|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | **Programming Fundamentals** | |  | **(CL214)** | |  | **LABORATORY MANUAL** | |  | **Spring 2021** | |  | **C:\Users\Aamer\Desktop\nu-new.png**  **LAB 04** | |  | **Structure** | |  | **Engr. Ibrar Khan**  **Engr. Sana Saleh**  **Muhammad Adeel Akhtar 20I-1025 B** |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_ | | | \_\_\_ | | STUDENT NAME | | ROLL NO | | | SEC | |  | | | | | | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | | | LAB ENGINEER SIGNATURE & DATE | | | | | | | **MARKS AWARDED: /10** | | | | | | |  | | | | | | | **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), ISLAMABAD** | | | | | | |  | | | | | | | Prepared by: | Engr. Sana Saleh | | Date: | 15 Feb, 2019 | | | Verified by: | Engr. Shahid Qureshi | | Date: | 16 Feb, 2019 | | |

|  |  |
| --- | --- |
| **LAB 04** | **Structure** |

**Lab Objectives:**

1. To learn what are structures in C++
2. To learn how to define a structure and access its data members.
3. To learn about array of structures.

**Software Required:**

* Dev C++

**Introduction:**

1. **Structures in C++**

Structure is the collection of variables of different types under a single name for better visualization of problem. Arrays is also collection of data but arrays can hold data of only one type whereas structure can hold data of one or more types.

## How to declare a structure in C++ Programming?

The***struct***keyword declares a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure, termed as definition of structure. For example:

struct person {

char name[100];

int age;

float salary;

};

Here a structure **person** is declared which has three members: **name**, **age** and **salary**.

When a structure is declared, no memory is allocated. The structure definition is only the blueprint for the creation of variables. You can imagine it as a datatype. When you define an integer as below:

int foo;

The *int* specifies that, variable **foo** can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.

## How to declare a structure variable?

Once you have declared a structure person as above, you can declare a structure variable (or identifier) as:

person bill;

Here, a structure variable bill is defined which is of type structure **person**. When structure variable is declared only then, the required memory is allocated by the compiler. The memory of **float** is 4 bytes, memory of **int** is 4 bytes and memory of **char** is 1 byte. Hence, 108 bytes of memory is allocated for structure variable bill.

## How to access members of a structure?

The member of structure variable is accessed using dot operator. Suppose, you want to access ageof structure variablebilland assign 50 to it. You can perform this task by using following code below:

bill.age = 50;

## Example : C++ Structure

#include <iostream>

using namespace std;

struct person { // Defining a structure

char name[50];

int age;

float salary;

};

int main() {

person p1; // Declaring a struct variable

cout << "Enter Full name: ";

cin.get(p1.name, 50);

cout << "Enter age: ";

cin >> p1.age;

cout << "Enter salary: ";

cin >> p1.salary;

cout << "\nDisplaying Information." << endl;

cout << "Name: " << p1.name << endl;

cout << "Age: " << p1.age << endl;

cout << "Salary: " << p1.salary;

return 0;

}

Here a structure person is declared which has three members. Inside **main()** function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

1. **Arrays of Structures**

Since an array can contain similar elements, the combination having structures within an array is an array of structures. To declare an array of structures, you must first define a structure and then declare an array variable of that type. For example, to store addresses of 100 members of the council, you need to create an array.

Now, to declare a 100-element array of structures of type emp, we will write:

emp employees [100];

This creates 100 sets of variables that are organized as defined in the structure emp. To access a specific structure, index the structure name. For instance, to print the name of structure 8, write:

cout << employees[7].name ;

Following is an example:

#include <iostream>

#include<string>

using namespace std;

struct person { // Defining a structure

int age;

int salary;

};

int main() {

person p1[5]; // Declaring a struct variable

int i = 0;

char a[10];

for (int i = 0; i < 5; i++)

{

cout << "Enter age: ";

cin >> p1[i].age;

cout << "Enter salary: ";

cin >> p1[i].salary;

}

cout << "\nDisplaying Information.\n" << endl;

for (int i = 0; i < 5; i++)

{

cout << "Age: " << p1[i].age << endl;

cout << "Salary: " << p1[i].salary<<endl;

}

return 0;

}

1. **Memory Alignment**

The memory to store a particular structure or object is split into blocks of determined size. This size is called **alignment**.

