Contents

Unit 7
Pointers

7.0. Structure of the Unit

7.1. Unit Outcomes

After the successful completion of this unit, the student will be able to:

1. Explain the pointer declaration, reference, and dereference.
2. Describe the concept of dynamic memory allocation.
3. Apply pointer arithmetic and demonstrate pointers with functions.
4. Discuss memory leak and segmentation fault in pointers.
5. Develop programs using malloc(), free(), realloc() functions.

7.2. Introduction to Pointers

In programming, a pointer is a variable that stores the memory address of another variable. Every variable is assigned a specific memory location and each memory location is identified by a unique address. A pointer points to a specific memory location and holds the value stored in that location, as illustrated in Figure 7.1.

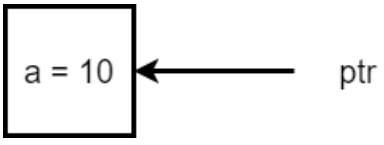

Figure 7.1 A pointer variable pointing to an integer variable

Important characteristics of pointers are as follows:

1. As a pointer points to a memory location, changing, modifying, returning data, etc. operations are simpler.
2. The data manipulation is directly on the memory location, which reduces the program execution time.
3. The memory allocation is done dynamically during run-time based on the actual requirements. With this, memory is utilized efficiently.
4. When the memory is no longer required for the variable, it is deallocated. This avoids duplication and memory utilization is better.
5. Several data structures such as arrays, queues, stacks, linked lists, etc. use pointers for information retrieval, storage, and processing. etc.
6. For large voluminous data, the pointer is the better choice than other static-allocation-based variables such as arrays, structures, etc.
7. As the memory is allocated on a need basis, the pointer is a good choice when the size of the data is not known. If the number of records is not fixed in an application, memory can be allocated and deallocated. This is not possible in arrays, structures, or any other built-in variables.

Example: Consider an example of creating a cell phone number directory of Uttarakhand state. There may be several additions and deletions of phone records. If this program was implemented using arrays, the size of the array should be predefined. This needs a fixed memory assignment, leading to wasting of memory. It can be avoided with the use of pointers. This application can be implemented using a linked list of pointers, where the memory locations are added and deleted based on the requirements.

Pointer Declaration: The pointer variable is declared as given below.

General Syntax: Type *variableName;
Example: `int *ptr;`

C program using pointer variables is shown in Program 1.

```
#include <stdio.h>
int main(void)
{
 int *ptr;
 int a=10;
 ptr = &a;
 printf("\n The value is %d",*ptr);
 printf("\n The address is %u”,ptr);
 return 0;
}
```

Sample Input/Output

```
The value is 10
The address is 1154169364
```

Program 1: Pointer Data Types

Here asterisk * is used to denote the variable of the type of pointer. ptr is the pointer variable of the integer type. integer. The ptr is pointing to a memory location where the value a is saved. The & is

an address operator used to get the address. The first printf statement prints the value 10. The second printf statement prints the address of a which is of the hexadecimal type, for example, 0XX67.
The asterisk is known as the dereferencing operator. Pointers are declared for different data types as given below.

```
int *p;
float *q;
double *ptr;
 char *ch;
```

Address Operator: The address operator is used to return the memory address of a variable. The address
operator is represented using &. The address operator is also used to pass and return reference variables.
This operator is also used to establish the pointer values.
The Dereferencing Operator: It is also known as the indirection operator represented using (*). It is used to return the value pointer by a pointer. The dereferencing operator can be also used to insert or modify the value of pointers. This is shown below.

```
#include <stdio.h>
int main()
{
    int x=40;
    int y;
    int *p;
    p=&x;
    y=*ptr;
    *p=50;
    printf("The value of x is : %d",x);
    printf("\n The value of y is : %d",y);
    return 0;
}
```

Program 2: dereferencing Operator

Types of Pointers: Some of the commonly used pointers are as follows:

1. NULL pointer: A Null pointer does not point to any location. If the pointer is to be pointed to an unknown location, it is initialized to NULL. It is used in data structures like stack, queue, tree, linked list, etc. It is declared as int *p = NULL;
2. void pointer: void pointer refers to a pointer that is not pointing to any data type. This pointer can hold the address of any data type using type conversion techniques.
3. Dangling pointer: When a block of data is deallocated, then a pointer pointing to that memory location is called as dangling pointer.

7.3. Pointer Arithmetic

Pointers are used in a limited number of arithmetic operations as listed below.

