m = (int**) malloc(sizeof(int*) * MAX);

	for(i=0; i<MAX;i++){

		m[i] = (int*) malloc(MAX * sizeof(int));
		for(j=0;j<MAX;j++){
			m[i][j] = (i+1) *(j+1);
		}
	}

	for(i=0;i<MAX;i++){
		  for(j=0;j<MAX;j++){
				printf("%d ",m[i][j]); /* stampo ogni cella della tavola*/
		  }
		  printf("\n");
		  free(m[i]); /* devo deallocare ogni array creato dinamicamente*/
	 }

	/* E' necessario deallocare anche la memoria dell'array di array */
	free(m);

}
```

8. Cosa stampa il seguente programma?

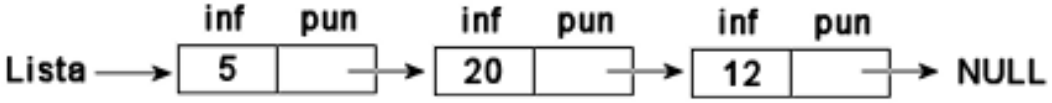
```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 5

int main(){

	int *m;
	int i,j;

	/* Rappresento la matrice come un unico array dove le righe della matrice
	   sono state concatenate una dopo l'altra
	*/
	m = (int*) malloc(sizeof(int) * MAX * MAX);

	for(i=0;i<MAX;i++){
		for(j=0;j<MAX;j++){
		/* Per accedere ad un elemento dell'array e' necessario calcolare
		   la giusta posizione: Es: la cella di riga 1, colonna 2
		   nella matrice (m[1][2]) e' l'elemento di posizione (1 * MAX + 2)
		   nell'array
		*/
			m[i * MAX + j] = (i+1) * (j+1);
		}
	}

	for(i=0;i<MAX;i++){
		  for(j=0;j<MAX;j++){
			printf("%d ",m[i * MAX + j]);
		  }
		  printf("\n");
	 }
	free(m);
}
```

----------LISTE----------

La rappresentazione grafica di un elemento della lista è la seguente:



mentre quella di una lista risulta essere:



9. **Creare una lista dinamica di interi in cui gli elementi sono mantenuti in ordine crescente: implementare i metodi che permetto di creare la lista, inserire un nuovo elemento e stampare la lista.**

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

/* Dichiaro la struttura che rappresenta un elemento della lista */

typedef struct EL{
	int valore;
	struct EL * next;
} Elemento;

/* Il tipo Lista e' semplicemente un puntatore al primo elemento della lista */

typedef Elemento *Lista;

/* procedura crea_lista: assegna alla testa di una lista il valore NULL,
	per simboleggiare che la lista e' vuota*/

void crea_lista(Lista * lista){
	*lista = NULL;
}

/* La procedura stampa_lista stampa in modo ricorsivo tutti gli elementi
della lista, dal primo all'ultimo */

void stampa_lista(Lista lista){

	if(lista != NULL){
		printf("%d ",lista->valore); /* stampa il valore dell'elemento corrente */
		stampa_lista(lista->next); /*Passo ricorsivo su cio' che rimane */
	}
	else{
		printf("\n");
	}

}


void inserisci(Lista *lista, int valore) {

/* Se la lista e' terminata o il valore corrente e' maggiore
del valore da inserire, allora e' il momento di creare il nuovo elemento */

	if(*lista == NULL || (*lista)->valore > valore){
	/* Alloca nuova area di memoria */
	Elemento * el = (Elemento*) malloc(sizeof(Elemento));
	el->valore = valore; /* Assegna il valore */
	el->next = *lista; /* Il nuovo elemento punta all'elemento corrente */
	*lista = el; /* L'elemento precedente punta al nuovo elemento */
	}
	else{
		/* Altrimenti procedi ricorsivamente sul resto della lista */
		inserisci(&((*lista)->next), valore);
	}
}
```

```c
int main(){

	Lista lista;
	crea_lista(&lista);

	printf("Lista ");
	stampa_lista(lista);


	inserisci(&lista, 3);
	inserisci(&lista, 1);
	 inserisci(&lista, 5);
	 inserisci(&lista, 3);
	 inserisci(&lista, 2);
	printf("Lista");
	stampa_lista(lista);
}
```

10. **Creare una lista bidirezionale di interi, ovvero una lista che permette lo scorrimento in entrambe le direzioni (dal primo elemento allultimo e dallultimo al primo). Implementare i metodi che permettano di**
    - **creare la lista ,**
    - **inserire un nuovo elemento al termine della lista,**
    - **stampare la lista in entrambi i sensi**

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


typedef struct EL{
	int valore;
	struct EL *next; /* Puntatore all'elemento successivo in lista */
	struct EL *prev; /* Puntatore all'elemento precedente in lista */
} Elemento;


/* La testa della lista e' un elemento in cui il puntatore next punta al primo elemento
della lista, mentre il puntatore prev punta all'ultimo elemento della lista */

typedef Elemento Lista;

/* Procedura che inizializza la lista come lista vuota (entrambi i puntatori a NULL */

void crea_lista(Lista * lista){

	lista -> next = NULL;
	lista -> prev = NULL;
}


/* Procedura che stampa la lista dal primo elemento all'ultimo */

void stampa_lista_forward(Lista lista){

	if(lista.next!=NULL){
		/* Invertendo l'ordine di printf e chiamata ricorsiva, si
		   ottiene una stampa dall'ultimo elemento al primo */
```

```
		printf("%d ",lista.next->valore);
		stampa_lista_forward(*(lista.next));
	}
}

/* Procedura che stampa la lista dall'ultimo elemento al primo */
void stampa_lista_backward(Lista lista){

	if(lista.prev!=NULL){
		printf("%d ",lista.prev->valore);
		stampa_lista_backward(*(lista.prev));
	}
}

/* Procedura per l'inserimento di un nuovo elemento al termine della lista */
void inserisci(Lista * lista, int valore){

/* Non c'e' bisogno di iterare fino all'ultimo elemento perche' il
suo puntatore e' disponibile nella testa della lista */

	Elemento *elemento = (Elemento*) malloc(sizeof(Elemento));
	elemento->valore = valore;
	elemento->next = NULL;
	elemento->prev = lista->prev;

	if(lista->prev !=NULL){
		/* aggiorno il puntatore di quello che era l'ultimo
		   elemento della lista
		*/
		lista->prev->next=elemento;
	}
	else{
		/* Nel caso la lista sia vuota, il nuovo elemento diventa il primo
		della lista */
		lista->next=elemento;
	}
	/* l'elemento diventa l'ultimo elemento della lista*/
	lista->prev=elemento;
}

int main(){

	Lista lista;

	crea_lista(&lista);
	inserisci(&lista,1);
	inserisci(&lista,3);
	inserisci(&lista,2);

	stampa_lista_forward(lista);
	printf("\n");
	stampa_lista_backward(lista);
	printf("\n");   }
```