# Quarto Appello di Programmazione I

03 Luglio 2013
Prof. Roberto Sebastiani

Codice:

| Nome | Cognome | Matricola |
|---|---|---|
| | | |

**La directory 'esame' contiene 4 sotto-directory: 'uno', 'due', 'tre' e 'quattro'. Le soluzioni vanno scritte negli spazi e nei modi indicati esercizio per esercizio.**
**NOTA: il codice dato non può essere modificato**

**Modalità di questo appello**

Durante la prova gli studenti sono vincolati a seguire le regole seguenti:

- Non è consentito l'uso di alcun libro di testo o fotocopia. In caso lo studente necessitasse di carta (?), gli/le verranno forniti fogli di carta bianca su richiesta, che dovranno essere riconsegnati a fine prova. È consentito l'uso di una penna. Non è consentito l'uso di alcuno strumento calcolatore.
- È vietato lo scambio di qualsiasi informazione, orale o scritta. È vietato guardare nel terminale del vicino.
- È vietato l'uso di telefoni cellulari o di qualsiasi strumento elettronico.
- È vietato allontanarsi dall'aula durante la prova, anche se si ha già consegnato. (Ogni necessità fisiologica va espletata PRIMA dell'inizio della prova.)
- È vietato qualunque accesso, in lettura o scrittura, a file esterni alla directory di lavoro assegnata a ciascun studente. Le uniche operazioni consentite sono l'apertura, l'editing, la copia, la rimozione e la compilazione di file all'interno della propria directory di lavoro.
- Sono ovviamente vietati l'uso di email, ftp, ssh, telnet ed ogni strumento che consenta di accedere a file esterni alla directory di lavoro. Le operazioni di copia, rimozione e spostamento di file devono essere circoscritte alla directory di lavoro.
- Ogni altra attività non espressamente citata qui sopra o autorizzata dal docente è vietata.

Ogni violazione delle regole di cui sopra comporterà automaticamente l'annullamento della prova e il divieto di accesso ad un certo numero di appelli successivi, a seconda della gravità e della recidività della violazione.

NOTA IMPORTANTE: DURANTE LA PROVA PER OGNI STUDENTE VERRÀ ATTIVATO UN TRACCIATORE SOFTWARE CHE REGISTRERÀ TUTTE LE OPERAZIONI ESEGUITE (ANCHE ALL'INTERNO DELL'EDITOR!!). L'ANNULLAMENTO DELLA PROVA DI UNO STUDENTE POTRÀ AVVENIRE ANCHE IN UN SECONDO MOMENTO, SE L'ANALISI DELLE TRACCE SOFTWARE RIVELASSERO IRREGOLARITÀ.

1 Scrivere un programma che, dato una stringa ed un file di testo contenente un numero indefinito di caratteri, verifichi che nel file di testo specificato esista almeno un'occorenza della stringa specificata. Il programma deve terminare la ricerca dopo aver trovato la prima occorrenza.

Al termine dell'esecuzione di questo programma, dovrà essere stampato a video un opportuno messaggio di avviso del tipo:
"Esiste almeno un'occorrenza della stringa specificata"
oppure "Non esiste un'occorrenza della stringa specificata"

Dato, ad esempio, in input il file `testo` contenente i seguenti dati:

```
Filastrocca delle parole: Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena, con dentro la "luna" e la "balena".
Ci sono parole per gli amici: Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone; parole per ogni sorta di persone.
Di G. Rodari.
```

se l'eseguibile è `a.out`, il comando

```
./a.out ale testo
```

stamperà a video il seguente messaggio:
`Esiste almeno un'occorrenza della stringa specificata`
perchè la stringa `ale` è contenuta nella parola `balena` presente nel file `testo`.

NOTA: è ammesso l'utilizzo delle funzioni di libreria `strstr` e `strcpy` della libreria `cstring`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **esercizio1.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

int main(int argc,char* argv[])
{
  fstream my_in;
  char tmp[100], target[100];
  int numero_parole=0, trovato, i;

  if (argc != 3) {
    cout << "Usage: ./a.out <stringa> <testo>\n";
    exit(0);
  }
  strcpy(target,argv[1]);
  my_in.open(argv[2],ios::in);

  my_in >> tmp; //lettura prima parola
  while ((!my_in.eof())&&(!strstr(tmp,target))) {
    my_in >> tmp; //lettura parola successiva
  }

  if (my_in.eof())
    cout << "Non esiste un'occorrenza della stringa specificata." << endl;
  else
    cout << "Esiste almeno un'occorrenza della stringa specificata." << endl;

  my_in.close();

  return(0);
}
```

1 Scrivere un programma che, dato un codice ed un file di testo contenente un numero indefinito di cifre, verifichi che nel file di testo specificato esista almeno un'occorenza del codice specificato. Il programma deve terminare la ricerca dopo aver trovato la prima occorrenza.

Al termine dell'esecuzione di questo programma, dovrà essere stampato a video un opportuno messaggio di avviso del tipo:
"Esiste almeno un'occorrenza del codice specificato"
oppure "Non esiste un'occorrenza del codice specificato"

Dato, ad esempio, in input il file `tabulato` contenente i seguenti dati:

