

Primo Appello di Programmazione I

12 Gennaio 2016
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 Si implementi nel file **esercizio1.cc** un programma che, presi in ingresso come parametri da riga di comando:

- il nome di un file di testo di input;
- un numero intero “**numero_righe**”;
- il nome di un file di testo di output;

legga le prime “**numero_righe**” righe del file di input e le salvi nel file di output, **invertendo l'ordine originale e salvando solo le righe dispari** (si assuma che la prima riga del file sia la numero “0”).

Ad esempio, dato il file **input.txt** con questo contenuto:

```
La donzelletta vien  
dalla campagna  
in sul calar del sole  
col suo fascio dell'erba  
e reca in mano  
un mazzolin di rose e viole.
```

Se l'eseguibile è **a.out**, il comando

```
./a.out input.txt 5 output.txt
```

creerà un nuovo file di nome **output.txt** con questo contenuto:

```
col suo fascio dell'erba  
dalla campagna
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più **80** caratteri.

Si ricorda che esiste la funzione della libreria **cstdlib**:

```
int atoi(char str[]);
```

che interpreta la stringa passata in ingresso come un numero intero e ne ritorna il valore in uscita.

NOTA 1: non e' ammesso aprire gli stream di ingresso e di uscita più di una volta durante l'esecuzione del codice, e la lettura del file di input deve avvenire in modo sequenziale.

NOTA 2: e' possibile utilizzare le funzioni della libreria “cstring” (quali, ad esempio, **strlen** o **strcmp**) e quelle della libreria “fstream” (quali, ad esempio, **getline** o **open**).

NOTA 3: non e' ammesso l'utilizzo di salti non condizionati (come ad esempio, ma non solo, istruzioni **break** al di fuori del costrutto **switch**, istruzioni **continue** e **return** nei cicli, eccetera).

NOTA 4: in caso si ritenga necessaria l'allocazione dinamica di variabili, si assuma che quest'ultima vada sempre a buon fine.

VALUTAZIONE: questo esercizio vale 8 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 soluzione_A11.cc

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

using namespace std;

// Spazio per il terminatore ('\0')
const int DIM_RIGA = 80 + 1;

int min(int a, int b);

int main(int argc, char* argv[]) {
    // Stream ingresso/uscita
    fstream my_in, my_out;
    char riga[DIM_RIGA];

    // Controllo parametri in ingresso
    if (argc != 4) {
        cerr << "Utilizzo: ./a.out <ingresso> <numero_righe> <uscita>\n";
        exit(-1);
    }

    // Apertura file in ingresso
    my_in.open(argv[1], ios::in);
    if (my_in.fail()) {
        cerr << "Apertura file " << argv[1] << " fallita.\n";
        exit(-1);
    }

    // Numero di righe da parametro da riga comando
    int numero_righe = atoi(argv[2]);
    // Allocazione dinamica
    char** righe = new char* [numero_righe];

    // Supponiamo che vada sempre a buon fine
    my_out.open(argv[3], ios::out);

    // Leggo la prima riga del file
    my_in.getline(riga, DIM_RIGA);
    int i;
    for(i = 0; !my_in.eof() && i < numero_righe; i++) {
        // Allocazione dinamica
        righe[i] = new char[strlen(riga) + 1];
        // "strncpy" non e' necessaria qui, riga non puo'
        // superare DIM_RIGA caratteri
        strcpy(righe[i], riga);
        // Leggo la riga successiva del file
        my_in.getline(riga, DIM_RIGA);
    }

    // Salvo le righe nel file di output
```

```

for(int j = min(i, numero_righe) - 1; j >= 0; j--) {
    // Salva solo righe dispari
    if((j % 2) == 1) {
        my_out << righe[j] << endl;
    }
    // Dealloco la riga gia' salvata
    delete [] righe[j];
}

// Chiusura stream
my_in.close();
my_out.close();
return(0);
}

int min(int a, int b) {
    return a < b ? a : b;
}

```

1 Si implementi nel file **esercizio1.cc** un programma che, presi in ingresso come parametri da riga di comando:

- il nome di un file di testo di input;
- un numero intero “**numero_righe**”;
- il nome di un file di testo di output;

legga le prime “**numero_righe**” righe del file di input e le salvi nel file di output, **invertendo l’ordine originale e salvando solo le righe pari** (si assuma che la prima riga del file sia la numero “0”).

