Lezione 13: Strutture dati dinamiche

Laboratorio di Elementi di Architettura e Sistemi Operativi

7 Maggio 2013

Riepilogo lezioni

Data	Argomento	Compitino
mar 07/05	Programmazione C: strutture	
mer 08/05	Programmazione C: Librerie	Si
mar 14/05	Programmazione C: Makefile	Si
mer 15/05	Introduzione architettura LC3	
mar 21/05	*Festa del patrono* (no lezione)	
mer 22/05	Programmazione LC3	Si
mar 28/05	Consegna progetti per esame	

I tipi di dati strutturati

Tipi strutturati

- In C è possibile definire dati composti da elementi eterogenei (*record*), aggregandoli in una singola variabile
- Individuata dalla keyword `struct`
- Simile alle classi (ma non hanno metodi!)
- Sintassi (definizione di tipo):

```
struct nome {
  campi
};
```

- I campi sono nel formato `tipo nomecampo;`
- Esempi:

```
struct complex {
  double re;
  double im;
};

struct identity {
  char nome[30];
  char cognome[30];
  char codicefiscale[15];
  int altezza;
  char stato_civile;
};
```

- Un definizione di `struct` equivale ad una definizione di tipo
- Successivamente una struttura può essere usata come un tipo per definire variabili e argomenti di funzione:

```
struct complex {
  double re;
  double im;
};

struct complex somma(struct complex num1,
                     struct complex num2)
{
  struct complex num3;
  ...
}
```

- Una struttura permette di accedere ai singoli campi tramite l'operatore '`.`', applicato a variabili del corrispondente tipo struct:

```
struct complex somma(struct complex num1,
                     struct complex num2)
{
  struct complex num3;
  num3.re = num1.re + num2.re;
  num3.im = num1.im + num2.im;
  return num3;
}
```

- È possibile definire un nuovo tipo a partire da una `struct` tramite la direttiva `typedef`
- Sintassi: `typedef` *`vecchio_tipo`* `nuovo_tipo;`
- Esempio:

```
typedef struct complex {
  double re;
  double im;
} complex_t;

complex_t num1, num2;
```

Allocazione dinamica di strutture

- Supponiamo di dichiarare una struttura per rappresentare punti nel piano cartesiano:

```
typedef struct pt {
    float x,y;
  } Point;
```

- La dichiarazione `struct pt *pp;` afferma che `pp` è un puntatore ad una `struct pt`;
- `*pp` è la struttura, `(*pp).x` e `(*pp).y` sono i membri
- *Attenzione:* l'espressione `*pp.x` è scorretta!
- Una notazione alternativa (e consigliata) usa l'operatore `->`:

```
pp->x;      pp->y;
```

- Allocazione e deallocazione si gestiscono come i tipi base:

```
Point* segment;
segment = (Point*) malloc(2*sizeof(Point));
/* vettore di due variabili point */
/* equivalente a Point segment[2]  */
if (segment == NULL)
{ fprintf(stderr, "Out of memory\n");
  return -1;
}
...
free(segment);
```

I tipi di dati dinamici

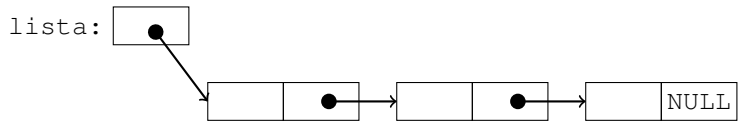
Strutture ricorsive

- Supponiamo di dover implementare una *lista* di numeri interi, di lunghezza non nota a priori
- Ogni elemento della lista è rappresentato da una struttura con due componenti:

```
typedef struct elem {
  int key;
  struct elem *next;
} elemen_t;
```

- `key` contiene il numero
- `next` è un puntatore al prossimo elemento della lista
- Una lista vuota è rappresentata da un puntatore `NULL`, una lista non vuota è rappresentata da un *puntatore al primo elemento*:

```
typedef elemen_t* lista_t;
```



- Test di lista vuota:

```
int is_empty(lista_t lista) {
  return(lista == NULL);
}
```

- **Esempio:** ricerca di un elemento nella lista:

```
...
lista_t lista, curr;
int chiave;
...
for(curr = lista; curr != NULL; curr = curr->next) {
  if(curr->key == chiave) {
    printf("Il valore %d e' presente nella lista.\n",
           chiave);
    break;
  }
}
if(curr == NULL) {
  printf("Il valore %d non e' presente nella lista.\n",
         chiave);
}
...
```

- Lunghezza di una lista:

```
int length(lista_t lista) {
  int n = 0;
  while(lista != NULL) {
    n++;
    lista = lista->next;
  }
  return n;
}
```

- Inserimento di un elemento in testa alla lista:

```
lista_t insert(lista_t lista, int key) {
  elemen_t *paux;
  paux = (elemen_t *)malloc(sizeof(elemen_t));
  paux->key = key;
  paux->next = lista;
  return paux;
}
```

- Valore del primo elemento:

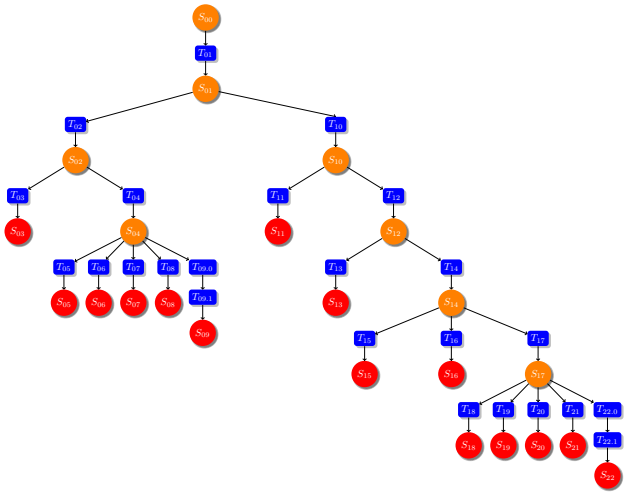
```
int head(lista_t lista) {
   if(lista != NULL) {
     return lista->key;
   }
   return 0;
}
```

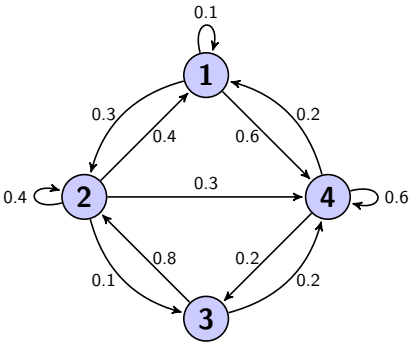
- Rimozione del primo elemento di una lista:

```
lista_t delete(lista_t lista) {
  lista_t ret;
  if(lista != NULL) {
    ret = lista->next;
    free(lista);
    return ret;
  }
  return NULL;
}
```

Altri esempi di strutture dati dinamiche

- Utilizzando le strutture ricorsive e l'allocazione dinamica della memoria, è possibile rappresentare anche altri tipi di dati dinamici, come *pile*, *alberi* e *grafi*.





Esercizi di oggi

- Dalla pagina Web del corso è possibile scaricare il file `liste.c` con il codice per la gestione delle liste.
- Il file va incluso nel programma con la direttiva `#include "liste.c"`