C: Oggetti dinamici

Elementi di Informatica 2018/2019

Oggetti dinamici

I puntatori possono essere usati per la creazione e la gestione di oggetti dinamici

Gli oggetti dinamici sono creati durante l'esecuzione del programma. Non è quindi necessaria una dichiarazione esplicita: *non serve il nome e non serve conoscerne il numero*.

La memoria viene allocata attraverso l'utilizzo di specifiche funzione: `malloc()`, che infatti restituisce un puntatore

Allocazione dinamica della memoria

Utilizzo della funzione `malloc()` - serve `stdlib.h`

```
p = (int *) malloc (sizeof(int));
```

Questa istruzione permette di allocare (dinamicamente) la memoria necessaria per un intero che sarà accessibile utilizzando il puntatore p.

`(int *)` è un *cast* al tipo specifico.

- Da questo momento possiamo accedere (tramite il puntatore p) a quella locazione di memoria e quindi all'intero senza l'utilizzo di una dichiarazione esplicita.

Allocazione dinamica della memoria

```
p = (int *) malloc (sizeof(int));
```

Possiamo modificare il valore puntato da `p` utilizzando la già nota rappresentazione:

```
*p=10;
```

E' necessario deallocare lo spazio per evitare che diventi successivamente inutilizzabile!

```
free(p);
```

Array e liste concatenate

- Array:
  - Occupazione memoria (sovrastima)
  - Velocità (inserimento/eliminazione e spostamento)
- Liste lineari:
  - Insieme di elementi omogenei memorizzati in una posizione qualsiasi (semplicità di inserimento/eliminazione e spostamento)

Primo esempio, crea lista

Creare una lista di N interi e visualizzarla

Primo esempio, crea lista

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);

int main() {
    struct lista *punt_lista;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
}
```

```
struct lista *crealista() {
  int n,i;
  struct lista *p,*paux;

  printf("Quanti elementi vuoi inserire? ");
  scanf("%d",&n);
  if(n==0) p=NULL;
  else {
    p=(struct lista *)malloc(sizeof(struct lista));
    printf("Inserisci valore: ");
    scanf("%d",&p->num);
    paux=p;
    for(i=2;i<=n;i++) {
 paux->next=(struct lista *)malloc(sizeof(struct lista));
      paux=paux->next;
      printf("Inserisci valore: ");
      scanf("%d",&paux->num);
    }
    paux->next=NULL;
  }
  return p;
}
```

```c
void visualizza_lista(struct lista *p) {
  while(p!=NULL) {
    printf("%d ",p->num);
    p=p->next;
  }
}
```

Esempio: calcola minimo

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);
int minimo(struct lista *);

int main() {
    struct lista *punt_lista;
    int min;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
    if(punt_lista!=NULL) printf("\nIl min vale: %d\n",
    minimo(punt_lista));
}
```

```c
int minimo(struct lista *p) {
  int min=p->num;

  while(p!=NULL) {
    if(p->num<min) min=p->num;
    p=p->next;
  }
  return min;
}
```

Crea lista, in testa

```
struct lista *crea_intesta() {
  int n,i;
  struct lista *p,*paux;

  printf("Quanti elementi vuoi inserire? ");
  scanf("%d",&n);
  if(n==0) p=NULL;
  else {
    p=(struct lista *)malloc(sizeof(struct lista));
    printf("Inserisci valore: ");
    scanf("%d",&p->num);
    p->next=NULL;
    for(i=2;i<=n;i++) {
      paux=(struct lista *)malloc(sizeof(struct lista));
      printf("Inserisci valore: ");
      scanf("%d",&paux->num);
      paux->next=p;
      p=paux;
    }
  }
  return p;
}
```

Inserisci elemento in una lista

La funzione deve **aggiungere** un elemento alla lista ad ogni chiamata.
L'inserimento può essere eseguito:

- in coda
- in testa
- in ordine

Inserisci elemento in coda

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *ins_coda(struct lista *);
void visualizza_lista(struct lista *);
```

```c
int main() {
    struct lista *punt_lista=NULL;
    int sel=-1;

    while(sel!=0) {
      printf("\n1- Inserisci elemento\n");
      printf("\n2- Visualizza elementi\n");
      printf("\n0- Esci\n");
      scanf("%d",&sel);
      switch (sel) {
        case 1:
          punt_lista=ins_coda(punt_lista);
          break;
        case 2:
          if(punt_lista!=NULL) {
            printf("\nLista elementi: ");
            visualizza_lista(punt_lista);
          }
          else printf("\nLista vuota\n");
          break;
        case 0:
          printf("\nProgramma terminato\n");
          break;
        default:
          printf("\nScelta non valida\n");
```

```c
struct lista *ins_coda(struct lista *p) {
  struct lista *p1,*paux;

  p1=(struct lista *)malloc(sizeof(struct lista));
  printf("Inserisci valore: ");
  scanf("%d",&p1->num);
  if(p==NULL) {
    p=p1;
    p->next=NULL;
  }
  else {
    p1->next=NULL;
    paux=p;
    while(paux->next!=NULL) paux=paux->next;
    paux->next=p1;
  }
  return p;
}
```

Inserisci elemento in testa

```
struct lista *ins_testa(struct lista *p) {
  struct lista *p1;

  p1=(struct lista *)malloc(sizeof(struct lista));
  printf("Inserisci valore: ");
  scanf("%d",&p1->num);
  if(p==NULL) {
    p=p1;
    p->next=NULL;
  }
  else {
    p1->next=p;
    p=p1;
  }
  return p;
}
```

Inserisci elemento in ordine

```
struct lista *ins_ord(struct lista *p) {
  struct lista *p1,*paux;
  p1=(struct lista *)malloc(sizeof(struct lista));
  printf("Inserisci valore: ");
  scanf("%d",&p1->num);
  if(p==NULL) {
    p=p1;
    p->next=NULL;
  }
  else {
    if(p1->num<p->num) {
      p1->next=p;
      p=p1;
    }
    else {
   paux=p;
   while(paux->next != NULL && p1->num > paux->next->num)
      paux=paux->next;
   p1->next=paux->next;
   paux->next=p1; }}
  return p;
}
```

Elimina elemento da una lista

```
struct lista *elimina(struct lista *p) {
    struct lista *paux,*p2;
    int el;
    int trovato=0;

    printf("Inserisci elemento da eliminare: \n");
    scanf("%d",&el);
    if(p!=NULL) {
        if(p->num == el) {
            p2=p;
            p=p->next;
            free(p2);
            return p;
        }
```

Elimina elemento da una lista

```
        else {
            paux=p;
            while(paux->next!=NULL && trovato!=1) {
              if(paux->next->num != el) paux=paux->next;
              else {
                  trovato=1;
                  p2=paux->next;
                  paux->next=paux->next->next;
                  free(p2);
                  return p;
                }
            }
        }
        if(!trovato) printf("Elemento non presente!\n");
    }
    else printf("La lista e' vuota!\n");
    return p;
}
```