The C++ standard requires that members are stored in memory in the order of declaration. Multiple consecutive members can be “packed” in one block. If the next member doesn’t fit into the remained bytes of the block, it’s saved into the following block. Those unused bytes in the previous block are called “**padding**”.

Normally the alignment is determined by the compiler for each structure type. In a simple scenario where the structure doesn’t contain other structures, the alignment is the size of the largest type stored in the structure. However, in some environments a simple type can have alignment requirement that is smaller than the size of the type. For example, double can have alignment of 4. So, it would be better to say that alignment of a structure is the maximum alignment of its contained types. Alignment is only allowed to be power of two by the standard.

**Example 1:**

|  |
| --- |
| struct Alignment4  {      char a;      char b;      //char[2] - padding      int i; // max alignment 4  Alignment4():a(1),b(2),i(3){}  }; // size is 8 |

**Example 2:**

|  |
| --- |
| struct Alignment8  {      int i;      //char[4]      double d; // max alignment 8      short s;      //char[6]      Alignment8():i(1), d(2), s(3){}  }; // size is 24 |

Alignments of the types are the following: int 4, double 8, short 2. The maximum is 8 so the alignment of the structure is 8.

Memory: **01 00 00 00 cd cd cd cd 00 00 00 00 00 00 00 40 03 00 cd cd cd cd cd cd**

**Example 3 (Optimization)**

We only need 14 bytes to store Alignment8 but it takes 24. The order of the members is the problem. The block is 8 bytes and when we pack d, it has to start from the next block as there is only 4 bytes left in the current block, which is not enough for d. i and s can fit into one block but they are separated by d. BetterAlignment8 (see below) takes only 16 bytes.

|  |
| --- |
| struct BetterAlignment8  {      int i;      short s;      //char[2]      double d; // max alignment 8      BetterAlignment8():i(1), d(2), s(3){}  }; // size is 16 |

Memory:  **01 00 00 00 03 00 cd cd 00 00 00 00 00 00 00 40**

This will reduce the memory consumption and possibly improve the performance as more elements will fit into the CPU cache.

1. **File Handling (Revision)**

The standard I/O header file, *iostream*, contains data types and variables that are used only for input and output from the standard I/O device. C++ also provides a header file called *fstream*, which is used for file I/O.

The *fstream* header file contains many data types and functions. The two main data types, which we will be dealing in this lab are mentioned below:

1. *ifstream* (input file stream)
2. *ofstream* (output file stream)

Following are the five steps for file I/O:

1. Include the header file *fstream* in the program.
2. Declare file stream variables.
3. Associate the file stream variables with the input/output sources.
4. Use the file stream variables with >>, <<, or other input/output functions.
5. Close the files.
   1. **Opening a file:**

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file. An open file is represented within a program by a stream (i.e., an object of one of these classes; in the previous example, this was myfile) and any input or output operation performed on this stream object will be applied to the physical file associated to it.  
  
In order to open a file with a stream object we use its member function open:  
  
open(filename,mode);  
  
Where filename is a string representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

|  |  |
| --- | --- |
| ios::in | Open for input operations. |
| ios::out | Open for output operations. |
| ios::binary | Open in binary mode. |
| ios::ate | Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file. |
| ios::app | All output operations are performed at the end of the file, appending the content to the current content of the file. |
| ios::trunc | If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one. |

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open:

|  |  |  |
| --- | --- | --- |
|  | ofstream myfile;  myfile.open ("example.bin", ios::out | ios::app | ios::binary); |  |

Each of the open member functions of classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

|  |  |
| --- | --- |
| **class** | **default mode parameter** |
| ofstream | ios::out |
| ifstream | ios::in |
| fstream | ios::in | ios::out |

* 1. **Closing a file:**

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function close. This member function takes flushes the associated buffers and closes the file:

|  |  |  |
| --- | --- | --- |
|  | myfile.close(); |  |

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.  
  
