Programmazione I - Laboratorio

Esercitazione 5

Struct, Union e allocazione dinamica della memoria

Gianluca Mezzetti[1] Paolo Milazzo[2]

1. Dipartimento di Informatica, Università di Pisa
`http://www.di.unipi.it/~mezzetti`
`mezzetti@di.unipi.it`

2. Dipartimento di Informatica, Università di Pisa
`http://www.di.unipi.it/~milazzo`
`milazzo@di.unipi.it`

Corso di Laurea in Informatica
A.A. 2012/2013

Allocazione dinamica della memoria (1)

Lo heap è un'area di memoria a disposizione di un programma in esecuzione

L'allocazione e la deallocazione della memoria all'interno di questo segmento sono a carico del programmatore.

É possibile allocare lo heap per mezzo della funzione

```
void* malloc(size_t n)
```

ritorna un puntatore ad *n* byte contigui riservati sullo heap.

Il risultato di `malloc` va castato al tipo opportuno

```
int main()
{
  int *ptr;

  ....

  ptr = (int*) malloc(24*sizeof(int));

  ....
}
```

Allocazione dinamica della memoria (2)

Varianti:

```
void* calloc(size_t nelem, size_t dimelem)
```

alloca un'area di memoria sufficiente ad ospitare un array di *nelem* elementi, ogniuno di dimensione *dimelem* byte ed in aggiunta ne azzera il contenuto.

```
void* realloc(void *ptr, size_t dimelem)
```

prende un puntatore ad un'area di memoria già allocata (da `malloc`) e ne cambia la dimensione allocata preservando il contenuto precedente

Allocazione dinamica della memoria (3) I

Ogni volta che viene allocata della memoria è necessario deallocare la memoria allocata utilizzando la funzione

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

Esempio:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
/* allocazione dinamica delle memoria */
int main(void)
{
        char* s; int i; srand(time(0));

        s = (char*) malloc(21*sizeof(char));

        for(i = 0; i < 20; i++) s[i] = rand()%255;
        s[20]='\0';
        printf("len(%s)=%lu\n", s, strlen(s));

        /* Commentare la free, compilare ed eseguire: il
            funzionamento in apparenza e' lo stesso.
```

Allocazione dinamica della memoria (3) II

```
        Usare valgrind per verificare che invece un problema di non
             deallocazione della memoria esiste: valgrind --tool=
            memcheck file */
         free(s);

  return 0;
}
```

Esempio: Stringa in input senza limiti - inputstringa.c I

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

#define BUFSIZE 16*sizeof(char)

int main(void)
{
        char s[BUFSIZE];
        int i=0,z=0;
        int ch;
        int fine=0;
        char* sfinale;

        sfinale = (char*)malloc(BUFSIZE);
        while(!fine){

                i=0;
                while(!fine && i<BUFSIZE) {
                        ch = getchar();
                        if(ch==EOF)
                        {
                                s[i] = '\0';
                                fine=1;
                        }
                        else
                        {
```

Esempio: Stringa in input senza limiti - inputstringa.c II

```
                                s[i] = ch;
                                i++;
                        }
                }

                sfinale[(z)*BUFSIZE]='\0';
                strncpy(&sfinale[(z)*BUFSIZE],s,BUFSIZE);

                if(!fine){
                        z++;
                        sfinale = (char*)realloc(sfinale, (z+1) *
                            BUFSIZE);
                }
        }

        printf("Hai inserito %s", sfinale);
        free(sfinale);

        /*printf("Hai inserito %s", sfinale);*//*Sbagliato*/
        return 0;
}
```

Struct (1)

- Una *Struct* è una collezione di una o più variabili (dette *fields*, o *campi*), anche di tipo differente, raggruppate sotto un unico nome.
- Le strutture sono uno dei modi in cui è possibile creare nuovi tipi.

