STRUCTURES AND UNIONS

- Introduction
- Structure definition
- Declaring structure variables
- Accessing structure members
- Structure initialization
- Copying and comparing structure variables
- Structures within Structures

Introduction

- Array- A user defined type, stores the data element of same data type.
- Structure-user-defined data type that can hold a collection of elements of different fundamental data types.
- Structure Definition: A structure is a collection of variables under a single name. These variables can be of different types, and each has a name which is used to select it from the structure.
- For example: A student has various data such as- Name, USN, Courses taken, Marks etc, the values of which cannot be stored in an array.

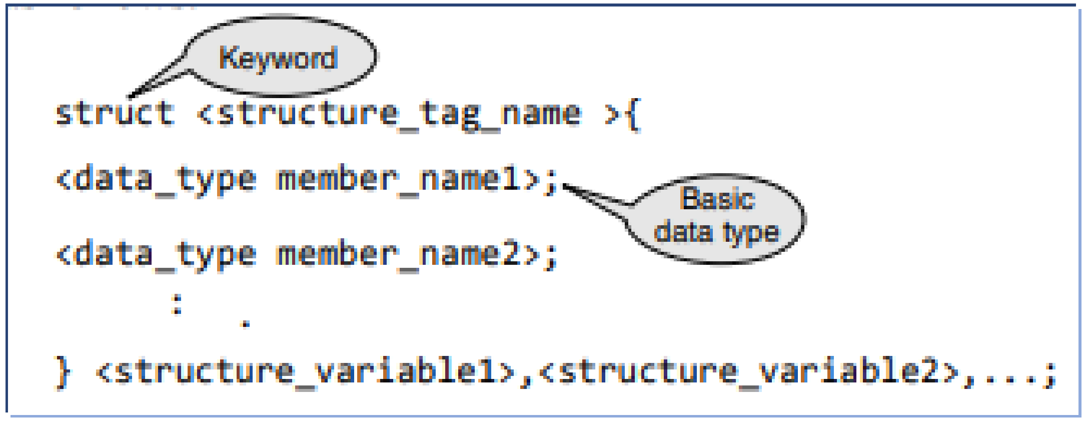
Need for a structure

- Structure helps to construct a complex data type which is more meaningful.
- Using structure, we can store different data type.
- It is also called the heterogeneous data type.

Need to store	Example	Data type
Roll number	624128	integer
Name	"Alice"	string
Age	20	integer
Date of birth	"1-1-2011"	string
Address	"1,New street,etc"	string
Fine	100.50	float

Declaring structures and structure variables

- A structure is declared by using the keyword struct followed by an optional structure tag followed by the body of the structure.
- The variables or members of the structure are declared within the body.
- The general format is:



```
struct <structure_tag_name >{
<data_type member_name1>;
<data_type member_name2>;
     :    .
} <structure_variable1>,<structure_variable2>,...;
```

Keyword
Basic data type

```
struct  student{
char name[20];
char usn[10];
int courses;
float  marks1, marks2,marks3;
} S1,S2,S3;
```

Declaring structures and structure variables

- The structure_tag_name is the name of the structure.
- The structure_variables are the list of variable names separated by commas.
- Each of these structure_variable names is a structure of type structure_tag_name.
- The structure_variable is also known as an instance variable of the structure.
- Each member_name declared within the braces is called a member or structure element.

Declaring structures and structure variables

- Like all data types, structures must be declared and defined.
- There are two different ways to declare and/or define a structure.

1. Tagged declaration
2. Typedef declaration

Declaring structures and structure variables

- Tagged declaration starts with the keyword **“struct”** followed by the tag name (structure name).

```
struct tag_name
{
data-type var-name1;
data-type var-name2;
:
data-type var-nameN;
};
```

```
struct product
{
int pid;
char name[20];
int qnt;
float price;
};
```

Declaring structures and structure variables

- Typedef declaration differs from tagged declaration in two ways:
- Keyword “typedef” is placed at the beginning of the declaration.
- It requires identifier at the end of structure block (“}”) and before the semicolon (;).

```
typedef  struct
{
data-type var-name1;
data-type var-name2;
:
data-type var-nameN;
}identifier;
```

```
typedef struct
{
int pid;
char name[20];
int qnt;
float price;
} product;
```

Declaring structures and structure variables

- Once structure is declared we can create variables of structure type.

