Terzo Appello di Programmazione I

08 Giugno 2011
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 Uno dei metodi più semplici per criptare un file di testo consiste nell'eseguire operazioni aritmetiche sul codice ASCII di ciascun carattere presente nel file originario restituendo in questo modo un nuovo codice ASCII per ogni carattere.

Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file, copi il testo contenuto nel primo file nel secondo, "criptando" il contenuto del testo presente nel primo file. Come algoritmo di criptazione si richiede di sommare `3` al valore ASCII di ciascun carattere presente nel file originario.

Ad esempio, dato il file `testo` con questo contenuto:

```
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 testo testo_criptato`
creerà un nuovo file di nome `testo_criptato` con questo contenuto:

```
Ilodvwurffd#ghooh#sduroh=#Idwhyl#dydqwl$#Fkl#qh#yxrohB
Gl#sduroh#kr#od#whvwd#slhqd/#frq#ghqwur#od#%oxqd%#h#od#%edohqd%1
Fl#vrqr#sduroh#shu#jol#dplfl=#Exrq#jlruqr/#Exrq#dqqr/#Vldwh#iholfl$
Sduroh#ehooh#h#sduroh#exrqh>#sduroh#shu#rjql#vruwd#gl#shuvrqh1
Gl#J1#Urgdul1
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più 255 caratteri.

Si utilizzi la seguente funzione della libreria `string`:

```
stream_in.getline(char str[], int num_char);
```

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

1 **esercizio1.cc**

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

int main(int argc,char* argv[]){

  fstream my_in, my_out;

  char tmp[256];

  if (argc != 3) {
    cout << "Usage: ./a.out <testo> <testo_criptato>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in.getline(tmp,256);
  while (!my_in.eof()) {

    for (int i=0; i < strlen(tmp); i++) {
      if (tmp[i]!='\0') {
tmp[i]=tmp[i]+3;
      }
    }
    if (strlen(tmp)>0)
      my_out << tmp <<endl;
    my_in.getline(tmp,256);
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Uno dei metodi più semplici per criptare un file di testo consiste nell'eseguire operazioni aritmetiche sul codice ASCII di ciascun carattere presente nel file originario restituendo in questo modo un nuovo codice ASCII per ogni carattere.

Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file, copi il testo contenuto nel primo file nel secondo, "criptando" il contenuto del testo presente nel primo file. Come algoritmo di criptazione si richiede di sommare `7` al valore ASCII di ciascun carattere presente nel file originario.

Ad esempio, dato il file `testo` con questo contenuto:

```
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 testo testo_criptato`
creerà un nuovo file di nome `testo_criptato` con questo contenuto:

```
Mpshz{yvjjh'klssl'whyvslA'Mh{l}p'h}hu{p('Jop'ul'}|vslF
Kp'whyvsl'ov'sh'{lz{h'wpluh3'jvu'klu{yv'sh')s|uh)'l'sh')ihsluh)5
Jp'zvuv'whyvsl'wly'nsp'htpjpA'I|vu'npvyuv3'I|vu'huuv3'Zph{l'mlspjp(
Whyvsl'ilssl'l'whyvsl'i|vulB'whyvsl'wly'vnup'zvy{h'kp'wlyzvul5
Kp'N5'Yvkhyp5
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più 255 caratteri.

Si utilizzi la seguente funzione della libreria `string`:

```
stream_in.getline(char str[], int num_char);
```

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

1 **esercizio1.cc**

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

int main(int argc,char* argv[]){

  fstream my_in, my_out;

  char tmp[256];

  if (argc != 3) {
    cout << "Usage: ./a.out <testo> <testo_criptato>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in.getline(tmp,256);
  while (!my_in.eof()) {

    for (int i=0; i < strlen(tmp); i++) {
      if (tmp[i]!='\0') {
tmp[i]=tmp[i]+7;
      }
    }
    if (strlen(tmp)>0)
      my_out << tmp <<endl;
    my_in.getline(tmp,256);
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Uno dei metodi più semplici per criptare un file di testo consiste nell'eseguire operazioni aritmetiche sul codice ASCII di ciascun carattere presente nel file originario restituendo in questo modo un nuovo codice ASCII per ogni carattere.

Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file, copi il testo contenuto nel primo file nel secondo, "criptando" il contenuto del testo presente nel primo file. Come algoritmo di criptazione si richiede di sommare `5` al valore ASCII di ciascun carattere presente nel file originario.

Ad esempio, dato il file `testo` con questo contenuto:

```
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 testo testo_criptato`
creerà un nuovo file di nome `testo_criptato` con questo contenuto:

```
Knqfxywthhf%ijqqj%ufwtqj?%Kfyj{n%f{fsyn&%Hmn%sj%{ztqjD
In%ufwtqj%mt%qf%yjxyf%unjsf1%hts%ijsywt%qf%'qzsf'%j%qf%'gfqjsf'3
Hn%xtst%ufwtqj%ujw%lqn%frnhn?%Gzts%lntwst1%Gzts%fsst1%Xnfyj%kjqnhn&
Ufwtqj%gjqqj%j%ufwtqj%gztsj@%ufwtqj%ujw%tlsn%xtwyf%in%ujwxtsj3
In%L3%Wtifwn3
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più 255 caratteri.

Si utilizzi la seguente funzione della libreria `string`:

```
stream_in.getline(char str[], int num_char);
```

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

1 **esercizio1.cc**

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

int main(int argc,char* argv[]){

  fstream my_in, my_out;

  char tmp[256];

  if (argc != 3) {
    cout << "Usage: ./a.out <testo> <testo_criptato>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in.getline(tmp,256);
  while (!my_in.eof()) {

    for (int i=0; i < strlen(tmp); i++) {
      if (tmp[i]!='\0') {
tmp[i]=tmp[i]+5;
      }
    }
    if (strlen(tmp)>0)
      my_out << tmp <<endl;
    my_in.getline(tmp,256);
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

1 Uno dei metodi più semplici per criptare un file di testo consiste nell'eseguire operazioni aritmetiche sul codice ASCII di ciascun carattere presente nel file originario restituendo in questo modo un nuovo codice ASCII per ogni carattere.

Scrivere nel file `esercizio1.cc` un programma che, presi come argomenti del `main` i nomi di due file, copi il testo contenuto nel primo file nel secondo, "criptando" il contenuto del testo presente nel primo file. Come algoritmo di criptazione si richiede di sommare `6` al valore ASCII di ciascun carattere presente nel file originario.

Ad esempio, dato il file `testo` con questo contenuto:

```
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 testo testo_criptato`
creerà un nuovo file di nome `testo_criptato` con questo contenuto:

```
Lorgyzxuiig&jkrrk&vgxurk@&Lgzk|o&g|gtzo'&Ino&tk&|{urkE
Jo&vgxurk&nu&rg&zkyzg&voktg2&iut&jktzxu&rg&(r{tg(&k&rg&(hgrktg(4
Io&yutu&vgxurk&vkx&mro&gsoio@&H{ut&mouxtu2&H{ut&gttu2&Yogzk&lkroio'
Vgxurk&hkrrk&k&vgxurk&h{utkA&vgxurk&vkx&umto&yuxzg&jo&vkxyutk4
Jo&M4&Xujgxo4
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più 255 caratteri.

