Chapter 9 STRUCTURES

- A structure is a collection of variables, possibly of different types; grouped together under a single name for convenient handling.

  Or is a complex data type declaration that defines a physically grouped list of variables to be placed under one name in a block of memory, allowing the different variables to be accessed via a single pointer.
- **Advantages**
- Structures help to organize complicated data, particularly in large programs,
- permits a group related variables to be treated as a unit instead of separate entities.

8.1 Syntax:

1st method (single structure)

```
Struct struct name
{
Members;
}
struct type;
```

Example: 1

```
struct classroom
{
Char reg[16];
Char names[30];
Int age;
}
student;
```

Syntax:

2nd method (single structure)

```
Struct struct name
{
Members;
} ;
Struct name struct type;
```

Example: 1

```
struct classroom
{
Char reg[16];
Char names[30];
Int age;
};
classroom;
```

8.2 Initialisation

To initialise a structure we have first to create instance of structure name.

```
St.reg=“BIT/0125/11/E”;
St.name=“Gakuba John”;
St.age=“28”;
```

8.3 Assign using scanf()

```
Scanf(“%s”,&st.reg);
Scanf(“%s”,&st.names);
scanf(“%d”,&st.age);
```

8.4 To display the content of structure

```
Printf(“%s”,st.reg);
```

Example 2

Let's create a data structure suitable for storing the date.

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

Other data structures may be defined as consisting of the same composition as the date structure.

```
struct  date   todays_date;
```

defines a variable called todays_date to be of the same data type as that of the newly defined data type struct date.

8.5 Assigning values to structure elements

To assign todays_date to the individual elements of the structure todays_date, the statement

```
todays_date.day = 31;
todays_date.month = 8;
todays_date.year = 2006;
```

Note the use of the . element to reference the individual elements within todays_date.

UNIVERSITY OF KIGALI

Exercise 1: /* Program to illustrate a structure */

```
#include<stdio.h>
struct date          /* Global definition of type date*/
{
int day;
int month;
int year;
};
main()
{
struct date today;
today.day=31;
today.month=8;
today.year=2006;
printf("Today's date is: %d/%d/%d\n",today.day, today.month, today.year);
getchar();
}
```

8.6 Array of structures

Syntax:

```
Struct struct name
{
Members
} struct object[size];
```

Example:

```
Struct classroom
{
Char reg[16];
Char name[30];
Int age;
}student[100];
```

Exercise 2: /* Program to initialise student marks */

```
#include<stdio.h>
#include<conio.h>
struct classroom
{
char reg[16];
char name[30];
int age;
}st;
void main()
{
clrscr();
printf("Record student information\n");                    //student st=new
    student();
printf("Enter reg number:");
scanf("%s",&st.reg);
printf("\nEnter student name:");
scanf("%s",&st.name);
printf("\nEnter age:");
scanf("%d",&st.age);
clrscr();
printf("\nStudent has the following information");
printf("\n %s",st.reg);
printf("\t %s",st.name);
```

Exercise 3: /* Program to initialise employee details */

C

Exercises

1) Make a program which print out names, pages and price of three different books.
2) Write a c program to display marks of three subjects for 4 students.

Book by initialization

```
#include <stdio.h>
main()
{
struct book1
{
char book[30];
int pages;
float price;
};
struct book1 bk1={"C++",300,278.5},bk2={"Java",300,700};
//clrscr();
printf("\n Book name: %s",bk1.book);
printf("\n No of pages:%d",bk1.pages);
printf("\n Book price: %f",bk1.price);
printf("\n\n Book name: %s",bk2.book);
printf("\n No of pages:%d",bk2.pages);
printf("\n Book price: %f",bk2.price);
//system("PAUSE");
getchar();
```

UNIVERSITY OF KIGALI

Book

```
#include<stdio.h>
struct book
{
char name[25];
float price;
int page;
};
main()
{
struct book  b1, b2,b3;
printf("Enter the name, price(in dollars) and the number of pages:\n");
scanf("%s %f %d",&b1.name,&b1.price,&b1.page);
scanf("%s %f %d",&b2.name,&b2.price,&b2.page);
scanf("%s %f %d",&b3.name,&b3.price,&b3.page);
printf("Name=%s\t, Price=%.2f\t,  Page= %d\n",b1.name,b1.price,b1.page);
printf("Name=%s\t, Price=%.2f\t,  Page=%d\n",b2.name,b2.price,b2.page);
printf("Name=%s\t, Price=%.2f\t,  Page=%d\n",b3.name,b3.price,b3.page);
getchar();
}
```

UNIVERSITY Of KIGALI

Chapter 8 Pointer

A pointer is a variable that **points** to another variable, this means that a pointer used to store address memory of another variable.