Ad esempio, dato il file **input.txt** con questo contenuto:

```
La donzelletta vien  
dalla campagna  
in sul calar del sole  
col suo fascio dell’erba  
e reca in mano  
un mazzolin di rose e viole.
```

Se l’eseguibile è **a.out**, il comando

```
./a.out input.txt 5 output.txt
```

creerà un nuovo file di nome **output.txt** con questo contenuto:

```
e reca in mano  
in sul calar del sole  
La donzelletta vien
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più **80** caratteri.

Si ricorda che esiste la funzione della libreria **cstdlib**:

```
int atoi(char str[]);
```

che interpreta la stringa passata in ingresso come un numero intero e ne ritorna il valore in uscita.

NOTA 1: non e’ ammesso aprire gli stream di ingresso e di uscita più di una volta durante l’esecuzione del codice, e la lettura del file di input deve avvenire in modo sequenziale.

NOTA 2: e’ possibile utilizzare le funzioni della libreria “cstring” (quali, ad esempio, **strlen** o **strcmp**) e quelle della libreria “fstream” (quali, ad esempio, **getline** o **open**).

NOTA 3: non e’ ammesso l’utilizzo di salti non condizionati (come ad esempio, ma non solo, istruzioni **break** al di fuori del costrutto **switch**, istruzioni **continue** e **return** nei cicli, eccetera).

NOTA 4: in caso si ritenga necessaria l’allocazione dinamica di variabili, si assuma che quest’ultima vada sempre a buon fine.

VALUTAZIONE: questo esercizio vale 8 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 soluzione_A12.cc

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

using namespace std;

// Spazio per il terminatore ('\0')
const int DIM_RIGA = 80 + 1;

int min(int a, int b);

int main(int argc, char* argv[]) {
    // Stream ingresso/uscita
    fstream my_in, my_out;
    char riga[DIM_RIGA];

    // Controllo parametri in ingresso
    if (argc != 4) {
        cerr << "Utilizzo: ./a.out <ingresso> <numero_righe> <uscita>\n";
        exit(-1);
    }

    // Apertura file in ingresso
    my_in.open(argv[1], ios::in);
    if (my_in.fail()) {
        cerr << "Apertura file " << argv[1] << " fallita.\n";
        exit(-1);
    }

    // Numero di righe da parametro da riga comando
    int numero_righe = atoi(argv[2]);
    // Allocazione dinamica
    char** righe = new char* [numero_righe];

    // Supponiamo che vada sempre a buon fine
    my_out.open(argv[3], ios::out);

    // Leggo la prima riga del file
    my_in.getline(riga, DIM_RIGA);
    int i;
    for(i = 0; !my_in.eof() && i < numero_righe; i++) {
        // Allocazione dinamica
        righe[i] = new char[strlen(riga) + 1];
        // "strncpy" non e' necessaria qui, riga non puo'
        // superare DIM_RIGA caratteri
        strcpy(righe[i], riga);
        // Leggo la riga successiva del file
        my_in.getline(riga, DIM_RIGA);
    }

    // Salvo le righe nel file di output
```

```

for(int j = min(i, numero_righe) - 1; j >= 0; j--) {
    // Salva solo righe pari
    if((j % 2) == 0) {
        my_out << righe[j] << endl;
    }
    // Dealloca la riga gia' stampata
    delete [] righe[j];
}

// Chiusura stream
my_in.close();
my_out.close();
return(0);
}

int min(int a, int b) {
    return a < b ? a : b;
}

```

1 Si implementi nel file **esercizio1.cc** un programma che, presi in ingresso come parametri da riga di comando:

- il nome di un file di testo di input;
- un numero intero “**numero_righe**”;
- il nome di un file di testo di output;

legga le prime “**numero_righe**” righe del file di input e le salvi nel file di output, **invertendo l'ordine originale e salvando solo le righe che cominciano per vocale** (maiuscole o minuscole; ai fini dell'esercizio, “j” e “y” non sono vocali).

Ad esempio, dato il file **input.txt** con questo contenuto:

```
La donzelletta vien  
dalla campagna  
in sul calar del sole  
col suo fascio dell'erba  
e reca in mano  
un mazzolin di rose e viole.
```

Se l'eseguibile è **a.out**, il comando

```
./a.out input.txt 5 output.txt
```

creerà un nuovo file di nome **output.txt** con questo contenuto:

```
e reca in mano  
in sul calar del sole
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più **80** caratteri.

Si ricorda che esiste la funzione della libreria **cstdlib**:

```
int atoi(char str[]);
```

che interpreta la stringa passata in ingresso come un numero intero e ne ritorna il valore in uscita.

NOTA 1: non e' ammesso aprire gli stream di ingresso e di uscita più di una volta durante l'esecuzione del codice, e la lettura del file di input deve avvenire in modo sequenziale.

NOTA 2: e' possibile utilizzare le funzioni della libreria “cstring” (quali, ad esempio, **strlen** o **strcmp**) e quelle della libreria “fstream” (quali, ad esempio, **getline** o **open**).

NOTA 3: non e' ammesso l'utilizzo di salti non condizionati (come ad esempio, ma non solo, istruzioni **break** al di fuori del costrutto **switch**, istruzioni **continue** e **return** nei cicli, eccetera).

NOTA 4: in caso si ritenga necessaria l'allocazione dinamica di variabili, si assuma che quest'ultima vada sempre a buon fine.

VALUTAZIONE: questo esercizio vale 8 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 soluzione_A13.cc

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

using namespace std;

// Spazio per il terminatore ('\0')
const int DIM_RIGA = 80 + 1;
const char VOCALI[11] = "aAeEiIoOuU";

int min(int a, int b);
bool vocale(char c);

int main(int argc, char* argv[]) {
    // Stream ingresso/uscita
    fstream my_in, my_out;
    char riga[DIM_RIGA];

    // Controllo parametri in ingresso
    if (argc != 4) {
        cerr << "Utilizzo: ./a.out <ingresso> <numero_righe> <uscita>\n";
        exit(-1);
    }

    // Apertura file in ingresso
    my_in.open(argv[1], ios::in);
    if (my_in.fail()) {
        cerr << "Apertura file " << argv[1] << " fallita.\n";
        exit(-1);
    }

    // Numero di righe da parametro da riga comando
    int numero_righe = atoi(argv[2]);
    // Allocazione dinamica
    char** righe = new char* [numero_righe];

    // Supponiamo che vada sempre a buon fine
    my_out.open(argv[3], ios::out);

    // Leggo la prima riga del file
    my_in.getline(riga, DIM_RIGA);
    int i;
    for(i = 0; !my_in.eof() && i < numero_righe; i++) {
        // Allocazione dinamica
        righe[i] = new char[strlen(riga) + 1];
        // "strncpy" non e' necessaria qui, riga non puo'
        // superare DIM_RIGA caratteri
        strcpy(righe[i], riga);
        // Leggo la riga successiva del file
        my_in.getline(riga, DIM_RIGA);
    }
```

```

// Salvo le righe nel file di output
for(int j = min(i, numero_righe) - 1; j >= 0; j--) {
    // Salva solo righe che cominciano per vocale
    if(vocale(righe[j][0])) {
        my_out << righe[j] << endl;
    }
    // Dealloca la riga gia' salvata
    delete [] righe[j];
}

// Chiusura stream
my_in.close();
my_out.close();
return(0);
}

int min(int a, int b) {
    return a < b ? a : b;
}

bool vocale(char c) {
    return (strchr(VOCALI, c) != NULL);
}

```

1 Si implementi nel file **esercizio1.cc** un programma che, presi in ingresso come parametri da riga di comando:

- il nome di un file di testo di input;
- un numero intero “**numero_righe**”;
- il nome di un file di testo di output;

legga le prime “**numero_righe**” righe del file di input e le salvi nel file di output, **invertendo l’ordine originale e salvando solo le righe che cominciano per consonante** (maiuscole o minuscole; ai fini dell’esercizio, “j” e “y” sono consonanti).

Ad esempio, dato il file **input.txt** con questo contenuto:

```
La donzelletta vien  
dalla campagna  
in sul calar del sole  
col suo fascio dell’erba  
e reca in mano  
un mazzolin di rose e viole.
```

Se l’eseguibile è **a.out**, il comando

```
./a.out input.txt 5 output.txt
```

creerà un nuovo file di nome **output.txt** con questo contenuto:

```
col suo fascio dell’erba  
dalla campagna  
La donzelletta vien
```

Per semplicità si assuma che ogni riga del file di input abbia una lunghezza di al più **80** caratteri.

Si ricorda che esiste la funzione della libreria **cstdlib**:

```
int atoi(char str[]);
```

che interpreta la stringa passata in ingresso come un numero intero e ne ritorna il valore in uscita.

NOTA 1: non e’ ammesso aprire gli stream di ingresso e di uscita più di una volta durante l’esecuzione del codice, e la lettura del file di input deve avvenire in modo sequenziale.

NOTA 2: e’ possibile utilizzare le funzioni della libreria “cstring” (quali, ad esempio, **strlen** o **strcmp**) e quelle della libreria “fstream” (quali, ad esempio, **getline** o **open**).

NOTA 3: non e’ ammesso l’utilizzo di salti non condizionati (come ad esempio, ma non solo, istruzioni **break** al di fuori del costrutto **switch**, istruzioni **continue** e **return** nei cicli, eccetera).

NOTA 4: in caso si ritenga necessaria l’allocazione dinamica di variabili, si assuma che quest’ultima vada sempre a buon fine.

VALUTAZIONE: questo esercizio vale 8 punti (al punteggio di tutti gli esercizi va poi sommato 10).