```
114750157 451865 450159
682524769024 69237
287 87276247 798469
824597804724
```

se l'eseguibile è `a.out`, il comando

```
./a.out 727 tabulato
```

stamperà a video il seguente messaggio:
`Esiste almeno un'occorrenza del codice specificato`
perchè il codice 727 è contenuto nel file `tabulato`.

NOTA: è ammesso l'utilizzo delle funzioni di libreria `strstr` e `strcpy` della libreria `cstring`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **esercizio1.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

int main(int argc,char* argv[])
{
  fstream my_in;
  char tmp[100], target[100];
  int numero_parole=0, trovato, i;

  if (argc != 3) {
    cout << "Usage: ./a.out <codice> <tabulato>\n";
    exit(0);
  }
  strcpy(target,argv[1]);
  my_in.open(argv[2],ios::in);

  my_in >> tmp; //lettura prima parola
  while ((!my_in.eof())&&(!strstr(tmp,target))) {
    my_in >> tmp; //lettura parola successiva
  }

  if (my_in.eof())
    cout << "Non esiste un'occorrenza del codice specificato." << endl;
  else
    cout << "Esiste almeno un'occorrenza del codice specificato." << endl;

  my_in.close();

  return(0);
}
```

1 Scrivere un programma che, dato una stringa ed un file di testo contenente un numero indefinito di caratteri, verifichi che nel file di testo specificato esista almeno un'occorenza della stringa specificata. Il programma deve terminare la ricerca dopo aver trovato la prima occorrenza.

Al termine dell'esecuzione di questo programma, dovrà essere stampato a video un opportuno messaggio di avviso del tipo:
"Esiste almeno un'occorrenza della stringa specificata"
oppure "Non esiste un'occorrenza della stringa specificata"

Dato, ad esempio, in input il file `testo` contenente i seguenti dati:

```
Filastrocca delle parole: Fatevi avanti! Chi ne vuole?
Di parole ho la testa piena, con dentro la "luna" e la "balena".
Ci sono parole per gli amici: Buon giorno, Buon anno, Siate felici!
Parole belle e parole buone; parole per ogni sorta di persone.
Di G. Rodari.
```

se l'eseguibile è `a.out`, il comando

```
./a.out ale testo
```

stamperà a video il seguente messaggio:
`Esiste almeno un'occorrenza della stringa specificata`
perchè la stringa `ale` è contenuta nella parola `balena` presente nel file `testo`.

NOTA: è ammesso l'utilizzo delle funzioni di libreria `strstr` e `strcpy` della libreria `cstring`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **esercizio1.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

int main(int argc,char* argv[])
{
  fstream my_in;
  char tmp[100], target[100];
  int numero_parole=0, trovato, i;

  if (argc != 3) {
    cout << "Usage: ./a.out <stringa> <testo>\n";
    exit(0);
  }
  strcpy(target,argv[1]);
  my_in.open(argv[2],ios::in);

  my_in >> tmp; //lettura prima parola
  while ((!my_in.eof())&&(!strstr(tmp,target))) {
    my_in >> tmp; //lettura parola successiva
  }

  if (my_in.eof())
    cout << "Non esiste un'occorrenza della stringa specificata." << endl;
  else
    cout << "Esiste almeno un'occorrenza della stringa specificata." << endl;

  my_in.close();

  return(0);
}
```

1 Scrivere un programma che, dato un codice ed un file di testo contenente un numero indefinito di cifre, verifichi che nel file di testo specificato esista almeno un'occorenza del codice specificato. Il programma deve terminare la ricerca dopo aver trovato la prima occorrenza.

Al termine dell'esecuzione di questo programma, dovrà essere stampato a video un opportuno messaggio di avviso del tipo:
"Esiste almeno un'occorrenza del codice specificato"
oppure "Non esiste un'occorrenza del codice specificato"

Dato, ad esempio, in input il file `tabulato` contenente i seguenti dati:

