Discipline Courses-I
Semester-I
Paper: Programming Fundamentals
Unit-IV
Lesson: User defined data types-I,II,III

Lesson Developer: Sushila Madan
College/Department: Lady Shri Ram College, University of Delhi

Table of Contents

1.1

41 USER DEFINED DATA TYPES -I

Till now you know about the predefined data types such as int, char, and float. Though these data types are sufficient to solve any problem there may occur situations when these data types are not adequate to solve a particular problem. C++ provides a mechanism for users to create their own data types which greatly enhances the flexibility of programming language. You have observed that a simple data type can store only one value at a time. In contrast, in a structured data type, each data item is a collection of other data items. Similarly union is used to efficiently manage the storage space by sharing memory locations by variables.

Learning objectives

After this lesson you would be able to:

- Know what is enumerator and to initialize enumerator
- Know what is typedef
- Understand Structure
- Learn to use Structures and arrays of structures
- Differentiate between typedef, enum and structures
- Learn to make use of complex nested structures
- Learn Union and anonymous Union
- Understand the difference between Union and Structure

41.1 User Defined Data Types I : Enumerated Data Type

You have learnt that C++ simple data type is divided into three categories: integral, floating point and enum. In subsequent units you worked mainly with integral and floating data types. In this section you will learn about enum type. The C++ language provides users great flexibility on re-defining these existing types or re-defining new ones. To define your own special data types of any name such as flag, student, friend etc. Any data type which is not predefined is a user data type. You can assign any values to the data types which are defined by you. Sometimes you want to express the idea that a variable can use for a specific purpose and should only be able to have a small number of values. For this purpose you use data types which is defined by us. Enumeration is one such data type. It makes our code easier to understand. Enumerated types, on the other hand, made the code more self-documenting.

41.1.1 What is an enumerated Data Type?

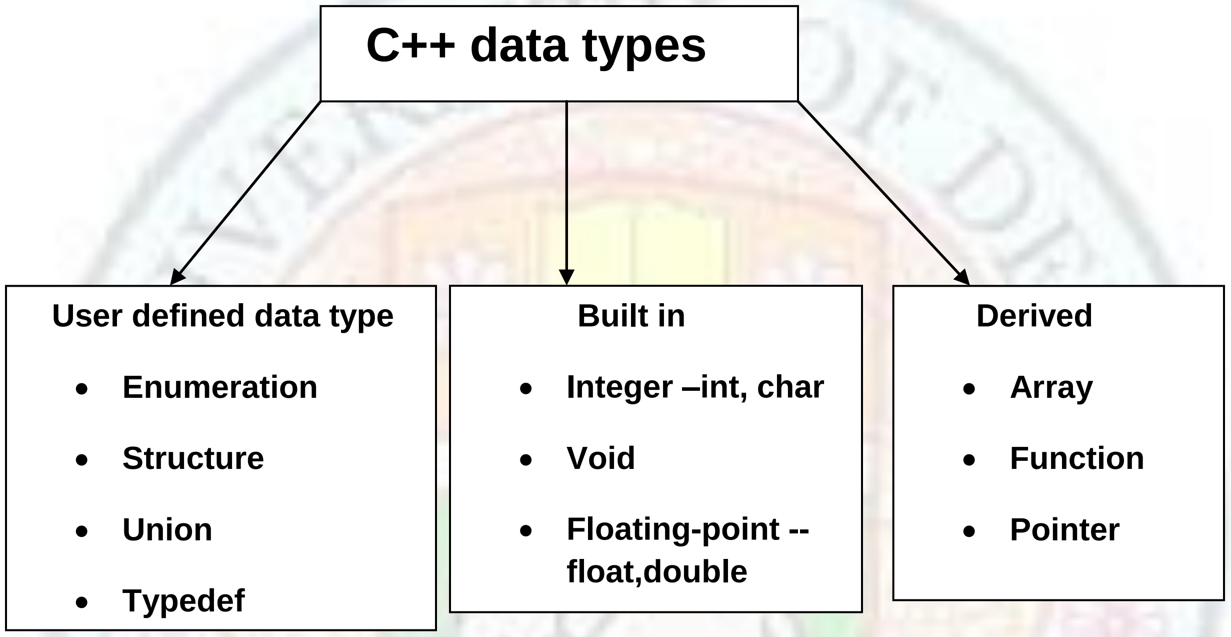
An **enumerated** data type is a user defined data type. Enumerated data type is mostly used when you know the list of values that a data type can have. For instance, if you want to define different sections of class **nursery** and you already know that nursery class have three sections i.e **oyster, coral and pearl**. You can define an enumerated data for the class nursery. Then the variable for this type can be declared which contain only these values at a time.

```
enum nur_class {oyster,coral,pearl} sec1, sec2;
```