- Structure variable can be declared either globally or locally.

Global declaration of structure variable:

•When we declare structure variable outside the *main ()* function then it becomes global and it can be accessed from anywhere within the program.

•Structure variable can be declared using the following syntax:

struct <*struct_name*> var-name;

Declaring structures and structure variables

- Once structure is declared we can create variables of structure type.

- Structure variable can be declared either globally or locally.

Global declaration of structure variable:

•When we declare structure variable outside the *main ()* function then it becomes global and it can be accessed from anywhere within the program.

•Structure variable can be declared using the following syntax:

struct <*struct_name*> var-name;

Declaring structures and structure variables

Global declaration of structure variable:

```
struct product
{
int pid;
char name[20];
int qnt;
float price;
};
struct product p1,p2;  // global declaration
void main()
{
// main body
}
```

```
struct product
{
int pid;
char name[20];
int qnt;
float price;
} p1,p2;
void main()
{
// main body
}
```

```
typedef struct {
int pid;
char name[20];
int qnt;
float price;
} product;
product p1,p2;  //
global declaration
```

Declaring structures and structure variables

Local declaration of structure variable:

- Structure variable can be declared as local by declaring it inside the main () function as follow:

Declaring structures and structure variables

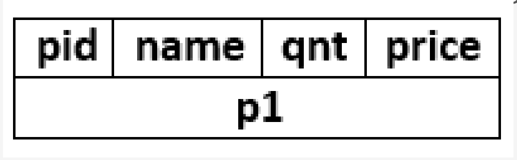
Local declaration of structure variable:

```
struct product
{
int pid;
char name[20];
int qnt;
float price;
};
void main()
{
// Local declaration
struct product p1,p2;
}
```

```
typedef struct
{
int pid;
char name[20];
int qnt;
float price;
} product;
void main()
{
// Local declaration
   product p1,p2;
}
```

Memory requirements of structures and structure variables

•Memory representation for the structure variable can be given as shown into the below figure.



•Each structure member is allocated separate memory area.

•Total memory required by the structure variable can be calculated as follow:

pid (int) = 2 bytes + name (char) 20 bytes + qnt (int) 2 bytes + price(float) 4 bytes = **28 bytes**

- Declaring structures and structure variables
  - The typedef keyword allows the programmer to create a new data type name for an existing data type.
  - No new data type is produced but an alternate name is given to a known data type.
  - The general form of the declaration statement using the typedef keyword is given as follows.
  - typedef <existing data type> <new data type ,….>;
  - The following examples show the use of typedef.
  - typedef int id_number;
  - typedef float weight;
  - typedef char lower_case;

Accessing structure members

- We can access individual structure members using two operators:
- The structure member operator (.) also called as “direct selection operator”, “dot” or “period” operator.
- The structure pointer operator (->) also called as “arrow operator”.
- To refer to a member in a structure we need to refer to both the structure variable and structure member respectively.

Syntax:

Struct_Var. member name

Initializing structure members

- Structure Members can be initialized statically or dynamically

Static:

```
struct product
{
   int pid;
   char name[20];
   int qnt;
   float price;
};
```

```
void main()
{
   struct product p1,p2;

   // individual  member initialization.

   p1.pid = 101 ;
   strcpy( p1.name , “Laptop” );
   p1.qnty = 10 ;
   p1.price = 35000.00 ;

   // group initialization  method

   p2 = {102 , “Mobile” , 150 , 12000.00 } ;
}
```

Initializing structure members

Dynamic

```
struct product
{
   int pid;
   char name[20];
   int qnt;
   float price;
};
```

```
void main()
{
   struct product p1;
   // member initialization using scanf () function.

   printf("\n Enter Prduct ID, Name, QNT and Price
:" ) ;
   scanf("%d %s %d %f", &p1.pid, &p1.name,
&p1.qnty, &p1.price ) ;
   printf("%d %s %d %f", p1.pid, p1.name, p1.qnty,
p1.price ) ;

}
```

