Laboratory of
Data Structures and Algorithms

DATA STRUCTURES IN C

I. FUNDAMENTALS

1. **Introduction**

C programming language can process single or grouped variables, which enable global processing. An example of the second category is the **matrix**, which is in fact an ordered set of data of the same type (the order of the elements is realized by indices).

However, often it is useful to group the data other than the one used for matrices. This time the data are not necessarily of the same type and requires a global processing. This form of group is called **structure**.

Reference to elements of such groups doesn't use indices but a special way that include the name of structure. Components of the groups can be groups themselves. Furthermore, it is possible to define a hierarchy of groups.

A very simple example of data structure is the calendar. A calendar date consists of three elementary date: day, month and year. The day and the year are integers, while the month could be a string of characters.

2. **Declaration of structure**

The general syntax for a struct declaration in C is:

```
struct tag_name
        { type member1;
          type member2;
        …
        } identification_1, identification_2, …, identification_n;
```

Here tag_name or identification_i are optional in some contexts.

Notes:

- if identification _1, identification _2, …, identification_n are absent, then tag_name should be present;
- if tag_name is absent, then at least identification 1 should be present.

A variable of the structure type can be declared subsequently:

struct **tag_name identification _1, …, identification_n;**

Examples:
a. There will be declared the date of birth and date of employment as calendaristic_data's type structures (composed of day, month, and year):

```
struct calendar_data
     {int day;
     char month[11];
     int year;
     } birth_date, employment_date;
```

b. You can omit entering the *calendar_data*:

```
struct
     {int day;
     char month[11];
     int year;
     } birth_date, employment_date;
```

c. In a program we can firstly define the calendar_data as a name of a general structure and then, subsequently, we declare both the birth_date and the employment_date.:

```
struct calendar_data
     {int day;
     char month[11];
     int year;
     };
     ...
struct calendar_data  birth_date, employment_date;
```

The previous three code examples will have the same result.

Extrapolating the above ideas, we can easily define a general structure of personal data of employees of an institution that includes: name, address, place of birth, date of birth, date of employment, education, gender, etc.

```
  struct  personal_data
        { char name[100];
          char address[1000];
          struct calendar_data   birth_date, employment_date;
          char gender;
        };
    ........
  struct personal_data   manager, employees[1000];
```

The variable named *manager* is a structure of *personal_data* type, and *employees[1000]* is an array of structures.

3. **The access to the elements of a structure**
The access to the elements of a structure can be done in one of the following two ways:

struct_name.date_name

or

pointer -> date_name

where: - **struct_name** is the name of structure,

- **date_name** is the name of a specific component of the structure,

- **pointer** is a pointer to that structure.

4. **Typedef declarations**
By declaring a structure, we introduce a new type.
In general, a name can be assigned to a type, whether it is a predefined type or one defined by the programmer. This should be done by using the following syntax:

typedef tip nume_tip;

where
- **tip** is a predefined type or one previously defined by the programmer;
- **nume_tip** is the name allocated to the new type.

Example:
By using the statement

typedef double REAL;

the data

REAL x, y;

are of the double type.

II. ASSIGNMENT WORKFLOW

1. Write a program that reads the complex numbers from the keyboard and display their modulus. There will be a global definition (with typedef) for complex numbers (introduced as a structure).
For a complex number

z = x + i*y

modulus is the square root of x * x + y * y. The function for extracting the square root is sqrt and is defined in the header file named math.h. Therefore the modulus of a complex number will be calculated through a function.
The program will read the complex numbers and will display theirs modulus, till a non-numeric value will be finally introduced from the keyboard (then the program execution will be stopped).

O possible solution:

```
#include<stdio.h>
#include<math.h>

typedef struct {
                double x;
                double y;
               } COMPLEX;
double dmodul(COMPLEX *z);


int main()  /*  the program reads complex numbers and displays their module */
{
 COMPLEX complex;
 printf("\n Enter the real part and then, after a blank, the imaginary part ");
 printf("\n of the complex number  z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
  {
   printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
   printf("modulus = %g \n",dmodul(&complex));
   printf("\n\n Enter the real part and then, after a blank, the imaginary part ");
   printf("\n of the complex number  z = a + ib :");
   printf("\n(a non-numeric value will finish the program)\n");
  }
}

double dmodul(COMPLEX *z)
/*  calculates and displays the modulus of the complex number z */
{
  return sqrt(z->x * z->x + z->y * z->y);
}
```

2. Try to introduce in the above program a function for calculating the argument of a complex number.

If

    z = x + i*y

then

    arg z

is computed in the following way:

a. If x = y = 0, then arg z = 0.

b. If y = 0 and x != 0
   then: if x > 0, then arg z = 0;
        otherwise (which means x<0) then arg z = pi = 3.1415926535.

c. If x = 0 and y != 0
then: if y >0, then arg z = pi/2;
    otherwise (which means y<0) then arg z = 3*pi/2.

d. If x != 0 and y != 0 then
consider

    a = arctg(y/x)

If: x > 0 and y > 0, then arg z = a;
    x > 0 and y < 0, then arg z = 2*pi + a;
    x < 0, then arg z = pi + a.

By using #define we will introduce the constant value of PI in the program. The function arctan can be finding by using the name atan (we have to include math.h).

3. Split the above program in three files (with .cpp extension) using known principles and then start compiling the file containing the main function.

4. Write (in a new file) a function:

```
void sum_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

which assign the sum of complex numbers a and b to the resulting number c.

5. Write (in a new file) a function:

```
void dif_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

which assign the difference of complex numbers a and b to the resulting number c.