In the above example **nur_class** is the enumerated data type, **sec1 and sec2** are the variable declared for this data type and **oyster, pearl, coral** are all the values which these variables can hold.

Value addition: diagram

Heading text –Different data types



You can see from the above figure that in C++ we have three different kinds of data types i.e. user defined, built in and derived Here we will study about user defined data types.

Source: Programming with C++, D Ravichandran, Tata McGraw-Hill Publishing

41.1.2 Defining and Declaration of Enumerations

To define an enumeration type , you need the following items:

- A name for the data type
- A set of values for the data type

- A set of operations on values

Enumeration is defined by the user with the syntax

enum typename{enumerator-list}

Here **enum** is the C++ keyword and it signals the start of the enumeration type, typename is the name of the type that is being defined and the enumerator list contain all the names which a variable can hold. For example, the following defines the enumeration type exam, specifying the three possible values that a variable of this type can have

enum Exam{maths,eng,hindi};

You can also declare variables of this Exam type e1, e2.

Exam e1,e2;

/*An enumerated data type Exam has been defined and e1 is declare as a variable of type exam.*/

In the above example e1 can hold values maths, eng and hindi. These all values are called **enumerators**. The **enum** type Exam has three **enumerators: maths, eng, hindi**.

Value addition: Some more examples

Heading text – Easy code

Body text:

- Enumeration type are usually defined to make our code easier to understand.

Here are few examples:

```
enum Sex{female,male};

enum Day{mon,tue,wed,thur,fri};

enum Month{jan,feb,mar,apr,may};
```

- It is advisable to Capitalize the first letter of each name in the user-defined types. It helps distinguish standard C++ types like float and string from user defined types such as Month, Day, etc..

41.2 Assigning values to Enumerator

enum specifier automatically assigns values to all its enumerator ranging from 0,1,2,3,4 and so on.

```
enum Drinks {pepsi, maza, limca, coke};

// An enumerated data type drinks has been defined

    pepsi=0  ;        //enumerator pepsi will have value 0

    maza =1 ;              //maza will have value 1

    limca=2   ;              //limca will have value 2

    coke=3 ;          //coke will have value 3
```

You can also assign variable with one of the type value

```
    Drinks d1, d2;   //variables declared of type drinks

    d1=pepsi ;  //d1 is variable which is assigned to type value pepsi

    d2=limca ;  //d2 is variable which is assigned to type value limca
```

The enumerators also can also be used with arithmetic operators

For instance

```
    d1=pepsi;

    d2=limca;

    int diff=d2-d1;         //integer arithmetic

    cout<<diff;
```

The above code assign 0 to pepsi and 2 to limca. The variable diff stores d2-d1 i.e 2-0 which is 2. The cout statement will print value of diff which is 2.

value addition: Important Facts
Heading text enumerator with same value

Body text: Since enumerators are simply integer constants, it is legal to have several different enumerators with the same value: **enum** Qanswer{YES=1, OK=1, TRUE=1, NO=0, FALSE=0}; this would allow the following code: Qanswer, ans; cin>>ans; .. … if (ans==YES) cout<<" He said your answer is ok"<< endl;
Source: self made

41.3 Changing Default Values

Observe, if integer values are assigned to some of the enumerators, then the one that follow are given consecutive value. For example:

enum Month{JAN=1, FEB, MARCH, APR, MAY JUN, JUL}

will assign the numbers 1 through 7 to the seven months.