- Declaring structures and structure variables

```
typedef struct point{
int x;
int y; } Dot;
Dot left,right;
```

- When typedef is used to name a structure, the structure tag name is not necessary.

```
typedef struct     /* no structure tag name used */
{
float real;
 float imaginary;
} complex; /* means complex number */
complex u,v;
```

- Declaring structures and structure variables

```
struct
{ char name[100];
char address[200];
int year_of_birth;
int month_of_birth;
int day_of_birth; }monish, venkat, naresh;
```

```
struct personal_data
{ char name[100];
char address[200];
int year_of_birth;
int month_of_birth;
int day_of_birth; };
struct personal_data monish, venkat, naresh;
```

```
typedef struct personal_data
{ char name[100];
char address[200];
int year_of_birth;
int month_of_birth;
int day_of_birth; }PD;
PD monish, venkat, naresh;
```

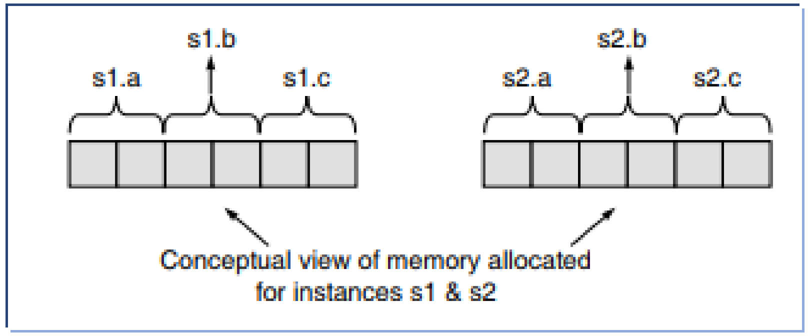
• Accessing the Members of a Structure

- The members of a structure can be accessed using the '.', dot operator.
- The general form of the statement for accessing a member of a structure is as follows. < structure_variable >.< member_name >;

```
struct myStruct
 {
  int a;
  int b;
  int c;
 } s1, s2;
```

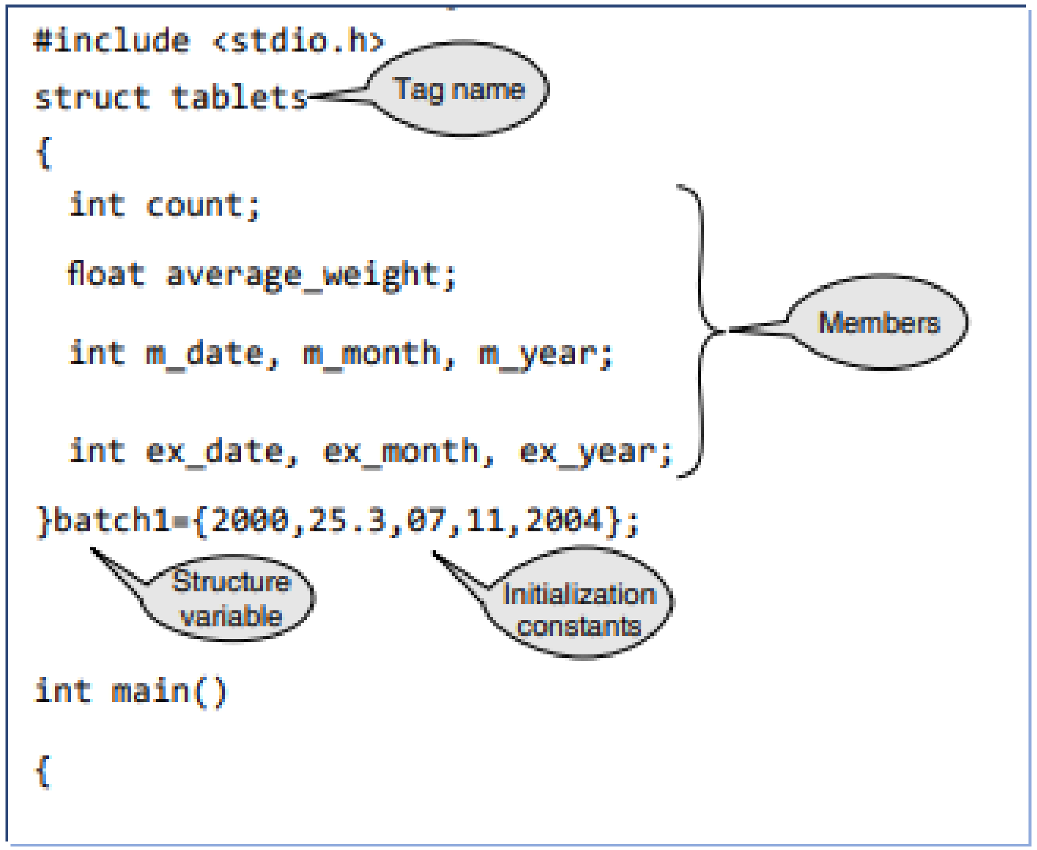
the first member can be accessed by the construct

```
s1.a
```



