Unit 5

Advanced Features

1. Unions

In the previous chapter, we studied the concept of structures. A union is similar to a structure but with a small difference. The union also is a group of elements which can be of different types like a structure but the difference is that only one of its members can be used at a time. It allows memory to be shared by different types of data.

Definition

A union is a variable that contains multiple members of possibly different data types grouped together under a single name. However, all of them share the same memory area. Hence, only one of the members is active at a time.

1

Oct. 2017 – *1M*
Define union.

1.1 Declaration

A union is declared in the same way as a structure except that the keyword 'union' is used instead of 'struct'.

1

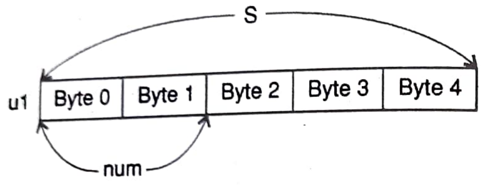
Apr. 2018 – *1M*
How is a union declared?

Syntax

```
union tag
  {
     union_members;
  } variable;
```

Example

```
union u
  {
    char s[5];
    int num;
  } u1;
```



The variable u1 will be allocated memory to accommodate the largest member of the union. In this *example*, it will be allocated 5 bytes. Only one member, i.e., either the string or the integer num will be stored and can be accessed at a time. Both do not exist simultaneously.

Only one member of a union can be used at a time.

Typically, a union is not used independently. It is often used within a structure. Let us consider an *example* to store smart-phone information. We want to store the following fields:

i. Name of the company
ii. IMEI number
iii. Type of phone (Android or IPhone A/I)
iv. If Android, store Android OS version name
v. If iPhone, store iPhone version number

```
struct smartphone
{
    char company[20];
    char IMEI[16];
    char type;
    union u
    {
      char android_os_name[20];
       int iphoneversion;
    }info;
};
```

Here, the union allows only one member to be stored i.e. either android_os_name or iphoneversion. *For example:*

```
struct smartphone s1;
s1.company= "Samsung";
s1.IMEI="152076249800002";
s1.type='A';
s1.info.android_os_name="Marshmallow";
```

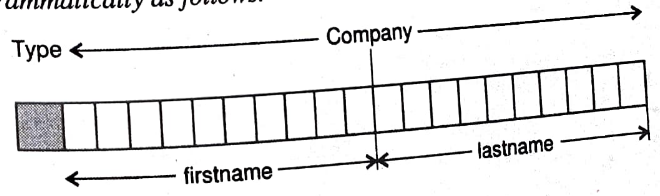
Let us consider another *example*. Suppose we want to store the following contact information:

i. Type of contact (C/P i.e. Company or Person).
ii. If company, store company name.
iii. If person, store first name and last name of person.

In this case, we can use a structure which contains two fields: type and details. If the type is 'C', we want to store the details of the company. If the type is 'P', we want to store two pieces of information for a person, i.e., first name and last name. Hence, the 'details' field will be a union.

```
struct person
{
   char firstname[20];
   char lastname[20];
};
struct data
{
      char type;
      union u
      {
        char company[40];
        struct person p;
      }details;
   }s;
```

It is shown diagrammatically as follows.



1.2 Accessing Members of the Union

Members of a union can be accessed using the dot operator, i.e.,

```
union_variable.member
```

Apr. 2018 – 4M
Define union. Explain how to access it's member. Give example.

1

For example, consider the union declared earlier:

```
union u
  {
    char s[5];
    int num;
  } u1;
```

The members of u1 are: u1.s and u1.num

Consider the second *example*.

```
struct data
{
char type;
    union u
      {
        char company[40];
        struct person p;
      }details;
}s;
```

In this case, the structure variable is s. The members of s are s.type and s.details. However, 'details' is a union variable. Its members are details.company and details.p. In this case, 'p' is a variable of the type 'person'. Hence, the members are: details.p.firstname and details.p.lastname.

Thus, the members are:

```
s.type
s.details.company
s.details.p.firstname
s.details.p.lastname
```

If we have a pointer to the union (similar to the pointer to a structure), the members are accessed using the → operator.

```
union_pointer→ member
```

Example

```
union u *ptr=&u1;        /* Initialize pointer to union */
```

The following expressions are valid.

```
u1.s              ptr→s
u1.num            ptr→num
```

1.3 Initialization

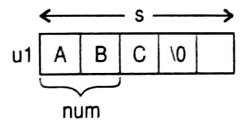
The initialization operation assigns a value to a variable when it is declared. In case of a union, since only one member of the union can be used at a time, only one member can be initialized. The initializer for the union is a value for the first member of the union, i.e., only the first member can be initialized.