In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function close.

**Example: Writing to text file:**

|  |
| --- |
| # include <iostream >  # include <fstream > // File handling library.  using namespace std;  int main ()  {  int x = 3;  float f = -2.348193 f;  char c = '^';  ofstream OutFile ; // File handling object .  OutFile . open ("abc.txt ", ios :: out ); // Open " abc.txt" for writing ( ios :: out ). This will //overwrite existing file  OutFile << x << " " << f << " " << c << " " << endl ;  OutFile . close (); // Close file .  return 0;  } |

**Example: Reading from text file:**

|  |
| --- |
| # include <iostream >  # include <fstream > // File handling library .  using namespace std;  int main ()  {  int x;  float f;  char c;  fstream InFile ; // File handling object .    InFile . open ("abc.txt ", ios :: in); // Open " abc.txt" for reading. Mode ‘ios :: in’ is  // optional. If no mode selected then by default  // ios :: in | ios :: out will be set)  InFile >> x >> f >> c;  InFile . close (); // Close file .  cout << " Data read is ... " << endl ;  cout << "x = " << x << ", f = " << f << ", c = " << c << endl ;  return 0;} |

**Practice Problems:**

FAST university want to create a Student Information System for 2021 Batch students. Following is the data they want to store for each student:

1. Name (char array)
2. Roll No.
3. Department (char array)
4. CGPA (float)

You program should be able to perform following task:

1. Read data of students from a text file and store it in an array.
2. Print data of all students.
3. Search a student by its Roll No and display its information.
4. Sort student data on basis of CGPA.
5. Ask user the information of late admission students.
6. Append this new student data in text file.

#include<iostream>

#include<fstream>

struct student{

char name[50];

char rollno[50];

float gpa;

char department[50];

};

struct roll{

char rollno[];

};

struct late{

char name[50];

char rollno[50];

float gpa;

char department[50];

};

void display(student s[9]);

void search(student s[9],roll r);

void sort(student s[9]);

void admission(student s[9],late l[],int&);

using namespace std;

int main()

{

int x;

student s[10];

roll r;

ifstream show;

show.open("Lab\_4\_Students\_data\_2021.txt");

for(int i=0;i<=8;i++)

{

show>>s[i].name;

show>>s[i].rollno;

show>>s[i].department;

show>>s[i].gpa;

}

display(s);

search(s,r);

sort (s);

cout<<"enter the no of late admission students "<<endl;

cin>>x;

late l[x];

admission(s,l,x);

}

void display(student s[9])

{

for(int i=0;i<=8;i++)

{

cout<<s[i].name<<" "<<s[i].rollno<<" "<<s[i].department<<" "<<s[i].gpa<<endl;

}

cout<<endl<<endl<<endl;

}

void search(student s[9],roll r)

{

int z=0,k=0;

cout<<"enter roll no"<<endl<<endl;;

cin>>r.rollno;

cout<<endl;

for(int i=0;i<=8;i++)

{

for(int y=0;r.rollno[y]!='\0';y++)

{

k++;

if(r.rollno[y]==s[i].rollno[y])

{

z++;

}

}

if(z==k)

{

cout<<"data of roll no entered is"<<endl<<endl<<endl;

cout<<s[i].name<<" "<<s[i].rollno<<" "<<s[i].department<<" "<<s[i].gpa<<endl;

break;

}

z=0,k=0;

}

if(z==0)

cout<<"wrong rollno"<<endl;

}

void sort(student s[9])

{

for(int z=0;z<=5;z++)

{

for(int i=0;i<=8;i++)

{

if(s[i].gpa<s[i+1].gpa)