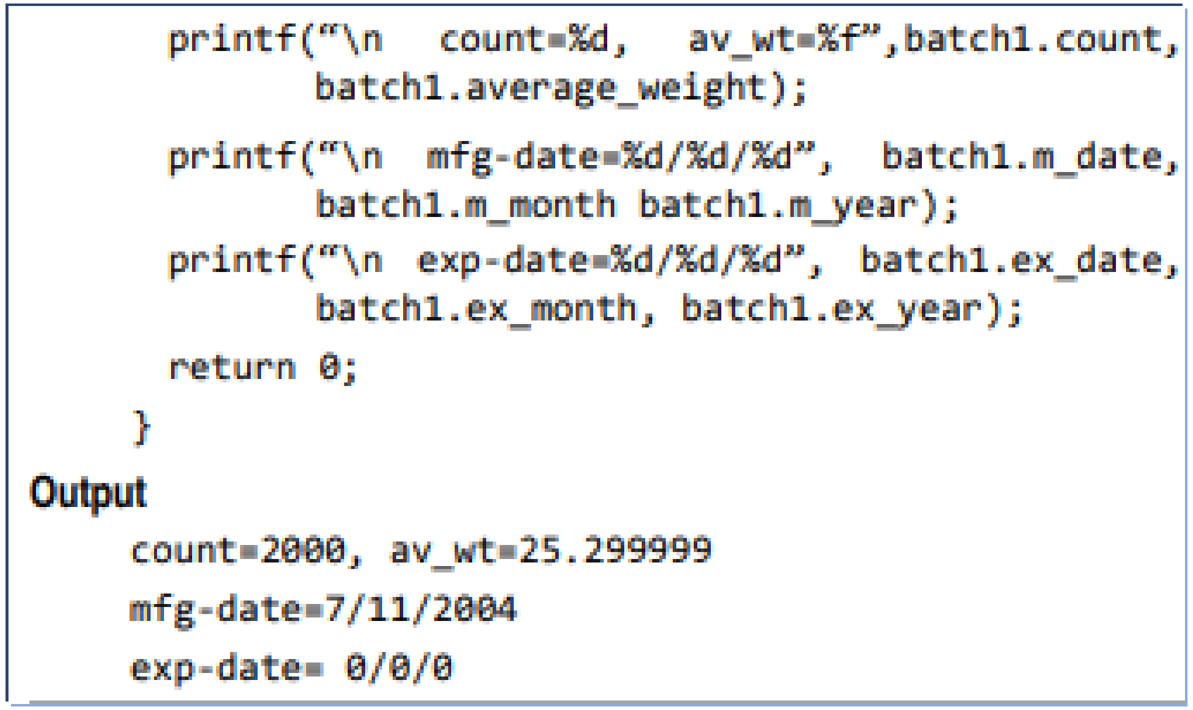
- To print this value assigned to the member on the screen, the following code is written. printf("%d", s2.b);