A pointer is a variable which contains the address in memory of another variable. A pointer "points to" that variable by holding a copy of its address.

Pointers use 2 operators (*, &) to access values and pointed memory address.

Because a pointer holds an address rather than a value, it has 2 parts.
The pointer itself holds the address and that address points to a value.
There is the pointer and the value pointed to.

Pointer are a fundamental part of C. If you cannot use pointers properly then you have basically lost all the power and flexibility that C allows.

The secret of C programming is in its use of pointers, because It is the only way to express some computations by producing compact and efficient code, by providing a very powerful tool.

C program uses Pointers explicitly with: Arrays, Structures and **Functions**

Declaring Pointers

A pointer is a variable, which has both the left and right values are addresses. The left value of a pointer is used to refer to the pointer itself, whereas the right value of a pointer, which is the content of the pointer, is the address
of another variable.

Pointer Syntax

The pointer declaration looks like this:

<data_type> *<pointer name>;

For example, you could declare a pointer that stores the address of an integer with the following syntax:

int *points_to_integer; or **int* ptr;**

the *. This is the key to declaring a pointer

If you declare multiple pointers on the same line, you must precede each of them with an asterisk:

```
int *pointer1, nonpointer1;          /* one pointer, one regular int
   */
int *pointer1, *pointer2;            /* two pointers */
```

The following shows different types of pointers:

```
char *ptr_c;                         /* declare a pointer to a character */
int  *ptr_int;                       /* declare a pointer to an integer */
float *ptr_flt;                      /* declare a pointer to a floating-point
   */
```

Retrieving an Address

To get the memory address of a variable (its location in memory), put the & sign in front of the variable name. This makes it give its address. This is called the address-of operator, because it returns the memory address.

Demonstration of pointers in program

```
Int A;
Int * ptr;
A=20
Ptr=&A
Printf(“The value of A=%d”,A);
Printf(“The value of A=%d”,*ptr);
Printf(“%d address pointed %u”,ptr);
Printf(“%d address of A %u”,&A);
```

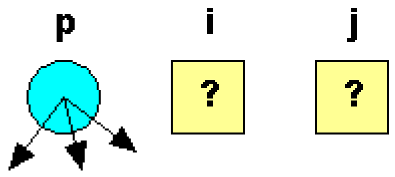
UNIVERSITY Of KIGALI

Example 1

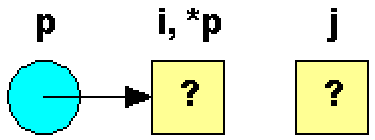
- #include <stdio.h>
  main()
  {
   int i,j;
  int *p;   /* a pointer to an integer */
   p = &i;   /***&** is called the **address operator***
                                  means, "Assign to **p** the address
  of **i**."/
  *p=5;
    j=i;
   printf("%d %d %d\n", i, j, *p);
   return 0;
  }

UNIVERSITY Of KIGALI

After **i, j** and **p** are declared, they look like this

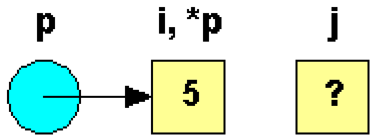


After the line **p = &I;**, **p** is initialized and it points to **i**, like this:



p is the location holding the address, while ***p** is the location pointed to by that address. Therefore ***p=5** means that the location pointed to by **p** should be set to 5,

like this:



Because the location ***p** is also **i**, **i** also takes on the value 5. Consequently, **j=i;** sets **j** to 5, and the **printf** statement produces **5 5 5**.

To define pointers requires 3 things:

Data type of the pointer: int *ptr

Pointer declaration: int * is a pointer data type

Name of the pointer variable: ptr is the name of the

Pointer.