1 soluzione_A14.cc

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

using namespace std;

// Spazio per il terminatore ('\0')
const int DIM_RIGA = 80 + 1;
// Elenco delle vocali
const char VOCALI[11] = "aAeEiIoOuU";

int min(int a, int b);
bool vocale(char c);

int main(int argc, char* argv[]) {
    // Stream ingresso/uscita
    fstream my_in, my_out;
    char riga[DIM_RIGA];

    // Controllo parametri in ingresso
    if (argc != 4) {
        cerr << "Utilizzo: ./a.out <ingresso> <numero_righe> <uscita>\n";
        exit(-1);
    }

    // Apertura file in ingresso
    my_in.open(argv[1], ios::in);
    if (my_in.fail()) {
        cerr << "Apertura file " << argv[1] << " fallita.\n";
        exit(-1);
    }

    // Numero di righe da parametro da riga comando
    int numero_righe = atoi(argv[2]);
    // Allocazione dinamica
    char** righe = new char* [numero_righe];

    // Supponiamo che vada sempre a buon fine
    my_out.open(argv[3], ios::out);

    // Leggo la prima riga del file
    my_in.getline(riga, DIM_RIGA);
    int i;
    for(i = 0; !my_in.eof() && i < numero_righe; i++) {
        // Allocazione dinamica
        righe[i] = new char[strlen(riga) + 1];
        // "strncpy" non e' necessaria qui, riga non puo'
        // superare DIM_RIGA caratteri
        strcpy(righe[i], riga);
        // Leggo la riga successiva del file
        my_in.getline(riga, DIM_RIGA);
    }
```

```

}

// Salvo le righe nel file di output
for(int j = min(i, numero_righe) - 1; j >= 0; j--) {
    // Salva solo righe che cominciano per consonante
    if(!vocale(righe[j][0])) {
        my_out << righe[j] << endl;
    }
    // Dealloco la riga gia' salvata
    delete [] righe[j];
}

// Chiusura stream
my_in.close();
my_out.close();
return(0);
}

int min(int a, int b) {
    return a < b ? a : b;
}

bool vocale(char c) {
    return (strchr(VOCALI, c) != NULL);
}

```

2 Completare il programma contenuto nel file `esercizio2.cc` implementando la **funzione ricorsiva**

```
void somma_array(int n1[], int n2[], int risultato[], int dim)
```

che, dati in ingresso due array `n1` e `n2`, ne calcoli la somma e la memorizzi nell'array `risultato`: i tre array sono di uguale dimensione `dim` (che assumiamo per semplicità essere un numero intero positivo). Per esempio, dati gli array `[9, 3, 5]` e `[1, 4, 7]`, il risultato ottenuto dalla somma è l'array `[10, 7, 12]`.

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

NOTA 2: Qualora ritenuta necessaria è ammessa la definizione di eventuali funzioni ausiliarie, a patto che siano a loro volta ricorsive o “wrapper” di altre funzioni a loro volta ricorsive.

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 5 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 codice_A21.cc

```
#include <iostream>

using namespace std;

void stampa_array(int array[], int dim);
void somma_array(int n1[], int n2[], int risultato[], int dim);

const int SIZE = 10;

int main() {
    int ar1[] = {5, 9, 2, 7, 10, 15, 3, 8, 4, 12};
    int ar2[] = {6, 3, 21, 1, 17, 11, 24, 9, 8, 19};
    int ar3[SIZE];

    cout << "Array1 = ";
    stampa_array(ar1, SIZE);
    cout << endl;
    cout << "Array2 = ";
    stampa_array(ar2, SIZE);
    cout << endl;

    somma_array(ar1, ar2, ar3, SIZE);

    cout << "Somma = ";
    stampa_array(ar3, SIZE);
    cout << endl;

    return 0;
}

void stampa_array(int array[], int dim) {
    for (int i = 0; i < dim; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione somma_array
```

2 soluzione_A21.cc

```
#include <iostream>

using namespace std;

void stampa_array(int array[], int dim);
void somma_array(int n1[], int n2[], int risultato[], int dim);
// Funzione ausiliaria
void somma_array_ric(int n1[], int n2[], int risultato[], int dim, int i);

const int SIZE = 10;

int main() {
```

```

int ar1[] = {5, 9, 2, 7, 10, 15, 3, 8, 4, 12};
int ar2[] = {6, 3, 21, 1, 17, 11, 24, 9, 8, 19};
int ar3[SIZE];

cout << "Array1 = ";
stampa_array(ar1, SIZE);
cout << endl;
cout << "Array2 = ";
stampa_array(ar2, SIZE);
cout << endl;