```
114750157 451865 450159
682524769024 69237
287 87276247 798469
824597804724
```

se l'eseguibile è `a.out`, il comando

```
./a.out 727 tabulato
```

stamperà a video il seguente messaggio:
`Esiste almeno un'occorrenza del codice specificato`
perchè il codice 727 è contenuto nel file `tabulato`.

NOTA: è ammesso l'utilizzo delle funzioni di libreria `strstr` e `strcpy` della libreria `cstring`.

VALUTAZIONE: questo esercizio vale 6 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 **esercizio1.cc**

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstring>

using namespace std;

int main(int argc,char* argv[])
{
  fstream my_in;
  char tmp[100], target[100];
  int numero_parole=0, trovato, i;

  if (argc != 3) {
    cout << "Usage: ./a.out <codice> <tabulato>\n";
    exit(0);
  }
  strcpy(target,argv[1]);
  my_in.open(argv[2],ios::in);

  my_in >> tmp; //lettura prima parola
  while ((!my_in.eof())&&(!strstr(tmp,target))) {
    my_in >> tmp; //lettura parola successiva
  }

  if (my_in.eof())
    cout << "Non esiste un'occorrenza del codice specificato." << endl;
  else
    cout << "Esiste almeno un'occorrenza del codice specificato." << endl;

  my_in.close();

  return(0);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di interi `v` di dimensione `N` ed un intero positivo `J`, ritorni un nuovo vettore ottenuto dallo spostamento a sinistra di $J$ posizioni del contenuto del vettore `v` e dall'inserimento di uno zero nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [2, 17, 44, 202, 5, 13]$ e l'intero $J = 3$, la funzione `shift` deve ritornare il vettore $[202, 5, 13, 0, 0, 0]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.cc**

```
using namespace std;
#include <iostream>

// Inserire qui sotto la dichiarazione della funzione shift

int* shift(int v[], int n, int j);
void left_shift(int v1[], int v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 15;
   int vector[] = {2, 17, 44, 202, 5, 13, 26, 7, 9, 131, 51, 79, 88, 96, 32};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a sinistra: ";
   cin >> J;

   int* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


int* shift(int v[], int n, int j) {
   int *new_v = new int[n];
   left_shift(v, new_v, n, j, 0);
   return new_v;
}

void left_shift(int v1[], int v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i+j < n) {
         v2[i] = v1[i+j];
      } else {
```

```
            v2[i] = 0;
        }
        left_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di float `v` di dimensione `N` ed un intero positivo `J`, ritorni un nuovo vettore ottenuto dallo spostamento a destra di $J$ posizioni del contenuto del vettore `v` e dall'inserimento di uno zero nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [0.3, 5.5, 13.2, 21.2, 7.9]$ e l'intero $J = 2$, la funzione `shift` deve ritornare il vettore $[0.0, 0.0, 0.3, 5.5, 13.2]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.cc**

```
using namespace std;
#include <iostream>

// Inserire qui sotto la dichiarazione della funzione shift

float* shift(float v[], int n, int j);
void right_shift(float v1[], float v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 10;
   float vector[] = {5.0, 13.2, 26.5, 7.2, 10.7, 9.3, 14.0, 81.0, 65.9, 32.1};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a destra: ";
   cin >> J;

   float* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


float* shift(float v[], int n, int j) {
   float *new_v = new float[n];
   right_shift(v, new_v, n, j, 0);
   return new_v;
}

void right_shift(float v1[], float v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i < j) {
         v2[i] = 0.0;
      } else {
```

```
            v2[i] = v1[i-j];
        }
        right_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di caratteri minuscoli `v` di dimensione `N` ed un intero positivo `J`, ritorni un nuovo vettore ottenuto dallo spostamento a destra di $J$ posizioni del contenuto del vettore `v` e dall'inserimento di del carattere `'x'` nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [d, c, p, m, z, l]$ e l'intero $J = 2$, la funzione `shift` deve ritornare il vettore $[x, x, d, c, p, m]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.cc**

```
using namespace std;
#include <iostream>

// Inserire qui sotto la dichiarazione della funzione shift

char* shift(char v[], int n, int j);
void right_shift(char v1[], char v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 12;
   char vector[] = {'c', 'l', 'w', 'a', 'p', 'r', 't', 'm', 'q', 'e', 'y', 'v'};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a destra: ";
   cin >> J;

   char* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


char* shift(char v[], int n, int j) {
   char *new_v = new char[n];
   right_shift(v, new_v, n, j, 0);
   return new_v;
}

void right_shift(char v1[], char v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i < j) {
         v2[i] = 'x';
      } else {
```

```
            v2[i] = v1[i-j];
        }
        right_shift(v1, v2, n, j, i+1);
    }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione ricorsiva `shift` che, dato un vettore di caratteri maiuscoli `v` di dimensione `N` ed un intero positivo `J`, ritorni un nuovo vettore ottenuto dallo spostamento a sinistra di $J$ posizioni del contenuto del vettore `v` e dall'inserimento di del carattere `'Y'` nelle posizioni lasciate libere.