Example

```
union u
{
  char s[5];
  int num;
};
```

Here, the first member is a string. Hence, only the string can be initialized.

```
union u u1 = {"ABC"};
```



1.4 Union within a Structure

Just as it is possible to have a structure within the other, the same can be done with unions. A union can be nested within a union or a structure.

This can be better understood with an *example*. Suppose we wish to store employee information viz., name, id and designation. If the designation is 'M' (for manager) we want to store the number of departments he manages and if his designation is 'W' (for worker), the department name should be stored.

The structure and union declarations will be:

i.

```
union info
{
  int no_of_depts;
  char deptname[20];
};
struct employee
{
  char name[20];
  int id;
  char desig;
  union info details;
} emp;
```

ii.

```
struct employee
{
  char name[20];
  int id;
  char desig;
   union info
   {
     int no_of_depts;
     char deptname[20];
   }details;
} emp;
```

The members of variable emp will be:

```
emp.name
emp.id
emp.desig
emp.details.no_of_depts(if emp.desig = 'M')
emp.details.deptname(if emp.desig = 'W')
```

We shall now write a program to accept cricket player information – Name, Player Type and Score. The score depends on Player type. If batsman- store batting average. If bowler store no. of wickets. If wicketkeeper - store no. of stumpings. Display the details of all batsmen, bowlers and wicketkeeper.

Union within structure

```
#include<stdio.h>
enum type { batsman=1, bowler=2, wicketkeeper=3};
struct player
{
  char name[20];
  enum type ptype;
  union
  {
    float batavg;
    int wickets;
    int stumpings;
```

```
  }info;
};
void accept(struct player p[])
{
  int i;
  for(i=0;i<11;i++)
  {
    printf("\nEnter the name :");
    gets(p[i].name);
    fflush(stdin);
    printf("\nEnter the player type(1-Batsman,2-Bowler,3-
            Wicketkeeper");
    scanf("%d",&p[i].ptype);
    switch(p[i].ptype)
    {
      case 1: printf("\nEnter the batting average :");
              scanf("%f",&p[i].info.batavg);
              break;
      case 2: printf("\nEnter the wickets taken :");
              scanf("%d",&p[i].info.wickets);
              break;
      case 3: printf("\nEnter the stumpings :");
              scanf("%d",&p[i].info.stumpings);
              break;
    }
    fflush(stdin);
  }
}
void displaysummary(struct player p[])
{
  int i;
  printf("\n\n ALL BATSMEN \n\n");
  for(i=0;i<11;i++)
  {
    if(p[i].ptype==1)
    {
      printf("%s\t",p[i].name);
      printf("%f\n",p[i].info.batavg);
    }
  }
  printf("\n\n ALL BOWLERS \n\n");
  for(i=0;i<11;i++)
  {
    if(p[i].ptype==2)
```

```
            {
              printf("%s\t",p[i].name);
              printf("%d\n",p[i].info.wickets);
            }
        }
        printf("\n\n WICKETKEEPER \n\n");
        for(i=0;i<11;i++)
        {
          if(p[i].ptype==3)
          {
              printf("%s\t",p[i].name);
              printf("%d\n",p[i].info.stumpings);
          }
        }
}
main()
{
 struct player p[11];
  accept(p);
  displaysummary(p);
}
```

1.5 Structure within a Union

Oct. 2017 – 4M
Define union and explain how union is used within a structure.

A union can have a structure as a member. This can be done in two ways:

1. Define the structure and variable within the union.
2. Define the structure outside and declare the structure variable within the union.

The structure and union declarations will be:

i.	ii.

`union union_name`
`{`
`  //members;`
`  struct structure_name`
`  {`
`    //members`
`  }variable;`
`};`
	`struct structure_name`
	`{`
	`    // members`
	`};`
	`union union_name`
	`{`
	`   //members`
	`   struct structure_name variable;`
	`};`

For example, consider the following. If we want to store **one** of the following information for person_identification:

i. PAN card number
ii. AADHAR number
iii. Reference name and contact number

Here, the PAN and AADHAR will be individual fields but the reference name and contact number will be a structure.

```
union person_id
{
  char PAN[11];
  char AADHAR[13];
  struct details
  {
     char name[20];
     char contact[20];
  }ref_person;
}p1;
```

Members of p1 will be either:

`p1.PAN or p1.AADHAR or (p1.ref_person.name and p1.ref_person.contact)`

2. Nested Union

One union can be declared within another union. Such a union is called a nested union.