Esempio di dichiarazione:

```
struct point {
        int x;
        int y;
};
```

Una variabile di tipo *struct point* può essere definita con:

```
struct point a;
struct point b = { 1 , 3 }; /* inizializzazione rapida */
```

I *fields* della struttura possono essere acceduti con la notazione

```
a.x = 10;
k = a.x;
```

Esempio: struct_point.c I

Dati due punti possiamo calcolare la distanza tra essi come segue

```
#include <math.h> /* compilare con -lm in fondo */
#include <stdlib.h>
#include <stdio.h>

struct point {
  int x;
  int y;
};

int main()
{
  struct point a;
  struct point b;
  float dist;
  int X,Y;

  printf("Inserisci le cordinate del primo punto: ");
  scanf("%d %d",&a.x,&a.y);

  printf("Inserisci le cordinate del secondo punto: ");
  scanf("%d %d",&b.x,&b.y);

  X = abs(a.x-b.x);
  Y = abs(a.y-b.y);
```

Esempio: struct_point.c II

```
  dist = sqrt(X*X+Y*Y); /* sqrt = radice quadrata */

  printf("La distanza tra (%d,%d) e (%d,%d) e' %.3f\n",a.x,a.y,b.x,
      b.y,dist);

  return 0;
}
```

Unions

- Le *Unions* sono strutture dati che raggruppano in maniera mutuamente esclusiva *fields*.

```
union data {
    time_t secsfrom1970;
    char[9] datatext;
};
```

- La dimensione della struttura dati è ottenuta considerando la massima dimensione tra i tipi dei fields.
- L'assenza di pattern-matching *obbliga* a ottenere in altri modi l'informazione su quale sia il tipo realmente contenuto.

Esempio: Unions - unions.c I

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

enum tipodata { /* enumerazione */
        TESTUALE=0, /* nuovo tipo i cui valori sono */
        NUMERICA=1  /* TESTUALE e NUMERICA */
};

union data {
    int secsda1970;
    char testodata[9];
};

struct datainserita {
        enum tipodata tipo;
        union data valore;
};
int main(void)
{
        struct datainserita d;
        d.tipo = TESTUALE;
        strcpy(d.valore.testodata,"01/12/12");

        /*....*/

        /*Uso corretto*/
        if(d.tipo == TESTUALE)
```

Esempio: Unions - unions.c II

```
                printf("Valore testuale %s \n",d.valore.testodata);

        /*Uso scorretto*/
        printf("Stessa union prendendo il field numerico %d ,
            prendendo i primi %lu byte \n",d.valore.secsda1970,
            sizeof(int));

        return 0;
}
```

Esempio: struct_array.c I

Possiamo realizzare array di struct (e unions)...

```
#include <math.h> /* compilare con -lm in fondo */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_PUNTI 5

struct point {
  int x;
  int y;
};

float distanza(struct point a);

int main()
{
  struct point punti[NUM_PUNTI];
  int minimo;
  int i;

  srand(time(0));

  /* inizializza casualmente i punti */
  for (i=0; i<NUM_PUNTI; i++)
  {
```

Esempio: struct_array.c II

```
    punti[i].x = rand()%100;
    punti[i].y = rand()%100;
    printf("punti[i] = (%d,%d)\n",punti[i].x,punti[i].y);
  }

  /* trova il piu' vicino all'origine */
  minimo=0;
  for (i=1; i<NUM_PUNTI; i++)
    if (distanza(punti[minimo])>distanza(punti[i]))
       minimo=i;

  printf("Il punto piu' vicino all'origine e': (%d,%d)\n",punti[
      minimo].x,punti[minimo].y);
}


float distanza(struct point a)
{
  return sqrt(a.x*a.x + a.y*a.y);
}
```

Esempio: struct_rect.c I

Le struct (e le unions) possono essere annidate

```
#include <stdio.h>

struct point
{
  int x;
  int y;
};

/* un rettangolo si puo' rappresentare con due punti (angoli
    opposti) */
struct rect
{
  struct point a;
  struct point b;
};

int main()
{
  struct rect r1 = { { 2 , 2 } , { 4 , 4 } };
  struct rect r2;

  r2.a.x = 3;
  r2.a.y = 3;
  r2.b.x = 7;
  r2.b.y = 7;
```

Esempio: struct_rect.c II