Advantages of pointers

- provides functions which can hold modify their calling arguments
- supports dynamic allocation routines
- improves the efficiency of certain routines

UNIVERSITY Of KIGALI

Example 1: Array and pointer arithmetic

```
#include <stdio.h>
main()
{
int arr[3]={1,2,3};
int *ptr;
ptr=arr;
printf("address: %p - array value:%d\n",ptr,*ptr);
ptr++;
printf("address: %p - array value:%d \n",ptr,*ptr);


}
```

Example 2: Declaring and assigning values to pointers

```
#include <stdio.h>
 main()
 {
char  c, *ptr_c;
 int  x, *ptr_x;
 float y, *ptr_y;
 c = 'A';
 x = 7;
 y = 123.45;
printf("c: address=0x%p, content=%c\n", &c, c);
printf("x: address=0x%p, content=%d\n", &x, x);
printf("y: address=0x%p, content=%5.2f\n", &y, y);
ptr_c = &c;
printf("ptr_c: address=0x%p, content=0x%p\n", &ptr_c, ptr_c);
printf("*ptr_c => %c\n", *ptr_c);
ptr_x = &x;
printf("ptr_x: address=0x%p, content=0x%p\n", &ptr_x, ptr_x);
printf("*ptr_x => %d\n", *ptr_x);
ptr_y = &y;
printf("ptr_y: address=0x%p, content=0x%p\n", &ptr_y, ptr_y);
printf("*ptr_y => %5.2f\n", *ptr_y);
```

OUTPUT

```
c: address=0x1B38, content=A
x: address=0x1B36, content=7
y: address=0x1B32, content=123.45
ptr_c: address=0x1B30, content=0x1B38
*ptr_c => A
ptr_x: address=0x1B2E, content=0x1B36
*ptr_x => 7
ptr_y: address=0x1B2C, content=0x1B32
*ptr_y => 123.45
```

UNIVERSITY Of KIGALI

Example 3: Program to add two numbers using call by reference

```
#include<stdio.h>
#include<conio.h>
int add(int*,int*);
void main()
{
int A,B,Sum;
clrscr();
printf("Enter two numbers:");
scanf("%d %d",&A,&B);
clrscr();
printf("Values of A and B before access their memory %d and %d",A,B);
Sum=add(&A,&B);
printf("The sum is %d"Sum);
printf("The values of A,B after accessing their memory %d and %d",A,B);
getch();
}
```

UNIVERSITY Of KIGALI

Exercise

Make a program to display an initialized 10 integers and the address location :

The program should contain an array , loop and pointer.

```
#include<stdio.h>
int main ()
{
 int arr[]={1,2,3,4,5,6,7,8,9,10};
 int *ptr;
ptr=arr;
int i;
for (i=0;i<10;i++){
printf("address %p \t value %d \n",&arr[i], *ptr);
ptr++;}
getchar();
return 0;
}
```

UNIVERSITY of KIGALI

A program to display address of elements before 0

```
#include <stdio.h>
int array[] = {4, 5, 8, 9, 8, 1, 0, 1, 9, 3};
int *array_ptr;
int main()
{
 array_ptr = array;
  while ((*array_ptr) != 0)
  ++array_ptr;
  printf("Number of elements before zero %d\n",
array_ptr - array);
system("PAUSE");
return (0);
```

UNIVERSITY Of KIGALI

```
int  count = 10, *temp, sum = 0;

        temp = &count;
        *temp = 20;
        temp = &sum;
        *temp = count;
        printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );

        count = 20, *temp = 20, sum = 20
```

UNIVERSITY Of KIGALI

Let us use this new knowledge to examine a couple of statements.

```
int var_x;
int* ptrX;
var_x = 6;
ptrX = &var_x;
*ptrX = 12;
printf("value of x : %d", var_x);
```

The first line causes the compiler to reserve a space in memory for a integer. The second line tells the compiler to reserve space to store a pointer. As you can notice the way you declare a pointer is simply to add a "*" asterick to the end of the datatype. A pointer is a storage location for an address. The third line should remind you of the *scanf* statements. The address "&" operator tells C to goto the place it stored **var_x**, and then give the address of the storage location to **ptrX**.

The fourth line is a bit more complex. An asterick * in front of a variable tells C to dereference the pointer, and go to memory. Then you can make assignments to variable stored at that location. Since **ptrX** points to **var_x**, line 4 is equivalent to this command: var_x = 12;

Now you may be wondering, why are pointers so complex, or I've heard the using pointers can cause problems. It can, and for those who aren't careful misuse of pointer can do a lot of damage. Suppose that we forget to type in line 3 ptrX = &var_x; when we entered the program. What would happen if we executed it, who knows? Without this line ptrX is never assigned an address. Basically it points to some random data anywhere in memory.
If you executed line 4 without line 3. You could get very wierd results. Since ptrX hasn't been pointed to our var_x, maybe its points to system memory, and then you assign a value someplace you shouldn't and your computer crashes. This may not always happen, but it is certainly a possibility, so be very careful when using pointers. Make sure they are assigned to something before you use them.

UNIVERSITY Of KIGALI