

Problem 1 (15 points)

Write a program to reverse the characters of a standard string entered by the user and print the result to the screen, i.e. hello → olleh.

```
#include <iostream>
using namespace std;
int main() {
    string s;
    cout << "Enter string" << endl;
    cin >> s;
    const int n = s.length();
    char str[n + 1];
    int j = 0;
    for (int i = n - 1; i >= 0; i--) {
        str[j++] = s[i];
    }
    str[j] = 0; \\ Add the null character at the end

    cout << str << endl;
}
```

OR:

```
#include <iostream>
using namespace std;
int main() {
    string s;
    cout << "Enter string" << endl;
    cin >> s;
    const int n = s.length();
    char str[] = s.c_str();
    int start = 0, end = n - 1;
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
    cout << str << endl;
}
```

Problem 2 (15 points)

Write a function *max* to find the maximum of an array of n numbers. Your code should include both the *main* and the *max* functions. The *main* function should read the array size and the array elements from the user, pass it to the function, receive the maximum element from the function, and print it to the screen.

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

int max(int x[], int n)
{
    int max = x[0];
    for(int i = 1; i < n; i++)
    {
        if(x[i] > max) max = x[i];
    }
    return max;
}

int main()
{
    int n;
    cout << "Enter number of numbers" << endl;
    cin >> n;
    int * x = new int[n];
    for(int i = 0; i < n; i++)
    {
        cin >> x[i];
    }
    cout << "Max = " << max(x, n) << endl;
}
```

Problem 3 (15 points)

- a. (10 points) Write a function *factorial* to find the factorial of a number $n \geq 0$ **iteratively** (using loops). Your code should include the *main* and the *factorial* functions. The *main* function should read the value of n from the user, pass it to the factorial function, receive the calculated factorial from the function, and print the result to the screen. An error message should be displayed if the number is less than 0, and the program should end. *Remember:* $0! = 1$ and $n! = 1 * 2 * 3 * \dots * n$ for $n \geq 1$

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

long factorial(int n)
{
    long f = 1;
    for(int i = 2; i <= n; i++)
    {
        f *= i;
    }
    return f;
}

int main()
{
    int n;
    cout << "Enter number:" << endl;
    cin >> n;
    if(n < 0)
    {
        cout << "Error: Negative number" << endl;
        return -1; // or any number other than 0
    }
    else
    {
        cout << "Factorial: " << factorial(n) << endl;
    }
}
```

- b. (5 points) Repeat part (a) using a **recursive** function. *Remember:* $0! = 1$ and $n! = n * (n - 1)!$ for $n \geq 1$

```
long factorial(int n)
{
    if(n <= 1) return 1;
    return n * factorial (n-1);
}
```

Problem 4 (15 points)

Write a function *getGrade* that takes the score of a student and returns a letter representing the grade as follows:

- $85 \leq A \leq 100$
- $75 \leq B < 85$
- $65 \leq C < 75$
- $50 \leq D < 65$
- $0 \leq F < 50$

The *main* function should read scores from a file named *scores.csv*, pass each score to the function, get the grade back from the function, and print it to the screen.

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

char getGrade(int score)
{
    if(score >= 85 && score <= 100) return 'A';
    else if(score >= 75) return 'B';
    else if(score >= 65) return 'C';
    else if(score >= 50) return 'D';
    else return 'F';
}

int main()
{
    ifstream file;
    file.open("scores.csv");
    int score;
    char grade;
    while(!file.eof())
    {
        file >> score;
        grade = getGrade(score);
        cout << "Grade: " << grade << endl;
    }
}
```

Problem 5 (30 points)

Design and implement a class *complex* to represent a complex number. Write the declaration and the definition of the class. Also, write a main function to test your class.

The class should include the following member functions:

- A default constructor that takes no arguments and create a complex number with value 0
- A constructor that takes the real and imaginary parts of the complex number.
 - The imaginary part should be optional. If not given, it is assumed to be zero
- A copy constructor that copy the values of another complex number
- A destructor
- Getters and setters for both real and imaginary parts of the complex number
- A function to print the complex number in the form $x + i y$ where x and y are the real and imaginary parts respectively
- A function to calculate the absolute value of the complex number
- For two complex numbers a and b , and a double (real) number x , your class should be able to perform the following operations:

```
-a; // negate the complex number

a + b; // Add two complex numbers
a + x; // Add a real number and a complex number
x + a; // Add a complex number and a real number

a += b; // Add a complex number to the other
a += x; // Add a real number to a complex number

cin >> a; // Read real and imaginary parts
cout << a; // Print the complex number in the form x+iy
```

See solution on following pages

complex.h

```
#ifndef COMPLEX_H
#define COMPLEX_H
#include <iostream>
using namespace std;

class complex {
private:
    double re, im;
public:
    complex();
    complex(double r, double i = 0);
    complex(const complex& c);
    ~complex();

    double real() const;
    double imaginary() const;
    complex& real(double r);
    complex& imaginary(double i);

    void print() const;

    double abs() const;

    complex& operator-(); // -a

    complex operator+(const complex& rhs); // a+b
    complex operator+(double rhs); // a+x
    friend complex operator+(const complex& lhs, double rhs);

    complex& operator+=(const complex& rhs); // a+=b
    complex& operator+=(double rhs); // a+=x

    friend ostream& operator<<(ostream& out, const complex &c);
    friend istream& operator>>(istream& in, complex &c);

};

#endif
```

complex.cpp

```
#include "complex.h"
#include <cmath>

complex::complex(): re(0), im(0) {}
complex::complex(double r, double i): re(r), im(i) {}
complex::complex(const complex& c): re(c.re), im(c.im) {}
complex::~~complex() {}

double complex::real() const {
    return re;
}

double complex::imaginary() const {
    return im;
}

complex& complex::real(double r) {
    re = r;
    return *this;
}

complex& complex::imaginary(double i) {
    im = i;
    return *this;
}

void complex::print() const {
    cout<< re << "+" << im;
}

double complex::abs() const {
    return sqrt(re*re + im * im);
}

complex& complex::operator-() {
    re = -re;
    im = -im;
    return *this;
}

complex complex::operator+(const complex& rhs) {
    return complex(re + rhs.re, im + rhs.im);
}

complex complex::operator+(double rhs) {
    return complex(re + rhs, im);
}

complex operator+(const complex& lhs, double rhs) {
    return complex (lhs.re + rhs, lhs.im);
}
```

```

complex& complex::operator+=(const complex& rhs) {
    re += rhs.re;
    im += rhs.im;
    return *this;
}

complex& complex::operator+=(const double& rhs) {
    re += rhs;
    return *this;
}

ostream& operator<<(ostream& out, const complex &c) {
    out << c.re << "+ i" << c.im;
}

istream& operator>>(istream& in, complex &c) {
    in >> c.re >> c.im;
}

```

main.cpp

```

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

int main() {
    complex c0; // complex()
    complex c1(3, 4); // complex(double, double)
    complex c2(c1); // complex(complex&)
    double d = 5;

    cin >> c3; // operator>>()
    cout << c1 << endl; // operator<<()
    cout << c2.abs() << endl; // abs()
    cout << c1.real() << endl; // real()
    cout << c1.imaginary() << endl; // imaginary()
    c1.print

    cout << -c2 << endl; // operator-()
    cout << c1 + c2 << endl; // operator+(complex)
    cout << c1 + d << endl; // operator+(double)
    cout << d + c1 << endl; // operator+(double, complex)

    c1 += 4; // operator +=(double)
    cout << c1 << endl;

    c1 += c2; // operator +=(complex)
    cout << c4 << endl;
}

```