You can also change the default values of the member or enumerator. You all know that by default first number has value 0. This default value can be explicitly changed by assigning integer value to the enumerator. For instance

enum colour{red, blue=9, pink, white};

```
    red =0 ;              // by default red=0

    blue=9 ;              // value 9 has been assigned to blue
```

```
        pink=10;

        /* the values will increase consecutively from the previous  value */
```

You can also give different values to different enumerators.

```
enum month{jan=1,feb=0,mar=7,apr=8,may=4,jun=6}
```

In the above example each month has different values. Since enumerators are integer constants, it is legal to have different values for different enumerators. You can also give same value to several different enumerator.

```
enum month{jan=1,feb=1,mar=7,apr=8,may=8,jun=6}
```

In the above example both jan and feb have value 1;and apr and may have same value of 8.

You all know that constant value does not change throughout the program. Enumeration consider enumerators as integer constant. While defining enumerators you should always keep in mind that the enumerators should be valid identifiers. For instance:-

enum grade{f, d, c, c+, b-} /* This definition is not valid because the characters '+','-' cannot be used in identifiers.*/

Enumerations can also be anonymous in C++:

```
enum { X= 2, Y=80, Z=100, P=500}
```

This is just a convenient way to define integer constants.

Enumerated types can also prevent a programmer from writing illogical code such as performing mathematical operations on the values of the enumerators. Generally printing of enum variable is as good as printing integer. For example,

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
enum month{JAN, FEB, MAR, APR, MAY, JUN};
int  i;
for(i=JAN;i<=JUN; i++)    //this will replace JAN, JUN with equivalent
```

```
 cout<< i <<"  ";   // numeric code
getch();
}
```

Output will be:

```
0 1 2 3 4 5
```

value addition: important Facts and a Program

Heading text Relational operator with enumerator

Body text:

You can also use enumerator with relational operators.

```
#include <iostream.h>
#include<conio.h>
void main()
{
clrscr();
enum week_day day{m=1,t,w};
int no;
cout<<"\n enter the number";
cin>>no;
if(no==w)
cout<<"\n day is Wednesday";
else
cout<<"\n day is not Wednesday";
getch();
}
```

In the above example, we have used the relational operator '=='. By default w has value 3, therefore if user enters 3 as input then 'day is Wednesday' will be printed .

Source: self made

41.4 Definition of typedef

The C++ language allows you to create a new name for a data type without changing the meaning of that data type. This allows you to name a data type with a name that is easier for you. This technique doesn't change anything about the data type, it only gives it a new name you can use in your program. To re-define the name of an existing data type, you use the **typedef** keyword. typedef literally stands for **'type definition'.**

typedef int amount;

In the above example **amount** is the new name for the data type **int**.

41.5 Declaration of typedef

The syntax for typedef data type is as follows

typedef *KnownDataType NewName*;

The **typedef** keyword is required to let the compiler know that you are redefining an existing data type. The *DataType* is any of those you have learned so far. It could be an **int**, an **unsigned int**, a **char**, a **double**, etc.

typedef int amount;

amount car,bus;

In the above declaration **amount** will be used as same as **int.**

Value addition: example

Heading text – new name for int datatype

Body text:

```
#include <iostream.h>
int main()
{
typedef int amount;
amount bus,car;
cout<<"\n enter the amount of car";
cin>>car;
cout<<"\n enter the amount of bus";
cin>>bus;


cout<<"\n amount of car is: <<car;
cout<<"\n amount of bus is:<<bus;
return (0);
}
```

You can see amount is used as a data type int in the above example.

Here are examples of creating new names of existing data types using **typedef**:

```
typedef unsigned int UINT;

typedef unsigned char UCHAR;

typedef unsigned long ULONG;

typedef long double LONGDOUBLE;
```

Value addition: Faq

Heading text – Difference between enumeration and type def

While using enumerated data type you are making special integers that fall within range, while a typedef statement doesnot define a new type it only provides a synonym for an existing type i.e., redefining data type with new name.

42 User Defined Data Types II

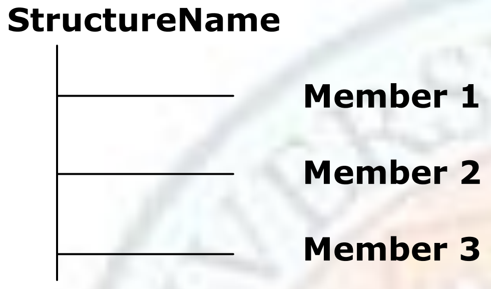
Till now you have learned about the enum and typedef user defined data type. Now we will discuss another kind of user defined data type which is known as structure. Structure makes the program more modular ,which is easier to modify because its design makes things more compact. A structure is a grouping of several variables under one name.

42.1 Structures

Structure is a collection of variables under a single name. Variables can be of any type: int , float, char etc. Some logical elements need to be stored under one roof, for instance, elements storing employee information(e.g. empname, empage, empdesignation, empid) need to be processed together. To handle such situation we use structures.

Declaration and Syntax of Structures

A structure can be graphically represented as given below:



The symbolic representation of the structure is:

```
struct  user _defined_name{
                        member 1
                        member 2
                        member 3
                        };
```

A structure definition is specified by the keyword **struct.** The keyword **struct** tells the compiler that a structure is being defined.

The general format for declaration of a structure is:

```
struct user _defined_name{
                        Data type member 1;
                        Data type member 2;
                        Data type member3;
                    };
```

The keyword **struct** and the braces are essential features. The user defined names are usually used. The data type is any valid data type in C++.

The following example shows how to define a structure (say date)

```
struct date{   //  The date is the structure tag .

                   int day;

                   int month;

                   int year;

               };
```

In the above example, we have not declared Structure variable, only the form of the data has been defined.

```
struct date{

                       int day;

                       int month;

                       int year;

               }joiningdate;
```

The above statement defines a structure type called date and declares a structure variable joiningdate.

Heading text – structure members
In a structure all the variables defined in a structure are also known as its members. In the above example, we can also say that day, month and year are the members of date.

Now let us write a structure definition statement to hold employee related information, like emp id, emp name, emp salary, emp profile and also declare a variable senior emp of the same type.

```
struct employee              // structure tag employee is declared

    {

        int empid;            // data types and variables are declared

        char empname[30];

        char empprofile[30];

        int empsalary;

    } senioremp;                  // structure variables is declared
```

All the members i.e. empid, empname, empprofile, empsalary are element of structure variable senioremp.

Value addition: Important Distinctions

Heading text – Structures in C AND C++and difference between C++ struct and C++class

In C and in C++ all structure members are public by default. Public members are accessible for all and everyone. A structure in C can not have functions.

Difference between C++ struct and C++class
A **C++ struct** is essentially same as **C++ class**. The only significant difference between **C++ struct** and **C++ class** is with the default access specifier assigned to the members.

C++ classes can be defined without explicitly specifying its member access specifier. For example:

```
class  Student

{ int totalmarks noofstudent ;

};
```

Is a valid definition of Student class. Since access specifier for its data members totalmarks and noofstudent is not specified, it is set by default to **private.** If we make it a **struct** instead of a class as given following:

```
struct Student

{ int totalmarks noofstudent ;

};
```

Then the data members are set by default to be public. But if you explicitly specify the access specifier then this problem can be solved

```
struct Student

     { private:

           int totalmarks noofstudent ;

     };
```

42.2 Accessing Structures

In structures, all the members are elements of structure variable and can be accessed by **dot operator .** The syntax is

Structure variable name. member name

```
senioremp.empid

senioremp.empname
```

Let us make a structure of employee and cout and cin its member.