```
  printf("Area di r1: %d\n",(r1.b.x-r1.a.x)*(r1.b.y-r1.a.y));
  printf("Area di r2: %d\n",(r2.b.x-r2.a.x)*(r2.b.y-r2.a.y));

  return 0;
}
```

Esempio: struct_alloc.c

E' possibile allocare dinamicamente struct nello heap!

ATTENZIONE: Quando si accede ad una struct tramite un puntatore bisogna usare -> al posto di .

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

struct point
{
  int x;
  int y;
};

int main()
{
  struct point *p = (struct point*) malloc(sizeof(struct point));

  p->x = 10;
  p->y = 15;

  printf("Il punto e' (%d,%d)\n",p->x,p->y);
}
```

Esempio: struct_lista.c I

Infine, e' possibile definire strutture ricorsive

```
/*
 crea una lista e la stampa
*/
#include <stdio.h>
#include <stdlib.h>

struct El {
  int valore;
  struct El* next;
};

int main()
{
  int n;
  struct El* lista;
  struct El* l;

  lista=NULL;
  do
  {
    printf("Inserisci un naturale o -1 per terminare\n");
    scanf("%d",&n);
    if (n>=0) {
      if (lista==NULL) /* prima iterazione */
      {
```

Esempio: struct_lista.c II

```
        lista = (struct El*) malloc(sizeof(struct El));
        l = lista;
      }
      else /* iterazioni successive */
      {
        l->next = (struct El*) malloc(sizeof(struct El));
        l = l->next;
      }
      l->valore = n;
      l->next = NULL;
    }
  }
  while (n>=0);


  l=lista;
  printf("numeri inseriti: ");
  while (l!=NULL)
  {
    printf("%d ",l->valore);
    l=l->next;
  }
  printf("\n");

  return 0;
}
```

Esercizio

Definire un nuovo tipo di dato capace di rappresentare una data come tre numeri (giorno, mese, anno).
Scrivere una procedura che ricevuta una data (per riferimento) la aggiorni al giorno successivo (ignorando gli anni bisestili);

Esercizio – date.c I

```c
/*
 calcola il giorno successivo
*/
#include <stdio.h>

struct data {
  int giorno;
  int mese;
  int anno;
};


void giorno_successivo(struct data *d);


int main()
{
  struct data d1 = { 31 , 1 , 1979};

  giorno_successivo(&d1);

  printf("%d/%d/%d\n", d1.giorno, d1.mese, d1.anno);

  return 0;
}
```

Esercizio – date.c II

```
void giorno_successivo(struct data *d)
{
  switch (d->mese)
  {

    case 2:
      if (d->giorno==28)
      {
        d->mese=3;
        d->giorno=1;
      }
      else d->giorno = d->giorno+1;
      break;

    case 4: case 6: case 9: case 11 :
      if (d->giorno==30)
      {
        d->mese=d->mese+1;
        d->giorno=1;
      }
      else d->giorno=d->giorno+1;

    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      if (d->giorno==31)
      {
        if (d->mese==12) {d->mese=1; d->anno=d->anno+1;}
        else d->mese=d->mese+1;
```

Esercizio – date.c III

```
            d->giorno=1;
        }
        else d->giorno=d->giorno+1;
    }
}
```

Esercizio

Definire un nuovo tipo di dato capace di rappresentare i dipendenti di una ditta: di tali dipendenti interessa il cognome, il numero degli anni di anzianita' maturati e lo stipendio. Si supponga che la dimensione massima del cognome sia di 40 caratteri.

Assumere che i dati dei dipendenti della ditta siano memorizzati in un array di dimensione `NUM_DIPENDENTI`.

Scrivere una funzione che, dato il cognome di un dipendente, restituisce il numero di anni di anzianità maturati.