{

swap(s[i],s[i+1]);

}

}

}

cout<<endl<<endl<<endl;

cout<<"sort data is"<<endl<<endl;

for(int y=0;y<=8;y++)

{

cout<<s[y].name<<" "<<s[y].rollno<<" "<<s[y].department<<" "<<s[y].gpa<<endl;

}

cout<<endl<<endl<<endl;

}

void admission(student s[9],late l[9],int &x)

{

for(int i=0;i!=x;i++)

{

cout<<"enter late admission student name"<<endl;

cin>>l[i].name;

cout<<"enter late admission student rollno"<<endl;

cin>>l[i].rollno ;

cout<<"enter late admission student department"<<endl;

cin>>l[i].department;

cout<<"enter late admission student gpa "<<endl;

cin>>l[i].gpa;

}

ofstream write;

write.open("Lab\_4\_Students\_data\_2021.txt");

for(int k=0;k<=8;k++)

{

write<<s[k].name;

write<<s[k].rollno;

write<<s[k].department;

write<<s[k].gpa<<endl;

}

for(int y=0;y!=x;y++)

{

write<<l[y].name;

write<<l[y].rollno;

write<<l[y].department;

write<<l[y].gpa;

}

}

Create a structure named Address, that contains street number, Sector, City, and country. Next, create a structure named Employee which should contain a structure Address along with the name, ID, and salary as its members. In main () using function, input and display the data of two employees enter by the user.

#include<iostream>

using namespace std;

struct Address{

char streetno[15];

char sector[20];

char city[20];

char country[20];

};

struct Employee{

Address addr;

char name[50];

char id[15];

int salary;

};

void display(Address addr,Employee emp[]);

int main()

{

Address addr;

Employee emp[2];

display(addr,emp);

for(int i=0;i<=1;i++)

{

cout<<emp[i].name<<endl;

cout<<emp[i].id<<endl;

cout<<emp[i].salary<<endl;

cout<<emp[i].addr.streetno<<endl;

cout<<emp[i].addr.sector<<endl;

cout<<emp[i].addr.city<<endl;

cout<<emp[i].addr.country<<endl;

}

}

void display(Address addr,Employee emp[2])

{

int i;

for(i=0;i<=1;i++)

{

cout<<"enter the name of employee"<<endl;

cin>>emp[i].name;

cout<<"enter the id of employee"<<endl;

cin>>emp[i].id;

cout<<"enter the salary of employee"<<endl;

cin>>emp[i].salary;

cout<<"enter the streetno of employee"<<endl;

cin>>emp[i].addr.streetno;

cout<<"enter the sector of employee"<<endl;

cin>>emp[i].addr.sector;

cout<<"enter the city of employee"<<endl;

cin>>emp[i].addr.city;

cout<<"enter the country of employee"<<endl;

cin>>emp[i].addr.country;

}

}

Show how data can be aligned in structure, so minimum number of bytes are required. Use ***sizeof*** command to check number of bytes occupied by each structure.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Example*** | ***No of bytes required by structure.*** | ***Structure after alignment*** | ***No of bytes required after data alignment*** |
| struct test1  {   short s;      int i;      char c; }; | 12 | Struct test1  {int I;  Short s;  Char c;}; | 8 |
| struct {  char a[3];  short int b;  long int c;  char d[3];  }; | 16 | struct {  long int c;  short int b;  char a[3];  char d[3];  }; | 12 |
| struct student{  char a;  int c;  char b;  }; | 12 | struct student{  int c;  char a;  char b;  }; | 8 |
| struct structure2  {         int id1;         char name;         int id2;         char c;         float a;  }; | 20 | struct structure2  {  int id1;  int id2;  float a;  char c;  }; | 16 |

**Bonus Questions:**

Declare a structure to represent a complex number (a number having a real part and an imaginary part). Write C++ functions to add, subtract, multiply and divide two complex numbers. Implement this task using the function. Store answer in a text file.