• Accessing the Members of a Structure



```
#include <stdio.h>
struct tablets
{
  int count;
  float average_weight;
  int m_date, m_month, m_year;
  int ex_date, ex_month, ex_year;
}batch1={2000,25.3,07,11,2004};

int main()
{
```



```
        printf("\n  count=%d,  av_wt=%f",batch1.count,
                batch1.average_weight);
        printf("\n  mfg-date=%d/%d/%d",  batch1.m_date,
                batch1.m_month batch1.m_year);
        printf("\n  exp-date=%d/%d/%d",  batch1.ex_date,
                batch1.ex_month, batch1.ex_year);
        return 0;
      }
```

Output

```
      count=2000, av_wt=25.299999
      mfg-date=7/11/2004
      exp-date= 0/0/0
```

Structures within Structures

- A structure can be placed within another structure. A structure may contain another structure as its member.
- Declare two structures separately and then group them in a high level structure.
- Ex: for declaration
-

```
typedef struct
{
char first_name[20];
char last_name[20];
}Name;
```

```
typedef struct
{
int dd;
int mm;
int yy;
}DATE;
```

```
typedef struct
{
int r_no;
NAME s_name;
DATE DOB
char course[10];
float fees;
};
```

POINTERS

- Introduction.
- Benefits of using pointers.
- Declaration and Initialization of pointers.
- Obtaining a value of a variable.
- Type casting a pointer.
- Arithmetic operations using pointers.
- Pointers and arrays.
- Pointers and strings.
- Pointers and functions.

POINTERS

Introduction.

- A pointer provides a way of accessing a variable without referring to the variable directly.
- The mechanism used for this is the address of the variable.
- A program statement can refer to a variable indirectly using the address of the variable.
- A pointer variable holds the memory address of another variable.
- They are called pointers for the simple reason that by storing an address, they 'point' to a particular point in memory.

POINTERS

Introduction.

- A pointer points to that variable by holding a copy of its address.
- Because a pointer holds an address rather than a value, it has two parts.
    - The pointer itself holds the address.
    - The address points to a value.

POINTERS

Benefits of Using Pointers

- Pointers can be used to:
- call by address, thereby facilitating the changes made to a variable in the called function to become permanently available in the function from where the function is called.
- return more than one value from a function indirectly
- pass arrays and strings more conveniently from one function to another
- manipulate arrays more easily by moving pointers to them (or to parts of them) instead of moving the arrays themselves

POINTERS

Benefits of Using Pointers

- create complex data structures, such as linked lists and binary trees, where one data structure must contain references to other data structures.
- communicate information about memory, as in the function malloc() which returns the location of free memory by using a pointer.
- compile faster, more efficient code than other derived data types such as arrays.

NOTE:

In C, there is an additional restriction on pointers—they are not allowed to store any memory address, but they can only store addresses of variables of a given type

Declaration and Initialization of Pointers

- A pointer has to be declared.
- It will have a value, a scope, a lifetime, a name; and it will occupy a certain number of memory locations.
- The pointer operator available in C is ‘*’, called value at address operator.
- It returns the value stored at a particular address.
- The value at address operator is also called indirection operator.
- A pointer variable is declared by preceding its name with an asterisk.
- The syntax for declaring a pointer variable is:

```
datatype * pointer_variable;
```

Declaration and Initialization of Pointers

For Example:

```
char * ptr;
```

This declaration is evaluated as: ptr is a pointer to char type data.

Table meaning of some pointer type variable declarations

Declaration	What it means
int p	P is an integer
int *p	P is a pointer to an integer
char p	P is a character
char *p	P is a pointer to a character
long p	P is a long integer
long *p	P is a pointer to a long integer

Declaration and Initialization of Pointers

WHAT IS THE OUTPUT OF FOLLOWING CODE?

```
int main(){
int *p;
float *q;
double *r;
printf("\n the size of integer pointer is %d", sizeof(p));
printf("\n the size of float pointer is %d", sizeof(q));
 printf("\n the size of double pointer is %d",sizeof(r));
 printf("\n the size of character pointer is %d", sizeof(char *));
return 0;}
```

Declaration and Initialization of Pointers

WHAT IS THE OUTPUT OF FOLLOWING CODE?

```
int main(){
int *p;
float *q;
double *r;
printf("\n the size of integer pointer is %d", sizeof(p));
printf("\n the size of float pointer is %d", sizeof(q));
 printf("\n the size of double pointer is %d",sizeof(r));
 printf("\n the size of character pointer is %d", sizeof(char *));
return 0;}
```

Declaration and Initialization of Pointers

why pointers should have data types

- C has data types of different size, i.e., objects of different types will have different memory requirements.
- It supports uniformity of arithmetic operations across different (pointer) types.