somma_array(ar1, ar2, ar3, SIZE);

cout << "Somma = ";
stampa_array(ar3, SIZE);
cout << endl;

return 0;
}

void stampa_array(int array[], int dim) {
    for (int i = 0; i < dim; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione somma_array

void somma_array(int n1[], int n2[], int risultato[], int dim) {
    somma_array_ric(n1, n2, risultato, dim, 0);
}

void somma_array_ric(int n1[], int n2[], int risultato[], int dim, int i) {
    if(i < dim) {
        risultato[i] = n1[i] + n2[i];
        somma_array_ric(n1, n2, risultato, dim, i + 1);
    }
}

```


2 Completare il programma contenuto nel file `esercizio2.cc` implementando la **funzione ricorsiva**

```
void prodotto_array(float a1[], float a2[], float risultato[], int lun)
```

che, dati in ingresso due array `a1` e `a2`, ne calcoli la somma e la memorizzi nell'array `risultato`: i tre array sono di uguale dimensione `lun` (che assumiamo per semplicità essere un numero intero positivo). Per esempio, dati gli array `[5.7, 3.0, 2.1]` e `[6.0, 2.0, 9.2]`, il risultato ottenuto dal prodotto è l'array `[34.2, 6.0, 19.32]`.

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

NOTA 2: Qualora ritenuta necessaria è ammessa la definizione di eventuali funzioni ausiliarie, a patto che siano a loro volta ricorsive o “wrapper” di altre funzioni a loro volta ricorsive.

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 5 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 codice_A22.cc

```
#include <iostream>

using namespace std;

void stampa_array(float array[], int dim);
void prodotto_array(float a1[], float a2[], float risultato[], int lun);

const int SIZE = 10;

int main() {
    float vet1[] = {5.7, 9, 2.1, 7, 5.6, 5.4, 3.1, 4.3, 4.2, 2.1};
    float vet2[] = {6, 3.5, 2.6, 1.2, 1.7, 11, 2.3, 9.3, 8.7, 9.2};
    float vet3[SIZE];

    cout << "Array1 = ";
    stampa_array(vet1, SIZE);
    cout << endl;
    cout << "Array2 = ";
    stampa_array(vet2, SIZE);
    cout << endl;

    prodotto_array(vet1, vet2, vet3, SIZE);

    cout << "Somma = ";
    stampa_array(vet3, SIZE);
    cout << endl;

    return 0;
}

void stampa_array(float array[], int dim) {
    for (int i = 0; i < dim; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione prodotto_array
```

2 soluzione_A22.cc

```
#include <iostream>

using namespace std;

void stampa_array(float array[], int lun);
void prodotto_array(float a1[], float a2[], float risultato[], int lun);
// Funzione ausiliaria
void prodotto_array_ric(float a1[], float a2[], float risultato[], int lun, int i);

const int SIZE = 10;

int main() {
```

```

float vet1[] = {5.7, 9, 2.1, 7, 5.6, 5.4, 3.1, 4.3, 4.2, 2.1};
float vet2[] = {6, 3.5, 2.6, 1.2, 1.7, 11, 2.3, 9.3, 8.7, 9.2};
float vet3[SIZE];

cout << "Array1 = ";
stampa_array(vet1, SIZE);
cout << endl;
cout << "Array2 = ";
stampa_array(vet2, SIZE);
cout << endl;

prodotto_array(vet1, vet2, vet3, SIZE);

cout << "Prodotto = ";
stampa_array(vet3, SIZE);
cout << endl;

return 0;
}

void stampa_array(float array[], int lun) {
    for (int i = 0; i < lun; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione prodotto_array

void prodotto_array(float a1[], float a2[], float risultato[], int lun) {
    prodotto_array_ric(a1, a2, risultato, lun, 0);
}

void prodotto_array_ric(float a1[], float a2[], float risultato[], int lun, int i) {
    if(i < lun) {
        risultato[i] = a1[i] * a2[i];
        prodotto_array_ric(a1, a2, risultato, lun, i + 1);
    }
}

```

2 Completare il programma contenuto nel file `esercizio2.cc` implementando la **funzione ricorsiva**

```
void moltiplica_array(long b1[], long b2[], long risultato[], int len)
```

che, dati in ingresso due array `b1` e `b2`, ne calcoli la somma e la memorizzi nell'array `risultato`: i tre array sono di uguale dimensione `len` (che assumiamo per semplicità essere un numero intero positivo). Per esempio, dati gli array `[1, 3, 2]` e `[5, 4, 3]`, il risultato ottenuto dal prodotto è l'array `[5, 12, 6]`.

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

NOTA 2: Qualora ritenuta necessaria è ammessa la definizione di eventuali funzioni ausiliarie, a patto che siano a loro volta ricorsive o “wrapper” di altre funzioni a loro volta ricorsive.

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 5 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 codice_A23.cc