```
# include<iostream.h>

#include<conio.h>
```

```
struct emp{                              //structure emp is declared
                        int empid;
                        char empname[30];
                        char empprofile[30];
                        int empsalary;
              }worker;
void main()
{
clrscr();
cout<<"\n enter employee id";
cin>>worker.empid ; //member empid is accessed by a structure variable
cout<<"\n enter employee name";
cin>>worker.empname;
cout<<"\n enter employee profile";
cin>>worker.empprofile";
cout>>"\n enter employee salary";
cin>>worker.empsalary;
getch();
}
```

42.2.1 Intializing Structure Elements

You can also assign values to the structure elements in two ways

1. **structure variable.member**=**value**

worker.empid=100;

worker.empsalary=2000;

2. If you have declared a structure variable then you can initialize all the members of the structure together like:

   structure tag structure variable={value1,value2,value3};

   emp worker ={100, "Ramesh", "Manager",2000};

   In the above example, you can initialize values in the same way as you do in the arrays.

- A program to intialize the members of a structure and to display the contents of the structure on the screen:

```
#include<iostream.h>

void main()

{

    struct school

            {

                    int rollno;

                    int age;

                    int marks;

                    char sex;

            }student;

      school student={46,15,300,'M'};

      cout<<"\n Rollno ="<<student.rollno;

      cout<<"\n Age="<<student.age;
```

```
        cout<<"\n Marks="<<student.marks;

        cout<<"\n Sex="<<student.sex;

        }
```

Output of the above progam

```
Rollno=46

Age=15

Marks=300

Sex=M
```

Value addition: Important Fact
If some of the members are not initialized, then the C++ compiler will automatically initialize them to zero.

42.3 Complex Structures

Structures can be complex or nested . A structure may consist of an element that itself is complex i.e., it is made up of arrays, structures etc. Thus an element of a structure may even be an array or a structure in itself. A structure consisting of such complex element is called a **complex structure.**

```
struct player {                         //first structure type is declared

          char name[40];                // name of player
          char league[5];               // name of league
          };

struct team  {                          //second structure type is declared

               player play;                       // nested structure
               char position[5];           // team position
               char owner[5];          // team owner

             } teama;             //variable for second structure is declared
```

42.3.1 Accessing nested structure's members

To process the individual elements in a nested structure, first represent the structure variable name and the first structure and then the field name of the first structure. Nested structure members are also accessed with a dot operator. Syntax is:

(Struct variablename).(struct firstname).fieldname = variable

```
struct addr{

            int houseno;

            char city;

          };

struct emp{

                  int empno;

                  char name;

                  addr address;   // address is a structure variable itself and
                                   //  it is a member of another structure.
                  float basic_pay;

            }worker;
```

To access the houseno member of address structure which is an element of another structure worker, you shall write

worker.address.houseno

To initialize houseno member of address structure,you can write

worker.address.houseno=68

- Program to read values and initialize value in nested structure.

```
#include<iostream.h>

#include<conio.h>

struct c1 {

        int course_no;

        int course_fee;

     };

struct c2{

        int stu_no;

        c1 btech;

        c1 mtech;

} course;

void main ()

{

        course.stu_no=40;

        course.btech.course_fee=400;

        course.mtech.course_fee=400;

        int y=course.btech.course_fee + course.mtech.course_fee;

        cout<<"\n Student no ="<<course.stu_no;

        cout<<"\n Total course fees  ="<<y;

        getch();

}
```

Output of the above program is:

Student no = 40

Total course fees =800

Value addition:

Difference between class and structures

1. In C++ access specifiers can be used in structures to implement data hiding
2. C++ structures can have data as well as functions.

42.4 Array of Structures

A member of a structure can be an array data type also. An array is a group of identical data which are stored in consecutive memory locations in a common variable. A similar type of structure placed in a common variable is called arrays of structures. For instance