Where is a pointer stored?

- A pointer is like any other variable in the sense that it requires storage space somewhere in the computer's memory, but it is not like most variables because it contains no data, only an address.
- Since it is an address, it actually contains a number referring to some memory location

Initializing Pointers

- Consider the following example:

```
#include int main()
 {
 int *p; /* a pointer to an integer */
printf("%d\n",*p);
return 0;
}
```

Note: A pointer should be initialized with another variable's memory address, with 0, or with the keyword NULL prior to its use; otherwise the result may be a compiler error or a run-time error.

Initializing Pointers

```
#include <stdio.h>
int main()
{
int i = 5;
int *ptr = &i;
printf(“\nThe address of i using &num is %p”, &i);
printf(“\nThe address of i using Ptr is %p”, ptr);
return 0;
}
```

Note: Type of the Pointer and the data type of the variable must be same.

Printing pointer value

```
#include <stdio.h>

int main()

{

int i = 5;

int *ptr = &i;

printf(“\nThe address of i using &num is %p”, &i);

printf(“\nThe address of i using Ptr is %p”, ptr);

return 0;

}
```

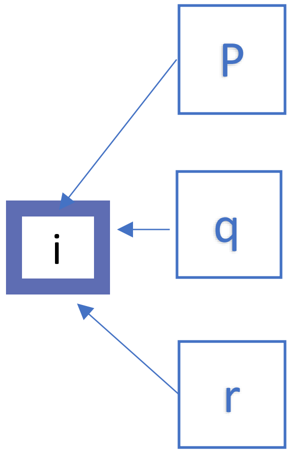
Examples:

```
int a=3, *ip;
 float *p;
char ch='A';
p=&a;
ip=&ch;
```

```
 int main(void)
{
int a=10, *p;
 p=&a;
printf("\n p = %p", p);
return 0;
}
```

Assigning constant to a pointer

```
int i=3;
 int *p, *q, *r;
p = &i;
q = &i;
r = p;
```



Assigning constant to a pointer

```
int *pi;
 pi= (int*)1000;
*pi = 5;
```

REMEMBER:

- *In C, pointers are not allowed to store any arbitrary memory address, but they can only store addresses of variables of a given type*

Examples: USING INDIRECTION OPERATOR and DEREFERENCING

```
#include int main() {
 int num = 5;
int *iPtr = &num;
printf(“\n The value of num is %d”, num);
num = 10;
printf(“\n The value of num after num = 10 is\ %d”, num);
 *iPtr = 15;
printf(“\n The value of num after *iPtr = 15 is\ %d”, num);
return 0; }
```

Examples: USING INDIRECTION OPERATOR and DEREFERENCING

```
main(){

int i=5;

int *p;

p = &i;

printf("\nValue of i = %d", i);

printf("\nValue of * (&i) = %d", *(&i));

return 0;

}
```

Note:

Printing the value of *(&i) is same as printing the value of i.

- always implies value at address.
- *(&i) is identical to i.

Examples: USING INDIRECTION OPERATOR and DEREFERENCING

Remember:

- The placement of the indirection operator before a pointer is said to dereference the pointer. (* ptr)
- The value of a dereferenced pointer is not an address, but rather the value at that address— that is, the value of the variable that the pointer points to.

```
int num = 5;
int *iPtr = &num;
printf("\n The value of num is %d", num);
num = 10;
            *iPtr = 15;
printf("\n The value of num after *iPtr = 15 is\
%d", num);
```

THE TWO OPERATORS- & and *

- Address of operator (&): It is used as a variable prefix and can be translated as ‘address of’.
  - &variable can be read as address of variable.
- Dereference operator (*): It can be translated by value pointed by or ‘value at address’.
  - *ptr can be read as ‘value pointed by ptr’.
  - It indicates that what has to be evaluated is the content pointed by the expression considered as an address.