Per esempio, dato il vettore $v = [L, P, D, A, R, I]$ e l'intero $J = 4$, la funzione `shift` deve ritornare il vettore $[R, I, Y, Y, Y, Y]$.

NOTA 1: **La funzione `shift` deve essere ricorsiva: al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli.**

NOTA 2: all'interno di questo programma **è ammesso** l'utilizzo di funzioni ausiliarie purchè ricorsive e senza cicli o chiamate a funzioni contenenti cicli.

NOTA 3: all'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 **esercizio2.cc**

```
using namespace std;
#include <iostream>

// Inserire qui sotto la dichiarazione della funzione shift

char* shift(char v[], int n, int j);
void left_shift(char v1[], char v2[], int n, int j, int i);

int main(){

   int J = 0;
   const int N = 10;
   char vector[] = {'A', 'Z', 'E', 'I', 'K', 'P', 'N', 'F', 'D', 'J'};

   cout << "Array iniziale: ";
   for(int i=0; i<N; i++) {
      cout << vector[i] << " ";
   }
   cout << endl;

   cout << "Numero spostamenti a sinistra: ";
   cin >> J;

   char* new_vect = shift(vector, N, J);

   cout << "Nuovo array: ";
   for(int i=0; i<N; i++) {
      cout << new_vect[i] << " ";
   }
   cout << endl;

   delete[] new_vect;

return 0;
}

// Inserire qui sotto la definizione della funzione shift


char* shift(char v[], int n, int j) {
   char *new_v = new char[n];
   left_shift(v, new_v, n, j, 0);
   return new_v;
}

void left_shift(char v1[], char v2[], int n, int j, int i) {
   if (i >= n) {
      return;
   } else {
      if (i+j < n) {
         v2[i] = v1[i+j];
      } else {
```

```
            v2[i] = 'Y';
        }
        left_shift(v1, v2, n, j, i+1);
    }
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `int`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `empty` restituisca `FALSE` se la pila contiene almeno un valore, e `TRUE` altrimenti;
- `push` inserisca il valore passato come parametro nella pila;
- `pop` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

## 3 stack_main.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    int val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : push\n"
             << "o : pop\n"
             << "t : top\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            push(s, val);
            break;
          case 'o':
            if (! pop(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Pop ok!\n";
            break;
          case 't':
            if (! top(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "Top: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

### 3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  int val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
void   push   (stack &s, int val);
retval pop    (stack &s);
retval empty  (const stack &s);
retval top    (const stack &s, int &result);
void   print  (const stack &s);

#endif // STACK_H
```

### 3 stack.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval empty (const stack &s)
{
    return (s == NULL ? TRUE : FALSE);
}

void push (stack &s, int  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval pop (stack &s)
{
    if (empty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval top (const stack &s, int &result)
{
    if (empty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `long`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `nempty` restituisca `TRUE` se la pila contiene almeno un valore, e `FALSE` altrimenti;
- `add` inserisca il valore passato come parametro nella pila;
- `shrink` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

## 3 stack_main.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    long val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : add\n"
             << "o : shrink\n"
             << "t : first\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            add(s, val);
            break;
          case 'o':
            if (! shrink(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Shrink ok!\n";
            break;
          case 't':
            if (! first(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "First: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

### 3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  long val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
retval shrink (stack &s);
void   add    (stack &s, long val);
retval first  (const stack &s, long &result);
void   print  (const stack &s);
retval nempty (const stack &s);

#endif // STACK_H
```

### 3 stack.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval nempty (const stack &s)
{
    return (s == NULL ? FALSE : TRUE);
}

void add (stack &s, long  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval shrink (stack &s)
{
    if (!nempty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval first (const stack &s, long &result)
{
    if (!nempty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di caratteri `char`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `nempty` restituisca `TRUE` se la pila contiene almeno un carattere, e `FALSE` altrimenti;
- `push` inserisca il carattere passato come parametro nella pila;
- `pop` elimini il carattere in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `top` legga il carattere in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i caratteri contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

## 3 stack_main.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    char val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : push\n"
             << "o : pop\n"
             << "t : top\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            push(s, val);
            break;
          case 'o':
            if (! pop(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Pop ok!\n";
            break;
          case 't':
            if (! top(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "Top: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

### 3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  char val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
void   push   (stack &s, char val);
retval pop    (stack &s);
retval nempty (const stack &s);
retval top    (const stack &s, char &result);
void   print  (const stack &s);

#endif // STACK_H
```

