UNIONS

Structure	Union
i. Access Members	
We can access all the members of structure at anytime.	Only one member of union can be accessed at anytime.
ii. Memory Allocation	
Memory is allocated for all variables.	Allocates memory for variable which variable require more memory.
iii. Initialization	
All members of structure can be initialized	Only the first member of a union can be initialized.
iv. Keyword	
'struct' keyword is used to declare structure.	'union' keyword is used to declare union.
v. Syntax	
`struct struct_name {element 1;element 2;element n; }struct_var_nm;`	`union union_name {element 1;element 2;element n; }union_var_nm;`

Definition:

•A union, is a collection of variables of different types, just like a structure. However, with unions, you can only store information in one field at any one time.

•Once a new value is assigned to a field, the existing data is wiped over with the new data.

•A union can also be viewed as a variable type that can contain many different variables (like a structure), but only actually holds one of them at a time (not like a structure).

•The size of a union is equal to the size of it's largest data member.

Usage:

- access individual bytes of larger type .
- variable format input records (coded records).
- sharing an area to save storage usage.
- unions not used nearly as much as structures .

Big-Endian:

means that the most significant byte of any multibyte data field is stored at the lowest memory address, which is also the address of the larger field.

Little-Endian:

means that the of any multibyte data field is stored at the lowest memory address, which is also the address of the larger field.

Example:

For example, consider the 32-bit number, 0xDEADBEEF. Following the Big-Endian convention, a computer will store it as follows:

Memory Location	Value
Base Address + 0	DE
Base Address + 1	AD
Base Address + 2	BE
Base Address + 3	EF

Figure 1. Big-Endian: The most significant byte is stored at the lowest byte address.

Example:

For example, consider the 32-bit number, 0xDEADBEEF. Following the Little-Endian convention, a computer will store it as follows:

Memory Location	Value
Base Address + 0	EF
Base Address + 1	BE
Base Address + 2	AD
Base Address + 3	DE

Figure 2. Little-endian: Least significant byte is stored at the lowest byte address.

BIT FIELDS

BIT FIELDS

- C permits to use small bit fields to hold data items and thereby to pack several data items in a word of memory
- Bit field is a set of adjacent bits whose size can be from 1 to 32 bit in length.
- A word can therefore be divided into a number of bit fields
- The name and size of bit fields are defined using a structure.
- The following restrictions apply to bit fields. You cannot:

1. Define an array of bit fields.
2. Take the address of a bit field.
3. Have a pointer to a bit field.

```
struct personal
{
unsigned gender:      1;
unsigned age:         7;
unsigned m_status: 1;
unsigned children:   4;
} emp;
```

Bit field	bit length	range of values
Gender	1	0 or 1
Age	7	0 to 127
M_status	1	0 or 1
Children	4	0 to 15

```
int main()
{
struct person_details
{
        unsigned gen:1;
        unsigned age:7;
        unsigned child:2;
        unsigned m_status:1;
}pd;
int i;
printf("\n enter ur gen(0-Female 1-
Female)");
scanf("%d",&i);
pd.gen=i;
printf("\n enter ur age");
scanf("%d",&i);
pd.age=i;
printf("\n enter how many childs ");
scanf("%d",&i);
pd.child=i;
printf("\n enter ur m_status (0-married 1-
Unmarried)");
scanf("%d",&i);
pd.m_status=i;
printf("\n person details are as follows");
printf("\n Gender ::");
if(pd.gen)
printf(" MALE");
else
printf(" FEMALE");
printf("\n Age :: %d",pd.age);
printf("\n Childrens :: %d",pd.child);
printf("\n Marriage status ::");
if(pd.m_status)
printf(" MARRIED");
Else
printf(" UNMARRIED");
printf("\n");
return 0;
}
```

Out put:

enter ur gen(0-Female 1-Male)1

enter ur age26

enter how many childs 0

enter ur m_status (0-married 1-unmarried)1

person details are as follows
Gender :: MALE
Age :: 26
Children's :: 0
Marriage status :: MARRIED

ENUMERATIONS

- Enumerated data type is an user defined data type.
- It is used to declare a variables that can have one of the values enclosed within the braces (known as enumeration constants).
- Compiler automatically assigns integer digits beginning with 0 to all enum constants.

```
enum Days{
    Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Sat
    urday };
```

Sunday=0, Monday=1 …… Saturday=6.

```
int main()
{
        enum bool_type
        {
                true=1,false=0
        }boolen;
        boolen=true;
        if(true)
        printf("\n 1st true block");
        else
        printf("\n 1st false block");
        if(boolen)
        printf("\n 2nd true block");
        else
        printf("\n 2nd false block");
        return 0;
}
```

PREPROCESSOR STATEMENTS

Preprocessor statements will executed before compilation.

These directives can be divided into 3 types

1. Macro substitution directives.
2. File inclusion directives.
3. Compiler control directives.

1. Macro substitution directives:

#define MAX 8

MAX → is called macro templet.
8 → is called macro expansion.

If in our program preprocessor findes any macro templet, it substitutes expansion in the place of templet.

Sample program:

```
#include<stdio.h>
#define MAX 8
main()
{
        int i;
        for(i=0;i<MAX;i++)
        printf("%d\t",i);
}
```

Output:

0 1 2 3 4 5 6 7

2. File Inclusion Directives

#include <stdio.h>
Searches in library directory for function proto types.

#include " stdio.h"
At first linker searches in current working directory if proto type declaration is not found, then it searches in library directory.

3 . Compiler control directives:

```
#include<stdio.h>
#ifndef MAX
#define MAX 8
#endif
main(){
        int i;
        for(i=0;i<MAX;printf("%d\t",i),i++);
}
```

In the above example if MAX macro is not defined in the program we are defining the MAX maco.

```
#include<stdio.h>
#define ISM
main()
{
#ifdef ISM
{
        printf("\n ifdef");
}
#else
{
        printf("\n else");
}
#endif
}
```

Apart from the above compiler control directives ANSI C added some more directives like.

```
#if expression
{
-----------
}
#elif expression
{
-----------
}
#endif
```

Macros Vs Enums:

- Macros scope is global.
- We can't restict the scope of macros to local.
- If we declare enum as a global that scope is global.
- If declare enum as local that scope is local.

TYPEDEFS

- The typedef keyword allows the programmer to create new names for types such as predefined or user defined.
- Typedefs can be used both to provide more clarity to your code and to make it easier to make changes to the underlying data types that you use.

- **Synatx:**

  typedef existed_data_type_name
  new_data_type_name;

```
main() {
        typedef int integer;
        integer i;
        printf("\n enter integer value::");
        scanf("%d",&i);
        printf("\n entered integer value is::%d",i);
        printf("\n");
}
```

Output:

```
 enter integer value::23
 entered integer value is::23
```