Syntax

```
union outer
{
  // members of outer union
  union inner
  {
     // members of inner union
   }inner_union_variable;
}outer_union_variable;
```

To access the members of the union, the syntax is:

```
        Outer_variable.inner_variable.member
```

Example

```
    union outer
    {
      char a;
      int b;
      union inner
      {
        char c;
        int d;
      } u_inner;
    }u_outer;
```

Here, u_outer is a variable of the outer union. This union has three members:

```
    u_outer.a;
    u_outer.b;
    u_outer.u_inner
```

However, `u_outer.u_inner` is a union which has two members. Hence, either `u_outer.u_inner.c` or `u_outer.u_inner.d` can be used.

3. Difference between Structure and Union

1 Apr. 2018 – 4M Write a difference between structure and union with example.

No.	Structure	Union
1.	All members of a structure occupy separate memory locations.	All members of the union share the same memory location.
2.	All members of a structure can be accessed at the same time.	Only one member of a union can be accessed at a time.
3.	The size of a structure variable is the sum of sizes of all members of the structure.	The size of a union is the size of the largest member of the union.
4.	All members of the structure can be initialized.	Only the first member of the union can be initialized.
5.	Structure members start at different memory locations.	Union members start at the same location in memory.
6.	Declared using keyword 'struct'.	Declared using keyword 'union'.

4. Pointers and Unions

We have seen how pointers can be used with structures. In the same way, pointers can be used with union. The address of a union variable can be obtained using the & operator. This address can be assigned to a pointer variable. The pointer must be declared as a pointer to a union.

Syntax

```
union union_name * pointer;
```

Example

```
      union u
      {
        int a;
        char b;
      };
  union u *ptr;
```

The pointer is assigned address of a union variable. To access the members of the union variable using a pointer, we use the -> operator.

Syntax

```
pointer = &union-variable;

pointer->member
```

Example

```
    union u u1, *ptr;
    ptr=&u1;
    printf("Members are %d %c", ptr->a, ptr->b);
```

5. Enumerated Data Type

It is a user defined data type with values ranging over a finite set of identifiers called enumeration constants. The enumerated data type along with its set of identifiers can be created by the following declaration.

```
enum data_type_name{const1, const2.........};
```

Or

```
enum data_type_name{const1=value, const2=value, ........};
```

Each identifier in the enumerated data type is represented internally as an integer.

Example

```
enum color{red, blue, green} ;
```

In this *example,* color is a new data type. Red, blue and green are enumeration constants, which represent the integer values of 0,1 and 2 respectively. If required, different values can be assigned to the enumeration constants.

Example

```
enum daysofweek{Sun=1, Mon, Tue, Wed, Thu, Fri, Sat};
```

Here, Mon is assigned value 2, Tue is assigned 3, and so on. Each constant can be assigned a different value as shown.

Example

```
enum color{red=10, blue=20, green=30};
```

Creating Variables

A variable of the enumerated type can be created as follows:

```
type_name variable;
```

Example

```
daysofweek oneday;
```

Here, oneday is a variable which can be assigned the value of an enumeration constant.

```
oneday=Wed; or oneday=4;
```

6. Bit Fields

The size of a structure or union is calculated according to the size of its individual members. In the case of a structure, it is the sum of sizes of individual members. In C, we can specify size (in bits) of structure and union members.

Syntax

```
struct structure_name
{
  type member_name:no_of_bits;
  ...
};
```

The number of bits in the bit-field should be lesser than or equal to the size of the specified type.

Example:

```
struct my_structure
{
int a:8;
  int b:8;
  int c:16;
};
```

Here, the bit-field a has size 8 bits, b has size 8 bits and c has size 16 bits. The total size of the above structure will be 32 bits or 4 bytes.

Need

The purpose of bit-fields is to use memory efficiently. In many cases we know that the value of a field or group of fields will never exceed a limit or is within a small range. Here, bit-fields help in optimizing memory utilization.

Use

For example, consider the structure date having three data members: day, month, year.

```
struct date
{
  unsigned int day,month, year;
};
```

Here, the sizeof(struct date) will be sizeof(day)+sizeof(month)+sizeof(year) which will be 12 bytes if an int occupies 4 bytes of memory.

However, we know that day can only take values 1 to 31 so only 5 bits are enough to store day values. Similarly, month can take values 1 to 12 and 4 bits are enough to store month.

Example

```
struct date
{
    unsigned int day : 5;
    unsigned int month : 4;
    unsigned int year;
};
```

Here, the sizeof(struct date) will be 8 bytes. The members day and month will together occupy 4 bytes and year will occupy 4 bytes.

7. Multi-File Programs

In case of large, complex programs, the code can be separated into multiple files. Each file can have access to variables and functions declared and defined within the same file. Functions and global variables can be shared between multiple files using appropriate storage classes like extern and static. Each file can access global shared variables and functions.