Si utilizzi la seguente funzione della libreria `string`:

```
stream_in.getline(char str[], int num_char);
```

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

1 **esercizio1.cc**

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

int main(int argc,char* argv[]){

  fstream my_in, my_out;

  char tmp[256];

  if (argc != 3) {
    cout << "Usage: ./a.out <testo> <testo_criptato>\n";
    exit(0);
  }

  my_in.open(argv[1],ios::in);
  my_out.open(argv[2],ios::out);

  my_in.getline(tmp,256);
  while (!my_in.eof()) {

    for (int i=0; i < strlen(tmp); i++) {
      if (tmp[i]!='\0') {
tmp[i]=tmp[i]+6;
      }
    }
    if (strlen(tmp)>0)
      my_out << tmp <<endl;
    my_in.getline(tmp,256);
  }

  my_in.close();
  my_out.close();
  return(0);
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `calcola_somma` che, presi come parametri due numeri interi `a` e `b` (tali che `a` $\leq$ `b`) e un terzo parametro intero `res`, restituisca tramite `res` la somma di tutti gli interi compresi tra `a` e `b`. Per esempio, dati due interi 2 e 5 la funzione `calcola_somma` calcola $2 + 3 + 4 + 5$ e tramite `res` restituisce alla funzione chiamante l'intero 14.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

//Inserire qui la dichiarazione della funzione calcola_somma

void calcola_somma(int a, int b, int &res);

int main() {

   int res = 0;
   int x1, x2;

   do {

      cout << "Inserisci primo numero: ";
      cin >> x1;

      cout << "Inserisci secondo numero: ";
      cin >> x2;

   } while(x1 > x2);

   calcola_somma(x1, x2, res);

   cout << "Sommatoria dei numeri compresi tra "<<x1 << " e "<< x2 << " = ";
   cout << res << endl;

   return 0;
}

//Inserire qui la definizione della funzione calcola_somma

void calcola_somma(int a, int b, int &res) {
   if (a==b) {
      res = a;
   } else {
      calcola_somma(a+1, b, res);
      res += a;
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `calcola_somma` che, presi come parametri due numeri interi `a` e `b` di tipo `long` (tali che `a` $\leq$ `b`) e un terzo parametro intero `res` di tipo `long`, restituisca tramite `res` la somma di tutti gli interi compresi tra `a` e `b`. Per esempio, dati due interi 5 e 9 la funzione `calcola_somma` calcola $5+6+7+8+9$ e tramite `res` restituisce alla funzione chiamante l'intero 35.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

//Inserire qui la dichiarazione della funzione calcola_somma

void calcola_somma(long a, long b, long &res);

int main() {

   long res = 0;
   long x1, x2;

   do {

      cout << "Inserisci primo numero: ";
      cin >> x1;

      cout << "Inserisci secondo numero: ";
      cin >> x2;

   } while(x1 > x2);

   calcola_somma(x1, x2, res);

   cout << "Sommatoria dei numeri compresi tra "<<x1 << " e "<< x2 << " = ";
   cout << res << endl;

   return 0;
}

//Inserire qui la definizione della funzione calcola_somma

void calcola_somma(long a, long b, long &res) {
   if (a==b) {
      res = a;
   } else {
      calcola_somma(a+1, b, res);
      res += a;
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `calcola_prodotto` che, presi come parametri due numeri interi `a` e `b` di tipo `long` (tali che `a` $\leq$ `b`) e un terzo parametro intero `res` di tipo `long`, restituisca tramite `res` il prodotto di tutti gli interi compresi tra `a` e `b`. Per esempio, dati due interi 3 e 6 la funzione `calcola_prodotto` calcola $3 * 4 * 5 * 6$ e tramite `res` restituisce alla funzione chiamante l'intero 360.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

//Inserire qui la dichiarazione della funzione calcola_prodotto

void calcola_prodotto(long a, long b, long &res);

int main() {

   long res = 0;
   long x1, x2;

   do {

      cout << "Inserisci primo numero: ";
      cin >> x1;

      cout << "Inserisci secondo numero: ";
      cin >> x2;