```
#include <iostream>

using namespace std;

void stampa_array(long array[], int dim);
void moltiplica_array(long b1[], long b2[], long risultato[], int len);

const int SIZE = 10;

int main() {
    long ar1[] = {5, 9, 2, 7, 5, 4, 19, 8, 1, 11};
    long ar2[] = {6, 3, 2, 1, 31, 11, 7, 9, 51, 19};
    long ar3[SIZE];

    cout << "Array1 = ";
    stampa_array(ar1, SIZE);
    cout << endl;
    cout << "Array2 = ";
    stampa_array(ar2, SIZE);
    cout << endl;

    moltiplica_array(ar1, ar2, ar3, SIZE);

    cout << "Somma = ";
    stampa_array(ar3, SIZE);
    cout << endl;

    return 0;
}

void stampa_array(long array[], int dim) {
    for (int i = 0; i < dim; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione moltiplica_array
```

2 soluzione_A23.cc

```
#include <iostream>

using namespace std;

void stampa_array(long array[], int len);
void moltiplica_array(long b1[], long b2[], long risultato[], int len);
// Funzione ausiliaria
void moltiplica_array_ric(long b1[], long b2[], long risultato[], int len, int i);

const int SIZE = 10;

int main() {
```

```

long ar1[] = {5, 9, 2, 7, 5, 4, 19, 8, 1, 11};
long ar2[] = {6, 3, 2, 1, 31, 11, 7, 9, 51, 19};
long ar3[SIZE];

cout << "Array1 = ";
stampa_array(ar1, SIZE);
cout << endl;
cout << "Array2 = ";
stampa_array(ar2, SIZE);
cout << endl;

multiplica_array(ar1, ar2, ar3, SIZE);

cout << "Prodotto = ";
stampa_array(ar3, SIZE);
cout << endl;

return 0;
}

void stampa_array(long array[], int len) {
    for (int i = 0; i < len; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione multiplica_array

void multiplica_array(long b1[], long b2[], long risultato[], int len) {
    multiplica_array_ric(b1, b2, risultato, len, 0);
}

void multiplica_array_ric(long b1[], long b2[], long risultato[], int len, int i) {
    if(i < len) {
        risultato[i] = b1[i] * b2[i];
        multiplica_array_ric(b1, b2, risultato, len, i + 1);
    }
}

```

2 Completare il programma contenuto nel file `esercizio2.cc` implementando la **funzione ricorsiva**

```
void aggiungi_array(double x1[], double x2[], double risultato[], int num);
```

che, dati in ingresso due array `x1` e `x2`, ne calcoli la somma e la memorizzi nell'array `risultato`: i tre array sono di uguale dimensione `num` (che assumiamo per semplicità essere un numero intero positivo). Per esempio, dati gli array `[3, 2, 1]` e `[7, 9, 5]`, il risultato ottenuto dalla somma è l'array `[10, 11, 6]`.

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

NOTA 2: Qualora ritenuta necessaria è ammessa la definizione di eventuali funzioni ausiliarie, a patto che siano a loro volta ricorsive o “wrapper” di altre funzioni a loro volta ricorsive.

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 5 punti (al punteggio di tutti gli esercizi va poi sommato 10).

2 codice_A24.cc

```
#include <iostream>

using namespace std;

void stampa_array(double array[], int num);
void aggiungi_array(double x1[], double x2[], double risultato[], int num);

const int SIZE = 10;

int main() {
    double ar1[] = {7.3, 2, 5.2, 7.8, 10, 15, 9.0, 8.1, 5, 13};
    double ar2[] = {6.2, 3.1, 14.7, 1, 19.4, 11.1, 27, 2.9, 8.1, 20.5};
    double ar3[SIZE];

    cout << "Array1 = ";
    stampa_array(ar1, SIZE);
    cout << endl;
    cout << "Array2 = ";
    stampa_array(ar2, SIZE);
    cout << endl;

    aggiungi_array(ar1, ar2, ar3, SIZE);

    cout << "Somma = ";
    stampa_array(ar3, SIZE);
    cout << endl;

    return 0;
}

void stampa_array(double array[], int num) {
    for (int i = 0; i < num; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione aggiungi_array
```

2 soluzione_A24.cc