```
struct employee

{

    char name [20];

    char profile[20];

    char add[30];

} worker[3] ;   //The worker is an array of  structures
```

The three elements of worker array can also be initialized when the array is defined as:

```
worker[3]= {{"l.k","manager","35"},

                {"d.k", "secretary","45"},

                 {"m.k","clerk","79"}};
```

The first record will have value(name=l.k, profile=manager,add=35), second record will have value(name=d.k, profile=secretary, add=45) and the third record will have value(name=m.k,profile=clerk,add=79)

- A program to initialize members of an array of structures and to display the contents of all the structures.

```
# include<iostream.h>

void main()

{

struct school{

                int age;

                char sex;

                int rollno;

           }student[3]={{12,'m',80},{13,'f',70},{13,'m',80}};

     for(int i=0;i<=3;i++)

  {

           cout<<"\n contents of structure are :"

           cout<<"\n roll no="<<student[i].rollno;

           cout<<"\n sex="<<student[i].sex;

           cout<<"\n age="<<student[i].age;

    }

}
```

Value addition: Notes
Distinction between Arrays and Structures

Arrays are used to store large sets of data and manipulate them but the disadvantage is that all the elements stored in an array must be of the same data type. If we need to use a collection of different data type items it is not possible using an array. When you require using a collection of data items of different data types you can use a structure. Structure is a method of packing data of different types. A structure is a convenient method of handling a group of related data items of different data types.

43 User Defined Data Types III

By now you know about the structures, classes, enum, typedef. Now you will learn about one more data type which is known as union. Union is similar to structures with a difference in the way the data is stored and retrieved.

43.1 What is Union?

Union is a memory location that is shared by two or more different variables, generally of different types. Union is similar to the structure in a way that it has similar syntax as structure. A union can take value of one of its member at a time. In union all the members start at same memory location

The symbolic representation of union is:

```
Union user defined name{

                                    Member1;

                                    Member2;

                                    Member3;

                 } variable1, variable2;
```

Value addition: Faq

Heading: Important note

- In C++, a union is a limited form of the class type. It can contain access specifiers (public, protected, private), member data, and member functions, including constructors and destructors. It cannot contain virtual member functions or static data members.

Default access of members in a union is public

- The *member-list* of a union represents the kinds of data the union can contain. A union requires enough storage to hold the largest member in its *member-list*.

43.1.1 Referencing Union

A "union declaration" specifies a set of variable values. The variable values are called "members" of the union and can have different types.

The keyword **union** is used to declare **union data type.**

```
union student

{

      int rollno;

     char gender;

} s1;
```

The list of members provide the data type with a description of the objects that can be stored in the union.

You can see that the declaration of union is same as of structures. A dot operator is used in between the union variable name and the field name.

```
s1.rollno

s1.gender
```

Value addition: Faq
Heading: what is a union in computer science?
• The name "union" stems from the type's formal definition. If one sees a type as the set of all values that type can take on, a union type is simply the mathematical union of its constituting types, since it can take on any value that its fields can. Also, because a mathematical union discards duplicates, if more than

one field of the union can take on a single common value, it is impossible to tell from the value alone which field was last written.

- In computer science, a union is a data structure that stores one of several types of data at a single location. There are only two safe ways of accessing a union object. One is to always read the field of a union most recently assigned; tagged unions enforce this restriction. The other is to only access functionality common to all types in the union. For example, if the fields are all subtypes of a common supertype, then it is always legal to perform operations on the union object that one can perform on the supertype.

43.2 Memory ~~location of~~ allocation to union data type

In union the largest member is allocated the memory location. While creating a struct each variable occupies the space in memory as assigned to it. When a union is declared the compiler will assign the space to the largest variable i.e. if you define a union student having s1 object defined in the previous section, the compiler would assign the same memory space to all members; and the compiler would find out which member needs the largest space, in this case that would be the int variable because an int variable occupies **2 Bytes**.

For instance; if student is defined as shown below