How to Create Multi-File Programs

A multi-file program contains several source code files (.c) and related header (.h) files. The source code files are compiled separately (.o files) and then linked by the linker to create the executable code.

For example, if there are three source-code files: mainprogram.c file1.c file2.c,

Step 1: Compile each non-main source code separately using the –c option. The –c option suppresses call to the linker and generates the object code file (.o).

`cc -c file1.c` creates file1.o

`cc -c file2.c` creates file2.o

Step 2: Compile the main file and link all the object files.

```
cc mainprogram.c file1.o file2.o
```

This creates the file ./a.out which is the executable file for the program.

Advantages

1. A program can be built, modified and tested as a set of "modules", and later combined into the final running program.
2. It reduces the time to develop and recompile a large program. Large programs are often broken into several files so that when the system is rebuilt after a change, only the files which changed need to be recompiled.

3. Improves security since a file can be used as a unit for data-hiding purposes.
4. To create libraries, where the contents of the library is compiled once and stored, and various programs use its facilities.

Solved Programs

1. Consider the following structure

```
union a
{
  int i;
  char ch[2];
};
union a u;
 u.ch[0]=3;
 u.ch[1]=2;
printf("%d, %d, %d\n", u.ch[0], u.ch[1], u.i);
```

Solution

3, 2, 770

The total number of bytes allocated for u is 2 bytes as shown below.

	ch[0]	ch[1]
u	00000011	00000010
	2000	2020

When the variable i is displayed, both the bytes will be considered. Hence the value is 770 (512 + 256 + 2 = 770)

2. Find memory requirement for

```
union stud
{
  char name[20];
  long int reg_ id;
}
```

Solution

The storage allocated for an union is the storage required for the largest member of the union. In the given union, the largest member is name[20]. So the required memory is: 20 bytes (since each char occupies 1 byte).

Exercises

A. Predict the Output

1.
```
union
{
union
{ char a;
  char b;
}u_ab;
union
{ int j;
  int k ;
} abc;
float z;
} pqr;
printf("%d",sizeof(pqr));
```

2.
```
union test
{
      int x;
      char arr[8];
      int y;
};
void main()
{
      printf("%d", sizeof(union test));
}
```

3. What would be the size of the following union declaration?

```
union uTemp
{
   double a;
   int b[10];
   char c;
}u;
(Assuming size of double = 8, size of int = 4, size of char = 1)
```

B. Programming Exercises

1. Write a program to store information about 'n' employees. The details are:

   name, emp_id, designation(M-Manager, D-Director, W-worker, details (for director-years of experience, for manager-name of the department, for worker-his specializations viz, electrician, mechanic, draftsman, etc.)

Use a menu to display details of

i. All directors

ii. All managers

iii All workers

C. Review Questions

1. Define union.
2. The size of a union is the total size of all its members. True/False?
3. A structure or union can contain another structure or union as its member. True/False?
4. All members of a union can be initialized. True/False.
5. Altering one member of union alters the value of all other members. True/False.
6. How is a union declared? Can it be initialized? Explain.
7. What are the differences between a structure and a union? Illustrate with an example.
8. Explain nesting of unions. How can members of nested unions be accessed?
9. Differentiate between structure and union.
10. Explain the concept of bit-fields with an example.
11. Explain how multi-file programs are created and executed.
12. What are the advantages of multi-file programs?
13. Explain the use of enumerated data type with an example.
14. Custom values can be assigned to enumerated data type. State true/false?
15. It is possible to specify the size of individual members in a structure or union. Explain.

Questions Asked in Previous Exams

1 Mark

A. Choose correct option

1. Which of the following comments about union is false? **[Oct. 2017]**
   i. Union is a structure whose members share same memory area.
   ii. The compiler will keep track of what type of information is currently stored.
   iii. Only one of the members of union can be assigned a value at particular time.
   iv. Size allocated for union is the size of its member needing the maximum storage.

B. Answer the following questions

1. How is a union declared? **[Apr. 2018]**
2. Define union. **[Oct. 2017]**
3. "A union cannot be nested in a structure". State true/false. **[Oct. 2017]**

4 Mark

1. Trace the output and justify **[Apr. 2018]**

```
union exam
{
   int a,b;
};
union exame;
e.a = 5
   e.b = 7
printf("%d",e.a);
```

2. Define union. Explain how to access its member. Give example. **[Apr. 2018]**
3. Write a difference between structure and union with example. **[Apr. 2018]**
4. Define union and explain how union is used within a structure. **[Oct. 2017]**

VISION