E-05: STRUTTURE E PUNTATORI

Fondamenti di Informatica e Laboratorio T-AB

Corso di Laurea in Ingegneria dell'Automazione

Università di Bologna, A.A. 2008/2009

Paolo Torroni, Marco Montali

30 Marzo 2009

Esercizio 1.

1. Si legga un numero intero n da input.
2. Si definisca un vettore `V` di n puntatori a `char`.
3. Si leggano n caratteri da input e si salvi ciascun carattere in una variabile dinamica creata appositamente, il cui indirizzo è conservato in un elemento di `V`.
4. Si stampino i caratteri puntati dagli elementi di `V` di indice dispari, seguiti dai caratteri puntati dagli elementi di `V` di indice pari.
5. Si liberi la memoria dalle strutture dati create.

Esercizio 2.

1. Si definisca un nuovo tipo di dato `L`, come record formato da un intero (campo `value`) e da un puntatore a `L` (campo `next`).
2. Si definisca una variabile `p` di tipo puntatore a `L`.
3. Mediante un ciclo, si definiscano 10 variabili dinamiche di tipo `L`, in modo che
   - `p` punti alla 1a variabile creata,
   - il puntatore di tale variabile punti alla 2a,
   - il puntatore della 2a punti alla 3a variabile,
   - etc.

   Si inizializzi l'ultimo puntatore al valore `NULL`.
4. Si inizializzino i campi `value` dei record creati alle potenze di due (2^1, 2^2, ..., 2^{10}), nell'ordine dal primo all'ultimo.
5. Si mostrino tutti i valori inizializzati, dal primo all'ultimo.
6. Si liberi la memoria dalle strutture dati create.

Esercizio 3.

Si scriva una funzione che:

- accetti come unico parametro un puntatore a `L` (vedi punto precedente),
- restituisca in uscita il numero di strutture collegate l'una all'altra, considerando il valore `NULL` come indicatore del raggiungimento dell'ultimo elemento (nel caso precedente, il risultato sarebbe 10).

Esercizio 4.

1. Si definisca un nuovo tipo di dato `LL`, come record formato da un intero (campo `value`) e da due puntatori a `L` (campi `prev` e `next`).
2. Si definiscano 10 variabili dinamiche di tipo `LL`, in modo che ciascun elemento punti al precedente (con `prev`) e al successivo (con `next`). Si inizializzino il primo puntatore `prev` e l'ultimo `next` al valore `NULL`.
3. Si mostrino tutti i valori, percorrendo gli elementi prima in un senso poi nell'altro.
4. Si liberi la memoria dalle strutture dati create.

NOTA: per la definizione dei tipi di dato `struct` con self-reference (ad es., il tipo `L` dell'esercizio 2, che contiene tra i campi un puntatore a `L`), si usi la seguente sintassi, con:

- la parola chiave `struct` prima del nome del nuovo tipo, **anche dentro l'elenco dei campi**,
- l'indicazione del nome del tipo sia **prima** sia **dopo** la lista dei campi:

```
typedef struct nome {
     tipo1 campo1;
     struct nome *campo2;
} nome;
```

SOLUZIONI

Esercizio 1.

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

typedef char* ptr_c;

int main () {
    int i, n;
    ptr_c *V;
    printf( "quanti caratteri? ... " );
    scanf( "%d", &n );
    V=( ptr_c* )malloc( n*sizeof( ptr_c ) );
    for( i=0; i<n; i++ ) {
        V[i]=( ptr_c )malloc( sizeof( char ) );
        printf( "%d -> ... ", i );
        do {
            scanf( "%c", V[i] );
        } while( ( *V[i] )==’\n’ );
    }
    for( i=0; i<n; i++ )
        printf( "%c", *V[i] );
    for( i=0; i<n; i++ )
        free( V[i] );
    free( V );
    return 0;
}
```

Esercizio 2.

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

// 1. definisci il tipo di dato
typedef struct L {
    int value;
    struct L *next;
} L;

// 3. 4. crea e inizializza n istanze di L
L *crea_lista( int n, int start_value ) {
    L *p=NULL, *last=NULL;
    int i;
    p=last=( L* )malloc( sizeof( L ) );
    p->value=start_value;
    p->next=NULL;
    for( i=1; i<n; i++ ) {
        last->next=( L* )malloc( sizeof( L ) );
        last->next->value=2*last->value;
        last->next->next=NULL;
        last=last->next;
    }
    return p;
}

// 5. mostra il contenuto
void mostra_contenuto( L *p ) {
    while( p!=NULL ) {
        printf( "%d\n", p->value );
        p=p->next;
    }
}

// 6. libera la memoria
void libera_memoria( L *p ) {
    L *current=p;
    while( current!=NULL ) {
        p=current->next;
        free( current );
        current=p;
    }
}

int main() {
    // 2. definisci la variabile p
    L *p=NULL;

    p=crea_lista( 10, 2 );
    mostra_contenuto( p );
    libera_memoria( p );
}
```

Esercizio 3.

```
int conta( L *p ) {
    int i=0;
    while( p!=NULL ) {
        i++;
        p=p->next;
    }
    return i;
}
```

Esercizio 4.

```
// 1. definisci il tipo di dato
typedef struct LL {
    int value;
    struct LL *prev;
    struct LL *next;
} LL;

// 2. crea 10 istanze di LL e inizializza
LL *crea_lista_doppia( int n, int start_value ) {
    LL *p=NULL, *current=NULL;
    int i;
    p=( LL* )malloc( sizeof( LL ) );
    p->value=start_value;
    p->prev=NULL;
    p->next=NULL;
    current=p;
    for( i=0; i<n; i++ ) {
```

```
        current->next=( LL* )malloc( sizeof( LL ) );
        current->next->value=2*current->value;
        current->next->next=NULL;
        current->next->prev=current;
        current=current->next;
        p->prev=current;
    }
    return p;
}

// 3. stampa in avanti
void mostra_contenuto_fwd( LL *p ) {
    printf( "\nPRINT FORWARD:\n" );
    while( p!=NULL ) {
        printf( "%d\n", p->value );
        p=p->next;
    }
}

// 3. stampa all'indietro
void mostra_contenuto_rev( LL *p ) {
    printf( "\nPRINT REVERSE:\n" );
    while( p->prev->next!=NULL ) {
        printf( "%d\n", p->value );
        p=p->prev;
    }
}

// 4. libera la memoria
void libera_memoria_LL( LL *p ) {
    LL *current=p;
    while( current->next!=NULL ) {
        current=current->next;
        free( p );
        p=current;
    }
    free( current );
}

int main() {
    // 2. definisci le variabili s ed e
    LL *p=NULL;

    p=crea_lista_doppia( 10, 1 );
    mostra_contenuto_fwd( p->next );
    mostra_contenuto_rev( p->prev );
    libera_memoria_LL( p );
}
```