   } while(x1 > x2);

   calcola_prodotto(x1, x2, res);

   cout << "Prodotto dei numeri compresi tra "<<x1 << " e "<< x2 << " = ";
   cout << res << endl;

   return 0;
}

//Inserire qui la definizione della funzione calcola_prodotto

void calcola_prodotto(long a, long b, long &res) {
   if (a==b) {
      res = a;
   } else {
      calcola_prodotto(a+1, b, res);
      res *= a;
   }
}
```

2 Scrivere nel file `esercizio2.cc` la dichiarazione e la definizione della funzione **ricorsiva** `calcola_prodotto` che, presi come parametri due numeri interi `a` e `b` (tali che `a` $\leq$ `b`) e un terzo parametro intero `res`, restituisca tramite `res` il prodotto di tutti gli interi compresi tra `a` e `b`. Per esempio, dati due interi 4 e 7 la funzione `calcola_prodotto` calcola $4 * 5 * 6 * 7$ e tramite `res` restituisce alla funzione chiamante l'intero 840.

NOTA: **La funzione deve essere ricorsiva ed al suo interno non ci possono essere cicli o chiamate a funzioni contenenti cicli. Può fare uso di eventuali funzioni ausiliarie purchè a loro volta ricorsive.**

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

2 **esercizio2.cc**

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

//Inserire qui la dichiarazione della funzione calcola_prodotto

void calcola_prodotto(int a, int b, int &res);

int main() {

   int res = 0;
   int x1, x2;

   do {

      cout << "Inserisci primo numero: ";
      cin >> x1;

      cout << "Inserisci secondo numero: ";
      cin >> x2;

   } while(x1 > x2);

   calcola_prodotto(x1, x2, res);

   cout << "Prodotto dei numeri compresi tra "<<x1 << " e "<< x2 << " = ";
   cout << res << endl;

   return 0;
}

//Inserire qui la definizione della funzione calcola_prodotto

void calcola_prodotto(int a, int b, int &res) {
   if (a==b) {
      res = a;
   } else {
      calcola_prodotto(a+1, b, res);
      res *= a;
   }
}
```

3 Nel file `esercizio3.cc` è presente la funzione `main` che prende da standard input la dimensione di una matrice (i.e. numero di righe `row` e colonne `col`) ed invoca le funzioni `crea_matrice` e `stampa_matrice` per rispettivamente creare (i.e. allocare ed inizializzare) e stampare una matrice. Sono inoltre dati l'header file `stampa.h` ed il file binario `stampa.o` contenenti la funzione `stampa_matrice` usata nel `main`. Scrivere nel file `esercizio3.cc` la definizione delle seguenti funzioni:

- `fatt` che, preso come parametro un `int` n, calcola e restituisce un `double` pari a $n!$ (dove $n!$ indica il fattoriale di n);
- `crea_matrice` che, preso come parametro un puntatore a puntatore a double `matrix` ed il numero di righe `row` e colonne `col` di una matrice, allochi dinamicamente ed inizializzi una matrice `row` $\times$ `col`. In particolare, ogni elemento della matrice `matrix[i][j]` deve essere pari a $i! + j!$.

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

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

3 esercizio3.cc

```
#include <iostream>
using namespace std;
#include "stampa.h"

double fatt(int n);
void crea_matrice(double** &matrix, int r, int c);

int main() {

   int row, col;
   double** A;

   do {
      cout << "Inserisci numero di righe: ";
      cin >> row;
   } while (row <= 0);

   do {
      cout << "Inserisci numero di colonne: ";
      cin >> col;
   } while (col <= 0);

   crea_matrice(A, row, col);
   cout << "Matrice: " << endl;
   stampa_matrice(A, row, col);

   return 0;
}

// Inserire qui la definizione delle funzioni fatt e crea_matrice

double fatt(int n){
   if (n==0){
      return 1.0;
   } else {
      return n*fatt(n-1);
   }
}

void crea_matrice(double** &matrix, int r, int c) {

   matrix = new double*[r];

   for (int i = 0; i < r; i++) {
      matrix[i] = new double[c];
   }

   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         matrix[i][j] = fatt(i) + fatt(j);
      }
   }
```

```
}
```