```
union student
 {
      int rollno;
      float marks;
      char name;
  }s2;
```

Then

```
s2.marks;
  s2.name;
  s2.rollno;
```

represent elements of s2

In the above example union student is having three variables (int, char and float) and creates a union object s2 of union type student. In the union student all integer, float, character will share the same memory location. Now compiler would find out which member needs the largest space and in this case float will need the largest space because float occupies 4 bytes. So the size occupied by any object of the student will be 4 bytes.

Value addition: Faq

Heading:Data types and size in bytes

Char=1byte

Short=2 bytes

Int=2byte

Long=2 0r 4 bytes

Float=4 bytes

Double=8 bytes

Long double=12 bytes

43.3 ~~Intialization~~ Initialization of Union

Union can be initialized. However , a union has only one active member at any given time. If you initialize all of the members of a union at the same time, the result is unreliable; since all members are using the same memory space, that space cannot accommodate all member values at the same time.

- A program to initialize the members of a union and to display the contents of the union.

```
# include<iostream.h>

# include<string.h>

void main()

{

union student

{

        char name[30];

        int rollno;

        float marks;

        double grandtotal;

} s1;

        strcpy(s1.name, “Rita”);

        s1.rollno=80;

        s1.marks=90;

        s1.grandtotal=300;

        cout<<”\n Name =”<< s1.name;

        cout<<”\n Roll No=”<< s1.rollno;

        cout<<”\n Marks=”<< s1.marks;

        cout<<”\n Grand total=”<< s1.grandtotal;

}
```

Output of the above program will be

```
Name=some garbage value

Roll No= some garbage value

Marks=some garbage value

Grand total=300
```

In the above program, the union consists of four members such as int, character array, float and double. Only grandtotal is displayed correctly as it is initialized in the end and for the rest members, some random garbage value will be displayed. The character array is of 30 bytes which is largest among the others i.e int,char,float.

Value addition: Faq

Heading:Important note

- Pointers to union may be used just like pointers to structures.

43.4 Structure as a member of a Union

Union can also be the member of a structure and structures can be member of a union. The notation for accessing a member of a union in a structure (or vice versa) is identical to that for nested structures.

- A program to declare a member of an union as a structure data type and to display the content of the union in C++

```
#include<iostream.h>

void main()

{

struct student{

        int marks1;

        int marks2;
        int marks3;
};
union result{
        char a;
        char b;
        char c;

        struct student s1;
};
union result grade;
        grade.a='A';
        grade.b='B';
        grade.c='C';
        grade.s1.marks1=90;
        grade.s1.marks2=80;
        grade.s1.marks3=60;
  cout<<"\n Marks1=:"<<garde.s1.marks1<<"Grade="<<grade.a;
  cout<<"\n Marks2=:"<<grade.s1.marks2<<"Grade="<<grade.b;
 cout<<"\n Marks3=:"<<grade.s1.marks2<<"Grade="<<grade.c;
}
```

Output will be:

Marks1=90 Grade=some junk value

Marks2=80 Grade= some junk value

Marks2=60 Grade= some junk value

Thus you can see that a structure can also be used as a member of a union. But since members of structure s1 are initialized last, therefore valid values are present in members of structures1 only and junk values are present in other members of union grade.

Value addition: Important note

Heading: Difference between structures and union

- A C++ struct will make sure to allocate space for each and every member in the struct. In CONTRAST THAT with a ***UNION***. A UNION uses the SAME EXACT PIECE OF MEMORY for ALL of the members inside of it.
- Union allocates the memory equal to the maximum memory required by the member of the union but structure allocates the memory equal to total memory required by the members.
- In Union, one block is used by all the members of union but in case of structure, each member has their own memory space

For eg:

```
struct example
{
  int integer;
  float floating_numbers;
}
```

the size allocated here is sizeof(int)+sizeof(float);
where as in a union

```
union example
{
  int integer;
  float floating_numbers;
}
```

size allocated is the size of the highest member.
so size is=sizeof(float);

43. 5 Anonymous Union

A special type of union has been added in C++, are called anonymous union. An anonymous union defines a union data type without a user defined name or tag in its declaration.

The general syntax of the anonymous union is:

```
union {

       Member1;

       Member2;

};
```

For example,

```
union{
          int x;
          char ans[3];
         };
```

both x and ans[ ] share the same memory location and can be accessed directly such as :

```
x = 20;
ans[0] = 'Y';
```

- A program to demonstrate the anonymous union declaration in C++

```
//anonymous union

#include<iostream.h>

void main()

{

  union{

              int a;

              float b;
```

```
    };
    cout<<"\n enter the value of a";
    cin>>a;
    cout<<"\n enter the value of  b";
    cin>>b;
    cout<< <<"\n content of union";
    cout<<"a="; << a<<endl;
    cout<<"b="; << b<<endl;

}
```

In the above program, union is declared without tag name.

Value addition: Did you know?
Heading:Important note
● Simply omitting the union name in the declarations does not make the union an anonymous union. For a union to qualify as an anonymous union, the declaration must not declare a variable of union type

Summary

- There are three types of data type i.e. derived data type, user defined data type, built in data types.
- Built in data types are for example, int , char, float, double.

- The reason for providing so many data types are to allow programmer to take advantage of hardware characteristics.
- Enumeration is an alternative method for naming integer constant.
- typedef is renaming data type with new name.
- A structure is a collection of variables of different data types referenced under one name.
- Structure can be complex.
- Structure elements can also be initialized.
- An array can also be used in structures.
- Union is a memory location that is shared by two or more different variables.
- The keyword **union** is used to declare **union data type.**
- In union the largest member allocated the memory location. While creating a struct each variable occupies the space in memory as assigned to it.
- If you initialize all of the members of a union at the same time, the result is unreliable.
- Union can also be the member of a structure and structures can be memberof a union.
- You can also define union data type without union tag. Such a union known as anonymous union.

Exercises

1. Define enumeration and typedef.
2. Differentiate between data types and user defined data types.
3. What are the advantages of enumeration?
4. Differentiate between typedef and enumeration.
5. Write a program to define the variables using type def and to display the contents of the variable.
6. Define structure.
7. Differentiate between structure and classes.
8. What is meant by array of structure?
9. What is meant by members of structure?
10. Write a program to demonstrate the anonymous union declaration.
11. What is the difference between union and structure?
12. How union can be initialized?
13. How many data item can be stored in a Union at any given time?
14. Define enumeration type, triangleType, that has the value scalene, isosceles, equilateral and notriangle. Write a program that prompts the user to input the length of the sides of a triangle and output the shape of the triangle.

15. Define the various ways of declaring the definitions if player is a structure, GAMES is a union and martialstatus is an enum
16. Develop a program in C++ to create the following items using a structured data type:
    Name of patient
    Sex
    Age
    Ward number
    Bed number
    Nature of the illness
    Date of admission
17. What is the advantage of declaring an anonymous union in C++?

Glossary

Array is a named list of finite number of similar data elements.

Data type means to identify the type of data.

Derived data types are those which are assigned by the user.

Enumeration is a method assigning integer constant to a variable

Enumerators are the actual values defined in the enumerator-list

Nested structure means structure within a structure.

Structure is a collection of variables of different data types referenced under one roof.

Union stores the value of different data types in a single location.

.

References

- Programming with C++, D Ravichandran, Tata McGraw-Hill Publishing
- Let us C++, Yashwant Kanetkar, BPB Publication
- C++ Programming Language D.S. Mallik CENGAGE LEARNING