Scrivere una funzione che restituisce il cognome del dipendente con stipendio minimo

Esercizio – dipendenti.c I

```
/*
  gestione dipendenti di una azienda
 */
#include <stdio.h>
#include <string.h>

#define NUM_DIPENDENTI 5

struct dipendente
{
  char cognome[40];
  int anni;
  int stipendio;
};

int anzianita(struct dipendente dip[], char *cognome);

char* piu_povero(struct dipendente dip[]);

int main()
{
  struct dipendente dipendenti[NUM_DIPENDENTI];

  strcpy(dipendenti[0].cognome,"rossi");
  dipendenti[0].anni = 3;
  dipendenti[0].stipendio = 1200;
  strcpy(dipendenti[1].cognome,"bianchi");
```

Esercizio – dipendenti.c II

```
  dipendenti[1].anni = 6;
  dipendenti[1].stipendio = 2200;
  strcpy(dipendenti[2].cognome,"verdi");
  dipendenti[2].anni = 1;
  dipendenti[2].stipendio = 800;
  strcpy(dipendenti[3].cognome,"gialli");
  dipendenti[3].anni = 12;
  dipendenti[3].stipendio = 12000;
  strcpy(dipendenti[4].cognome,"brambilla");
  dipendenti[4].anni = 5;
  dipendenti[4].stipendio = 1600;

  printf("%d\n", anzianita(dipendenti, "gialli"));

  printf("%s\n", piu_povero(dipendenti));
}


int anzianita(struct dipendente dip[], char *cognome)
{
  int i=0;
  int res=-1;
  while (res<0 && i<NUM_DIPENDENTI)
  {
    if (strcmp(cognome,dip[i].cognome)==0)
      res = dip[i].anni;
    i++;
```

Esercizio – dipendenti.c III

```
  }
  return res;
}

char* piu_povero(struct dipendente dip[])
{
  int i;
  int minimo = 0;
  for (i=1; i<NUM_DIPENDENTI; i++)
  {
    if (dip[i].stipendio<dip[minimo].stipendio)
      minimo=i;
  }
  return &dip[minimo].cognome[0];
}
```

Esercizio

Definire una lista di interi bidirezionale.
In questa lista ogni nodo dovra' avere un puntatore al nodo precedente (`prev`) e un puntatore al nodo successivo della lista (`next`), oltre al valore contenuto (`val`, un intero).

Naturalmente se il nodo precedente o il nodo successivo non esistesse, il corrispondente puntatore sarebbe `NULL`.

Si noti che a differenza delle liste semplici, per manipolare una lista bidirezionale basta avere un puntatore a uno qualunque dei suoi nodi, non necessariamente alla testa.

Scrivere una programma nel cui `main` viene inizializzata una lista bidirezionale con 5 valori casuali.

Scrivere una funzione `stampa_lista` che, ricevuto un puntatore a un nodo qualsiasi di una lista bidirezionale, la stampi in questo modo:

```
// <-> 1 <-> 2 <-> 3 <-> 4 <-> //
```

Esercizio – lista_bidir.c I

```
/*
 lista bidirezionale
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct lista_bidir {
  struct lista_bidir *prev, *next;
  int val;
};

void stampa_lista(struct lista_bidir *el);

int main()
{
  int i;
  struct lista_bidir *lista, *tmp;

  srand(time(0));

  lista = (struct lista_bidir*) malloc(sizeof(struct lista_bidir));
  lista->prev=NULL;
  lista->val=rand()%100;
  tmp = lista;
  for (i=1; i<5; i++)
  {
```

Esercizio – lista_bidir.c II

```
    tmp->next = (struct lista_bidir*) malloc(sizeof(struct
        lista_bidir));
    tmp->next->prev = tmp;
    tmp=tmp->next;
    tmp->val = rand()%100;
  }
  tmp->next=NULL;

  stampa_lista(lista->next->next);

  /* libero la memoria allocata */
  while (lista!=NULL)
  {
    tmp = lista;
    lista = lista->next;
    free(tmp);
  }
}


void stampa_lista(struct lista_bidir *el)
{
  struct lista_bidir *tmp;

  if (el!=NULL)
  {
    tmp = el->prev;
```

Esercizio – lista_bidir.c III

```
    while (tmp!=NULL) {el=tmp; tmp=tmp->prev;}
    /* ora el punta al primo elemento */

    printf("// <-> ");
    while (el!=NULL)
    {
      printf("%d <-> ",el->val);
      el=el->next;
    }
    printf("//\n");
  }
}
```