3 Nel file `esercizio3.cc` è presente la funzione `main` che prende da standard input la dimensione di una matrice (i.e. numero di righe `row` e colonne `col`) ed invoca le funzioni `crea_matrice` e `stampa_matrice` per rispettivamente creare (i.e. allocare ed inizializzare) e stampare una matrice. Sono inoltre dati l'header file `stampa.h` ed il file binario `stampa.o` contenenti la funzione `stampa_matrice` usata nel `main`. Scrivere nel file `esercizio3.cc` la definizione delle seguenti funzioni:

- `fatt` che, preso come parametro un `int` n, calcola e restituisce un `double` pari a $n!$ (dove $n!$ indica il fattoriale di n);
- `crea_matrice` che, preso come parametro un puntatore a puntatore a double `matrix` ed il numero di righe `row` e colonne `col` di una matrice, allochi dinamicamente ed inizializzi una matrice `row` $\times$ `col`. In particolare, ogni elemento della matrice `matrix[i][j]` deve essere pari a $i + j!$.

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

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

3 esercizio3.cc

```
#include <iostream>
using namespace std;
#include "stampa.h"

double fatt(int n);
void crea_matrice(double** &matrix, int r, int c);

int main() {

   int row, col;
   double** A;

   do {
      cout << "Inserisci numero di righe: ";
      cin >> row;
   } while (row <= 0);

   do {
      cout << "Inserisci numero di colonne: ";
      cin >> col;
   } while (col <= 0);

   crea_matrice(A, row, col);
   cout << "Matrice: " << endl;
   stampa_matrice(A, row, col);

   return 0;
}

// Inserire qui la definizione delle funzioni fatt e crea_matrice

double fatt(int n){
   if (n==0){
      return 1.0;
   } else {
      return n*fatt(n-1);
   }
}

void crea_matrice(double** &matrix, int r, int c) {

   matrix = new double*[r];

   for (int i = 0; i < r; i++) {
      matrix[i] = new double[c];
   }

   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         matrix[i][j] = i + fatt(j);
      }
   }
```

```
}
```

3 Nel file `esercizio3.cc` è presente la funzione `main` che prende da standard input la dimensione di una matrice (i.e. numero di righe `row` e colonne `col`) ed invoca le funzioni `crea_matrice` e `stampa_matrice` per rispettivamente creare (i.e. allocare ed inizializzare) e stampare una matrice. Sono inoltre dati l'header file `stampa.h` ed il file binario `stampa.o` contenenti la funzione `stampa_matrice` usata nel `main`. Scrivere nel file `esercizio3.cc` la definizione delle seguenti funzioni:

- `power` che, preso come parametro un `int` n, calcola e restituisce un `double` pari a 2^n;
- `crea_matrice` che, preso come parametro un puntatore a puntatore a double `matrix` ed il numero di righe `row` e colonne `col` di una matrice, allochi dinamicamente ed inizializzi una matrice `row` $\times$ `col`. In particolare, ogni elemento della matrice `matrix[i][j]` deve essere pari a $2^i + 2^j$.

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

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

3 esercizio3.cc

```
#include <iostream>
using namespace std;
#include "stampa.h"

double power(int n);
void crea_matrice(double** &matrix, int r, int c);

int main() {

   int row, col;
   double** A;

   do {
      cout << "Inserisci numero di righe: ";
      cin >> row;
   } while (row <= 0);

   do {
      cout << "Inserisci numero di colonne: ";
      cin >> col;
   } while (col <= 0);

   crea_matrice(A, row, col);
   cout << "Matrice: " << endl;
   stampa_matrice(A, row, col);

   return 0;
}

// Inserire qui la definizione delle funzioni power e crea_matrice

double power(int n){
   if (n==0){
      return 1.0;
   } else {
      return 2.0*power(n-1);
   }
}

void crea_matrice(double** &matrix, int r, int c) {

   matrix = new double*[r];

   for (int i = 0; i < r; i++) {
      matrix[i] = new double[c];
   }

   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         matrix[i][j] = power(i) + power(j);
      }
   }
```

```
}
```