### 3 stack.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval nempty (const stack &s)
{
    return (s == NULL ? FALSE : TRUE);
}

void push (stack &s, char val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval pop (stack &s)
{
    if (!nempty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval top (const stack &s, char &result)
{
    if (!nempty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menù per gestire una pila di numeri `double`. Scrivere, in un nuovo file `stack.cc`, le definizioni delle funzioni dichiarate nello header file `stack.h` in modo tale che:

- `init` inizializzi la pila;
- `deinit` liberi la memoria utilizzata dalla pila;
- `empty` restituisca `FALSE` se la pila contiene almeno un valore, e `TRUE` altrimenti;
- `add` inserisca il valore passato come parametro nella pila;
- `shrink` elimini il valore in testa alla pila, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `first` legga il valore in testa alla pila e lo memorizzi nella variabile passata come parametro, restituendo `TRUE` se l'operazione è andata a buon fine, e `FALSE` altrimenti;
- `print` stampi a video il contenuto della pila, nell'ordine in cui i valori contenuti ne verrebbero estratti.

La pila deve essere implementata internamente come una lista concatenata i cui elementi sono allocati dinamicamente.

N.B.: Ad eccezione delle operazioni `print` e `deinit`, si ricorda che ogni altra operazione sulla pila deve avere costo costante, indipendentemente dal numero di valori contenuti nella struttura dati.

VALUTAZIONE: questo esercizio vale 7 punti (al punteggio di tutti gli esercizi va poi sommato 10).

## 3 stack_main.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


int main ()
{
    char scelta;
    stack s;

    double val;

    init(s);

    do
    {
        cout << "Operazioni possibili:\n"
             << "u : add\n"
             << "o : shrink\n"
             << "t : first\n"
             << "p : print\n"
             << "e : esci\n"
             << "Che operazione vuoi eseguire? ";
        cin >> scelta;
        switch (scelta) {
          case 'u':
            cout << "Valore da inserire? ";
            cin >> val;
            add(s, val);
            break;
          case 'o':
            if (! shrink(s))
                cout << "Stack vuoto!\n";
            else
                cout << "Shrink ok!\n";
            break;
          case 't':
            if (! first(s, val))
                cout << "Stack vuoto!\n";
            else
                cout << "First: " << val << endl;
            break;
          case 'p':
            cout << "Contenuto dello stack: ";
            print(s);
            break;
          default:
            if (scelta != 'e') {
                cout << "Operazione non valida: " << scelta << endl;
            }
        }
```

```
    } while (scelta != 'e');

    deinit(s);

    return 0;
}
```

### 3 stack.h

```
#ifndef STACK_H
#define STACK_H

struct node {
  double val;
  node *next;
};

typedef node *stack;

enum retval {FALSE = 0, TRUE = 1};

void   init   (stack &s);
void   deinit (stack &s);
retval shrink (stack &s);
void   add    (stack &s, double val);
retval first  (const stack &s, double &result);
void   print  (const stack &s);
retval empty  (const stack &s);

#endif // STACK_H
```

### 3 stack.cc

```
using namespace std;
#include <iostream>

#include "stack.h"


void init (stack &s)
{
    s = NULL;
}

void deinit (stack &s)
{
    node *n = s;
    while (n != NULL)
    {
      node *tmp = n;
      n = n->next;
      delete tmp;
    }
```

```
    s = NULL;
}

retval empty (const stack &s)
{
    return (s == NULL ? TRUE : FALSE);
}

void add (stack &s, double  val)
{
    node *n = new node;

    n->val = val;
    n->next = s;
    s = n;
}

retval shrink (stack &s)
{
    if (empty(s))
      return FALSE;

    node *first = s;
    s = s->next;
    delete first;

    return TRUE;
}

retval first (const stack &s, double &result)
{
    if (empty(s))
      return FALSE;

    result = s->val;
    return TRUE;
}

void print (const stack &s)
{
    node *n = s;
    while (n != NULL)
    {
        cout << n->val << " ";
        n = n->next;
    }
    cout << endl;
}
```