```
#include <iostream>

using namespace std;

void stampa_array(double array[], int num);
void aggiungi_array(double x1[], double x2[], double risultato[], int num);
// Funzione ausiliaria
void aggiungi_array_ric(double x1[], double x2[], double risultato[], int num, int i);

const int SIZE = 10;

int main() {
```



```

double ar1[] = {7.3, 2, 5.2, 7.8, 10, 15, 9.0, 8.1, 5, 13};
double ar2[] = {6.2, 3.1, 14.7, 1, 19.4, 11.1, 27, 2.9, 8.1, 20.5};
double ar3[SIZE];

cout << "Array1 = ";
stampa_array(ar1, SIZE);
cout << endl;
cout << "Array2 = ";
stampa_array(ar2, SIZE);
cout << endl;

aggiungi_array(ar1, ar2, ar3, SIZE);

cout << "Somma = ";
stampa_array(ar3, SIZE);
cout << endl;

return 0;
}

void stampa_array(double array[], int num) {
    for (int i = 0; i < num; i++) {
        cout << array[i] << " ";
    }
}

// Inserire qui la definizione della funzione aggiungi_array

void aggiungi_array(double x1[], double x2[], double risultato[], int num) {
    aggiungi_array_ric(x1, x2, risultato, num, 0);
}

void aggiungi_array_ric(double x1[], double x2[], double risultato[], int num, int i) {
    if(i < num) {
        risultato[i] = x1[i] + x2[i];
        aggiungi_array_ric(x1, x2, risultato, num, i + 1);
    }
}

```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di `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 per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento 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, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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 res;
    double val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
            << " Push (u)\n"
            << " Pop (o)\n"
            << " Top (t)\n"
            << " Print (p)\n"
            << " Quit (q)\n";
        cin >> res;
        switch (res) {
            case 'u':
                cout << "Val? : ";
                cin >> val;
                if (!push(s, val)) {
                    cout << "Pila piena!\n";
                }
                break;
            case 'o':
                if (!pop(s)) {
                    cout << "Pila vuota!\n";
                }
                break;
            case 't':
                if (!top(s, val)) {
                    cout << "Pila vuota!\n";
                } else {
                    cout << "Top = " << val << endl;
                }
                break;
            case 'p':
                print(s);
                break;
            case 'q':
                break;
            default:
                cout << "Opzione errata\n";
        }
    } while (res != 'q');
```

```

    }
} while (res != 'q');

deinit(s);

return 0;
}

```

3 stack.h

```

#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    double *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, double n);
bool top(const stack &s, double &out);
bool pop(stack &s);
void print(const stack &s);

#endif

```

3 stack.cc

```

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new double[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
}

```

```

}

bool top(const stack &s, double &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, double n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}

```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di `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 per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento 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, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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 res;
    char val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
            case 'u':
                cout << "Val? : ";
                cin >> val;
                if (!push(s, val)) {
                    cout << "Pila piena!\n";
                }
                break;
            case 'o':
                if (!pop(s)) {
                    cout << "Pila vuota!\n";
                }
                break;
            case 't':
                if (!top(s, val)) {
                    cout << "Pila vuota!\n";
                } else {
                    cout << "Top = " << val << endl;
                }
                break;
            case 'p':
                print(s);
                break;
            case 'q':
                break;
            default:
                cout << "Opzione errata\n";
        }
    } while (res != 'q');
```

```

    }
} while (res != 'q');

deinit(s);

return 0;
}

```

3 stack.h

```

#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    char *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, char n);
bool top(const stack &s, char &out);
bool pop(stack &s);
void print(const stack &s);

#endif

```

3 stack.cc

```

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new char[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
}

```



```

}

bool top(const stack &s, char &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, char n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}

```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di `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 per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento 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, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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 res;
    int val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
            case 'u':
                cout << "Val? : ";
                cin >> val;
                if (!push(s, val)) {
                    cout << "Pila piena!\n";
                }
                break;
            case 'o':
                if (!pop(s)) {
                    cout << "Pila vuota!\n";
                }
                break;
            case 't':
                if (!top(s, val)) {
                    cout << "Pila vuota!\n";
                } else {
                    cout << "Top = " << val << endl;
                }
                break;
            case 'p':
                print(s);
                break;
            case 'q':
                break;
            default:
                cout << "Opzione errata\n";
        }
    } while (res != 'q');
```

```

    }
} while (res != 'q');

deinit(s);

return 0;
}

```

3 stack.h

```

#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    int *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, int n);
bool top(const stack &s, int &out);
bool pop(stack &s);
void print(const stack &s);

#endif

```

3 stack.cc

```

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

static bool is_empty(const stack &s)
{
    return s.index == 0;
}

static bool is_full(const stack &s)
{
    return s.index == s.dim;
}

void init(stack &s, int maxdim)
{
    s.index = 0;
    s.dim = maxdim;
    s.elem = new int[maxdim];
}

void deinit(stack &s)
{
    delete[] s.elem;
}