3 Nel file `esercizio3.cc` è presente la funzione `main` che prende da standard input la dimensione di una matrice (i.e. numero di righe `row` e colonne `col`) ed invoca le funzioni `crea_matrice` e `stampa_matrice` per rispettivamente creare (i.e. allocare ed inizializzare) e stampare una matrice. Sono inoltre dati l'header file `stampa.h` ed il file binario `stampa.o` contenenti la funzione `stampa_matrice` usata nel `main`. Scrivere nel file `esercizio3.cc` la definizione delle seguenti funzioni:

- `power` che, preso come parametro un `int` n, calcola e restituisce un `double` pari a 2^n;
- `crea_matrice` che, preso come parametro un puntatore a puntatore a double `matrix` ed il numero di righe `row` e colonne `col` di una matrice, allochi dinamicamente ed inizializzi una matrice `row` $\times$ `col`. In particolare, ogni elemento della matrice `matrix[i][j]` deve essere pari a $2^i + j$.

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

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

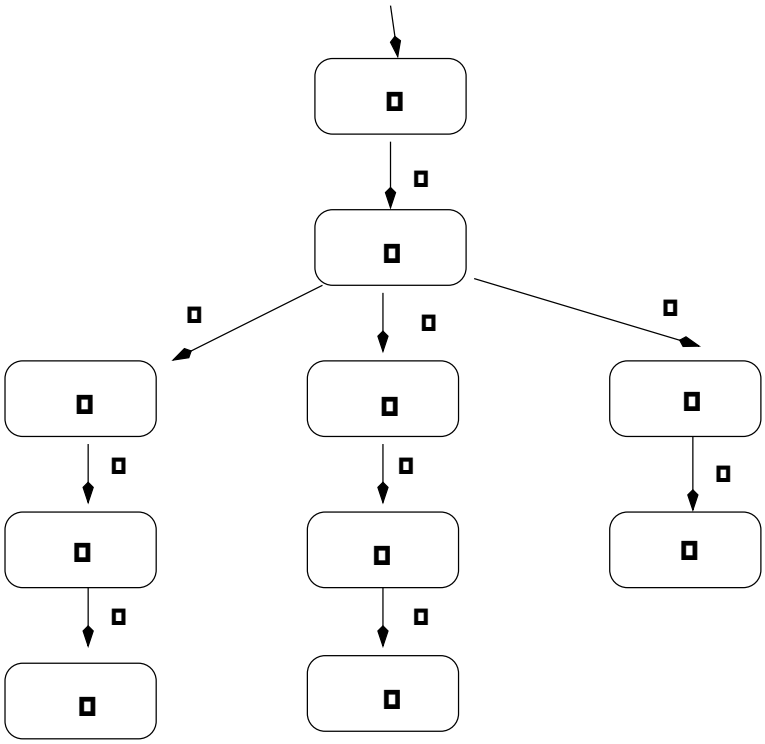
3 esercizio3.cc

```
#include <iostream>
using namespace std;
#include "stampa.h"

double power(int n);
void crea_matrice(double** &matrix, int r, int c);

int main() {

   int row, col;
   double** A;

   do {
      cout << "Inserisci numero di righe: ";
      cin >> row;
   } while (row <= 0);

   do {
      cout << "Inserisci numero di colonne: ";
      cin >> col;
   } while (col <= 0);

   crea_matrice(A, row, col);
   cout << "Matrice: " << endl;
   stampa_matrice(A, row, col);

   return 0;
}

// Inserire qui la definizione delle funzioni power e crea_matrice

double power(int n){
   if (n==0){
      return 1.0;
   } else {
      return 2.0*power(n-1);
   }
}

void crea_matrice(double** &matrix, int r, int c) {

   matrix = new double*[r];

   for (int i = 0; i < r; i++) {
      matrix[i] = new double[c];
   }

   for (int i = 0; i < r; i++) {
      for (int j = 0; j < c; j++) {
         matrix[i][j] = power(i) + j;
      }
   }
```

```
}
```