6. Write (in a new file) a function:

```
void mul_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

which assign the multiplication of complex numbers a and b to the resulting number c.

7. Write (in a new file) a function:

```
void div_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
```

which assign the division of complex numbers a and b to the resulting number c. What supplementary steps should you take?

8. Write a program that requires the introduction of two complex numbers and displays the results of their summing, difference, multiplication and division. The program will call all previous files written in paragraphs 4-7.

III. SOLUTIONS

2. L11_2.C

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159265358979

typedef struct {
                double x;
                double y;
               } COMPLEX;

double dmodul(COMPLEX *z);
double darg(COMPLEX *z);



int main()  /* the program reads complex numbers and displays their module and argument */
{
 COMPLEX complex;
 printf("\n Enter the real part and then, after a blank, the imaginary part ");
 printf("\n of the complex number  z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
  {
   printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
   printf("modulus=%g \t argument=%g \n",dmodul(&complex),darg(&complex));
   printf("\n\n Enter the real part and then, after a blank, the imaginary part ");
   printf("\n of the complex number  z = a + ib :");
   printf("\n(a non-numeric value will finish the program)\n");
  }
}

double dmodul(COMPLEX *z)
/* calculates and displays the modulus of the complex number z */
{
  return sqrt(z->x * z->x + z->y * z->y);
}

double darg(COMPLEX *z)
{
  double a;

  if(z->x==0 && z->y==0)
     return 0.0 ;
  if(z->y==0)
     if(z->x > 0)
        return 0.0;
```

```
    else /* y=0 and x<0 */
       return PI;
  if(z->x==0)
    if(z->y > 0)
       return PI/2;
    else /* x=0 and y<0 */
       return (3*PI)/2;

   /* x != 0 and y != 0 */
   a = atan(z->y/z->x);
   if(z->x < 0)
     return a+PI;
   else
     if(z->y < 0)      /* x>0 and y<0 */
      return 2*PI+a;
     else              /* x>0 and y>0 */
      return a;
}
```

3.1. L11_3_1.CPP

```
double dmodul(COMPLEX *z)
/* calculates and displays the modulus of the complex number z */
{
  return sqrt(z->x * z->x + z->y * z->y);
}
```

3.2. L11_3_2.CPP

```
double darg(COMPLEX *z)
{
  double a;

  if(z->x==0 && z->y==0)
    return 0.0 ;
  if(z->y==0)
    if(z->x > 0)
       return 0.0;
    else /* y=0 and x<0 */
       return PI;
  if(z->x==0)
    if(z->y > 0)
       return PI/2;
    else /* x=0 and y<0 */
       return (3*PI)/2;

   /* x != 0 and y != 0 */
   a = atan(z->y/z->x);
   if(z->x < 0)
     return a+PI;
   else
     if(z->y < 0)      /* x>0 and y<0 */
       return 2*PI+a;
     else              /* x>0 and y>0 */
       return a;
}
```

3.3. L11_3_3.CPP

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159265358979

typedef struct {
                double x;
                double y;
               } COMPLEX;

#include "l11_3_1.cpp"
#include "l11_3_2.cpp"

int main()  /* the program reads complex numbers and displays their module and argument */
{
 COMPLEX complex;
 printf("\n Enter the real part and then, after a blank, the imaginary part ");
 printf("\n of the complex number  z = a + ib :\n");
 while(scanf("%lf %lf",&complex.x,&complex.y)==2)
   {
   printf("a+ib= %g + i*(%g)\n",complex.x,complex.y);
   printf("modulus=%g \t argument=%g \n",dmodul(&complex),darg(&complex));
   printf("\n\n Enter the real part and then, after a blank, the imaginary part ");
   printf("\n of the complex number  z = a + ib :");
   printf("\n(a non-numeric value will finish the program)\n");
   }
}
```

4. L11_4.CPP

```
void sum_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x + b->x;
  c->y = a->y + b->y;
}
```

5. L11_5.CPP

```
void dif_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x - b->x;
  c->y = a->y - b->y;
}
```

6. L11_6.CPP

```
void mul_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  c->x = a->x * b->x + a->y * b->y;
  c->y = a->x * b->y + b->x * a->y;
}
```

7. L11_7.CPP

```
void div_c(COMPLEX *a, COMPLEX *b, COMPLEX *c)
{
  double numitor;
  numitor = b->x * b->x + b->y * b->y;
  if(numitor==0)
        exit(1);
  c->x = (a->x * b->x + a->y * b->y) / numitor;
  c->y = (a->y * b->x - a->x * b->y) / numitor;
}
```

8. L11_8.CPP

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>

typedef struct {
                double x;
                double y;
               } COMPLEX;

#include "l11_4.cpp"
#include "l11_5.cpp"
#include "l11_6.cpp"
#include "l11_7.cpp"

int main()

{
 COMPLEX a,b,c;
 printf("\n Enter the real part and then, after a blank, the imaginary part ");
 printf("\n of the complex number a :\n");
 if(scanf("%lf %lf",&a.x,&a.y)!=2)
        {
         printf("\n Error");
```

```
          exit(1);
         }
 printf("a = %g + i*(%g)\n",a.x,a.y);

 printf("\n Enter the real part and then, after a blank, the imaginary part ");
 printf("\n of the complex number b :\n");
 if(scanf("%lf %lf",&b.x,&b.y)!=2)
         {
          printf("\n Error");
          exit(1);
         }
 printf("b = %g + i*(%g)\n",b.x,b.y);

 sum_c(&a,&b,&c);
 printf("\na+b = %g + i*(%g)",c.x, c.y);
 dif_c(&a,&b,&c);
 printf("\na-b = %g + i*(%g)",c.x, c.y);
 mul_c(&a,&b,&c);
 printf("\na*b = %g + i*(%g)",c.x, c.y);
 div_c(&a,&b,&c);
 printf("\na/b = %g + i*(%g)\n",c.x, c.y);

 getch();
}
```