INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

Write a C Program

- Solve the equation $a\,x^2 + b\,x + c = 0$, where a, b, and c are of type double, and return the solutions as values of type Complex.

```
typedef struct complex { double r;  double i; } Complex;
void solveEquation(void) {
   double a, b, c, d;
   Complex s1, s2;
   // read a, b, c;
   d = b * b – 4 * a * c;
   if (d > 0.0) {
      s1.r = (- b + sqrt(d) ) / (2.0 * a); s1.i = 0.0;
      s2.r = (- b - sqrt(d) ) / (2.0 * a);  s2.i = 0.0;
   } else if (d == 0.0) {
      s1.r = (- b) / (2 .0* a); s1.i = 0.0;
      s2.r = (- b) / (2.0 * a); s2.i = 0.0;
   } else {
      s1.r = (- b) / (2.0 * a); s1.i = (sqrt(- d) ) / (2.0 * a);
      s2.r = (- b) / (2.0 * a); s2.i = (- sqrt(- d) ) / (2.0 * a);
   }
   // write s1, s2;
}
```

What Are the Features Supported by C in Your Program?

- Declaration of the user-defined type Complex.
- Declaration of variables of types double and Complex.
- Input and output of variables of type double.
- Operations of double values.
- Control statement if.

What Are the Differences between double and Complex?

- Double is a built-in data type and Complex is a user-defined data type.
- Data representation of double value is unknown.
- Operations of double values are well-defined.
- Values of double can only be accessed via well-defined operations.

Can Complex Be Implemented Almost Like a Built-in Type in C?

- What should be in the header file?
- What should be in the code file?
- How to hide the data representation?

What Is a Data Type?

- A set of values.
- A set of values and a set of operations.
- A set of operations.

A Set of Values

```
/* complex.h */
typedef struct complex { double r;  double i; } *Complex;
```

A Set of Values and Operations

```
/* complex.h */
typedef struct complex { double r;  double i; } *Complex;

extern Complex  newComplex(double, double) ;

extern Complex  add(Complex , Complex ) ;
extern Complex  sub(Complex, Complex) ;
extern Complex  mul(Complex, Complex) ;
extern Complex  div(Complex, Complex) ;

extern Boolean eql(Complex, Complex) ;
extern Boolean neq(Complex, Complex) ;
extern void assign(Complex , Complex) ;
```

A Set of Operations

```
/* complex.h */
typedef struct complex *Complex;

extern Complex  newComplex(double, double) ;

extern Complex  add(Complex , Complex ) ;
extern Complex  sub(Complex, Complex) ;
extern Complex  mul(Complex, Complex) ;
extern Complex  div(Complex, Complex) ;

extern Boolean eql(Complex, Complex) ;
extern Boolean neq(Complex, Complex) ;
extern void assign(Complex , Complex) ;
```

A Set of Operations

```
/* complex.c */
#include "complex.h"

struct complex { double r;  double i; };

Complex  newComplex(double r, double i)  { … }
Complex  add(Complex c1, Complex c2)  { … }
Complex  sub(Complex c1, Complex c2)  { … }
Complex  mul(Complex c1, Complex c2)  { … }
Complex  div(Complex c1, Complex c2)  { … }
Boolean eql(Complex c1, Complex c2)  { … }
Boolean neq(Complex c1, Complex c2)  { … }
void assign(Complex var, Complex exp)  { … }
```

Write a C Program

- Simulate a waiting queue of patients in a hospital.
- There is a doctor and a registrar.
- Patients are generated by the registrar at a random rate.
- Patients are served by the doctor at a random rate.

```
typedef struct doctor { char * name;  int doctorNo; } Doctor;
typedef struct registrar { char * name;  int registrarNo; } Registrar;
typedef struct patient { char * name;  int patientNo; } Patient;
typedef struct queue { Patient p;  struct queue *link; } *Queue;

void simulateHospital(void) {
   Doctor doc;
   Registrar reg;
   Queue que;

   while (1) {
      registrarGenPatient(req, que);
      doctorServePatient(doc, que);
   }
}
```

What Data Types Are There in Your Program?

- Doctor
- Registrar
- Patients
- Queue of Patients

Implement a Queue of Doctors, Registrars, or Patients

- What are similar properties for doctors, registrars, and patients?
- They are all humans.
- They have similar attributes (name, number, …).
- They have similar behaviors (eat, walk, wait, …).
- A queue of humans?

Implement a Queue of Humans in C

- Implement a union type Human that can contain values of types Doctor, Registrar, or Patient.

```
typedef struct doctor { char * name;  int doctorNo; } Doctor;
typedef struct registrar { char * name;  int registrarNo; } Registrar;
typedef struct patient { char * name;  int patientNo; } Patient;

typedef union human { Doctor d; Registrar r; Patient p; } Human;
typedef struct queue { Human h; struct queue *link; } *Queue;

void simulateHospital(void) {
   Doctor doc;
   Registrar reg;
   Queue que;

   while (1) {
      registrarGenPatient(req, que);
      doctorServePatient(doc, que);
   }
}
```

A Queue Definition for Both Type Patient and Type Human?

- The data representations for Queue of Patient and Queue of Human are very similar.
- The operations for Queue of Patient and Queue of Human are very similar.
- Is it possible to give a definition for both?

Implement a Queue of Any Type in C

- The data member is of type void *.
- Casting of pointer types is used to handle different types.

```
typedef struct queue { void *data;  struct queue *link; } *Queue;

void insert(Queue *, void *) ;
void * remove(Queue *);

Queue queue = 0;
int *data;

data = new(int);
*data = 1;
insert(&queue, (void *) data);
data = 0;
data = (int *) remove(&queue);
```

What Is Object-Oriented Programming?

- A program consists of several types of objects that interact with each other.
- Each object in the program has some attributes (data) and some behaviors (operations).
- An object interacts with another object by activating one of its behaviors.
- The activation of a behavior of an object may change attributes of that object.

Abstraction in Object-Oriented Programming

- Encapsulation (Types)
- Inheritance (Type Hierarchy)
- Polymorphism (Generic Types)

Reliable Object-Oriented Programming

- Type specification
  - Preconditions and postconditions for each operation of a type
  - Invariants of types
- Unit testing
- Exception handling

Content

- Encapsulation
- Reliability
- Inheritance
- Polymorphism