4 Un wordtree è una struttura dati che consente di memorizzare un gran numero di occorrenze di parole minuscole e recuperarle in modo estremamente efficiente.
Consiste in un albero di ricerca in cui ogni nodo ha 26 sottoalberi figli (uno per ogni carattere minuscolo) e contiene un intero che rappresenta il numero di occorrenze della parola rappresentata dal ramo di cui il nodo e' l'elemento terminale.

Ad esempio, la sequenza di parole `bidi bi bodi bi bum`, è rappresentata dal wordtree in figura.



Sono dati:

- il file `wordtree.h`, contenente la definizione delle strutture dati e gli header delle funzioni
- il file binario `wordtree.o`, contenente la funzione `stampa_wordtree` usata nel main
- il file `esercizio4.cc`, contenente la funzione main().

Definire in `esercizio4.cc` le funzioni `aggiungi_parola`, che aggiunge una nuova parola al wordtree, e `occorrenze_parola`, che resturuisce il numero di occorrenze di parola in wordtree.

Esempio: se il file `in` contiene il testo:

```
bimbo ba bi bo bu bimba
bi ba ba ba bumba
bidi bodi bim bum bam
```

il programma darà la seguente esecuzione:

```
>./a.out < in
bimbo occorre 1 volta
ba occorre 1 volta
bi occorre 1 volta
bo occorre 1 volta
bu occorre 1 volta
bimba occorre 1 volta
bi occorre 2 volte
ba occorre 2 volte
ba occorre 3 volte
ba occorre 4 volte
bumba occorre 1 volta
bidi occorre 1 volta
bodi occorre 1 volta
bim occorre 1 volta
bum occorre 1 volta
bam occorre 1 volta

WORDTREE:

ba: 4
bam: 1
bi: 2
bidi: 1
bim: 1
bimba: 1
bimbo: 1
bo: 1
bodi: 1
bu: 1
bum: 1
bumba: 1
```

VALUTAZIONE: questo esercizio permette di conseguire la lode se tutti gli esercizi precedenti sono corretti.

4 esercizio4.cc

```
using namespace std;

#include <iostream>
#include <cstdio>
#include <cstring>

#include "wordtree.h"

int main() {
  char prefix[MAXLENGTH]= "";
  wordtree t;
  int occ;
  char parola [MAXLENGTH];
  init(t);
  while (cin >>  parola) {
    aggiungi_parola(parola,t);
    occ = occorrenze_parola(parola,t);
    cout << parola << " occorre " << occ
         << ((occ==1) ? " volta\n" : " volte\n");
  }
  cout << "\nWORDTREE: \n\n";
  stampa_wordtree(t,prefix);
  return 0;
}

// Definire qui sotto init, aggiungi_parola e occorrenze_parola

void init (wordtree & t) {
  t = NULL;
}


int index(char c) {
  return (c-'a');
}

void aggiungi_parola(char * parola, wordtree & t) {
  int res;
  char c = *parola;
  if (t==NULL) {
    t = new node;
    for (int i=0;i<DIM;i++)
      t->subtree[i]=NULL;
    t->occorrenze=0;
  }
  if (c =='\0')
    t->occorrenze++;
  else {
    parola++;
    aggiungi_parola(parola, t->subtree[index(c)]);
  }
}
```

```
int occorrenze_parola(char * parola, wordtree & t) {
  int res;
  char c = *parola;
  if (t==NULL)
    res=0;
  else if (c ==’\0’)
    res = t-> occorrenze;
  else {
    parola++;
    res= occorrenze_parola(parola,t->subtree[index(c)]);
  }
  return res;
}
```