```

```

}

bool top(const stack &s, int &out)
{
    if (is_empty(s)) {
        return false;
    } else {
        out = s.elem[s.index-1];
        return true;
    }
}

bool push(stack &s, int n)
{
    if (is_full(s)) {
        return false;
    } else {
        s.elem[s.index++] = n;
        return true;
    }
}

bool pop(stack &s)
{
    if (is_empty(s)) {
        return false;
    } else {
        s.index--;
        return true;
    }
}

void print(const stack &s)
{
    for (int i = 0; i < s.index; i++) {
        cout << s.elem[i] << " ";
    }
    cout << endl;
}

```

3 Nel file `stack_main.cc` è definita la funzione `main` che contiene un menu per gestire una pila di `float`. 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 per contenere al più `dim` elementi;
- `deinit` liberi la memoria utilizzata dalla pila;
- `push` inserisca l'elemento passato come parametro nella pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `pop` tolga l'elemento in testa alla pila, restituendo `true` se l'operazione è andata a buon fine, e `false` altrimenti;
- `top` legga l'elemento 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, dall'elemento più vecchio al più recente.

La pila deve essere implementata con un array allocato dinamicamente, la cui dimensione è specificata dall'argomento `dim` della funzione `init`.

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 res;
    float val;
    stack s;

    int maxdim = -1;
    while (maxdim <= 0) {
        cout << "Inserire la dimensione massima della pila: ";
        cin >> maxdim;
    }

    init(s, maxdim);
    do {
        cout << "\nOperazioni possibili:\n"
             << " Push (u)\n"
             << " Pop (o)\n"
             << " Top (t)\n"
             << " Print (p)\n"
             << " Quit (q)\n";
        cin >> res;
        switch (res) {
            case 'u':
                cout << "Val? : ";
                cin >> val;
                if (!push(s, val)) {
                    cout << "Pila piena!\n";
                }
                break;
            case 'o':
                if (!pop(s)) {
                    cout << "Pila vuota!\n";
                }
                break;
            case 't':
                if (!top(s, val)) {
                    cout << "Pila vuota!\n";
                } else {
                    cout << "Top = " << val << endl;
                }
                break;
            case 'p':
                print(s);
                break;
            case 'q':
                break;
            default:
                cout << "Opzione errata\n";
        }
    } while (res != 'q');
```

```

    }
} while (res != 'q');

deinit(s);

return 0;
}

```

3 stack.h

```

#ifndef STRUCT_STACK_H
#define STRUCT_STACK_H

struct stack {
    int index;
    int dim;
    float *elem;
};

void init(stack &s, int maxdim);
void deinit(stack &s);
bool push(stack &s, float n);
bool top(const stack &s, float &out);
bool pop(stack &s);
void print(const stack &s);

#endif

```

3 stack.cc

- 4 Scrivere nel file **esercizio4.cc** la definizione della funzione **f**. Tale funzione dovrà essere **ricorsiva in coda** e dovrà implementare la funzione di Fibonacci.

NOTA 1:

non sono ammessi cicli.

NOTA 2:

le uniche funzioni ricorsive ammesse devono essere funzioni ricorsive in coda.

NOTA 3:

è ammesso l'uso di funzioni wrapper.

ESEMPIO:

```
> ./a.out
n? (n >= 0): 10
f( 0)= 1
f( 1)= 1
f( 2)= 2
f( 3)= 3
f( 4)= 5
f( 5)= 8
f( 6)= 13
f( 7)= 21
f( 8)= 34
f( 9)= 55
f(10)= 89
```

VALUTAZIONE:

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

4 codice_A41.cc

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

long long f (int n);

int main()
{
    int n;
    cout << "n? (n >= 0): ";
    cin >> n;
    for (int i = 0; i <= n; i++)
        cout << "f(" << setw(2) << i << ")="
            << setw(10) << f(i) << endl;
    return 0;
}

// Scrivere qui la definizione della funzione ricorsiva in coda f:
```

4 soluzione_A41.cc

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

long long f (int n);

int main()
{
    int n;
    cout << "n? (n >= 0): ";
    cin >> n;
    for (int i = 0; i <= n; i++)
        cout << "f(" << setw(2) << i << ")="
            << setw(10) << f(i) << endl;
    return 0;
}

// SOLUZIONE:

long long f1 (long long fimenodue, long long fimenouno, int i, int n)
{
    long long fn;
    if (i==n)
        fn = fimenodue+fimenouno;
    else
        fn = f1(fimenouno,fimenouno+fimenodue,i+1,n);
    return fn;
}

long long f (int n) {
```

```
long long fn;  
if (n==0||n==1)  
    fn = 1;  
else  
    fn = f1 (1,1,2,n);  
return fn;  
}
```