1. Increment and Decrement
2. Addition and Subtraction
3. Comparison
4. Assignment

Increment and Decrement: Let *p be an integer pointer that points to memory location 2000. Consider the size of the integer is 32 bits (4 bytes). When a pointer is incremented as p++, it jumps to an address 2004. When the pointer is decremented using p--, it will go back to the address 2000.
Addition and Subtraction: After adding an integer value to a pointer, the size of the data type is multiplied by the value and added to the pointer. This is shown in Program 3.

```
#include <stdio.h>
int main()
{
      int N = 40, M = 100;
      int *p1,*p2;
      // Pointer stores the address of N
      p1 = &N;
      printf("Pointer P1 before Increment: %p\n",p1);
      p1 = ++p1;
      printf("Pointer P1 after Increment %p\n ",p1);
      p2 = &M;
      printf("Pointer P2 before Decrement: %p\n",p2);
      p2 = --p2;
      printf("Pointer P2 after Decrement %p\n ",p2);
      return 0;
}
```

Sample Input/Output

```
Pointer P1 before Increment: 0x7fff326fb30c
Pointer P1 after Increment 0x7fff326fb310
Pointer P2 before Decrement: 0x7fff326fb308
 Pointer P2 after Decrement 0x7fff326fb304
```

Program 3: Increment and Decrement Operators on Pointers

. Comparison of pointers: Comparison operators can be used on pointers. The different comparison operators are <, <=, >, >=, == and !=.The output of the comparison is True or False.

```
#include <stdio.h>
int main()
{
      int x;
      int* p1 = &x;
      // declaring a pointer to the first element
      int* p2 = &x;
      if (p1 == p2) {
```

```
        printf("Pointer to x and y are Equal.");
        }
        else {
        printf("Pointer to x and y are not equal.");
        }
        return 0;
}
```

Sample Input/Output

```
Pointer to x and y are Equal.
```

Program 4: Comparison Operator on pointers

```
  #include <stdio.h>
  int main()
  {
        int x,y;
        printf("Enter two numbers:");
        scanf("%d %d",&x,&y);
        int *p1;
        int *p2;
        // Initializing pointer variables
        p1 = &x;
        p2 = &y;
        if (*p1 < *p2) {
        printf(
                "%d is less than %d.", *p1, *p2);
        }

        // Greater than operator
        if (*p1 > *p2) {
        printf(
                "%d is greater than %d.", *p1, *p2);
        }

        // Equal to
        if (*p1 == *p2) {
        printf(
                "%d is equal to %d.", *p1, *p2);
        }

        return 0;
}
```

Sample Input/Output

```
Enter two numbers 10
30 is greater than 10.
```

Program 5: Relational Operators on Pointers

Assignment: Pointers can be assigned with values or addresses of variables. One pointer can be assigned to another, where the addresses of pointers are copied. It is shown in Program 6.

```
  #include <stdio.h>
```

```
int main() {
    int num1, num2, *p1, *p2;
    num1 = 10;
    p1 = &num1;
    num2 = 20;
    p2 = &num2;
    printf("Address are %p and %p for p1 and p2",p1,p2);
    p1 = p2;
    printf("\n Address of p1 is changed to %p",p1);
    return 0;
}
```

Sample Input/Output:

```
Address are 0x7ffec02d185c and 0x7ffec02d1858 for p1 and p2
The address of p1 is changed to 0x7ffec02d1858
```

Program 6: Assignment Operation using pointers

7.4. Pointers and Functions

Pointers can be used to store the references to functions. In these cases, the pointer is known as a function pointer, and it contains the address of a function. Function pointers are efficient and are used in late or run-time binding. The function pointer is written as follows:

```
int (*ptr)(int x, int y); /* a pointer to function */
```

Pointers can also be passed to a function as given below.

```
#include <stdio.h>
void test_run( int a, int *ptr1 ) {
       a = 200;
      *ptr1 = 200;
      return;
}
int main( void )
{
    int x = 1, y = 2;
    printf( "Initial value of x = %d and y = %d\n", x, y );
    test_run(x, &y );
  printf( "After passing y to test function: x= %d, y = %d\n",
      x, y );
  return 0;
}
```

Sample Input/Output:

```
Initial value of x = 1 and y = 2
After passing y to test function: x= 1, y = 200
```

Program 7: Pointers passed as arguments to Function

In Program 7, an address of variable y is passed to the function test_run(). A pointer is used in the function to modify the value to 200. The value of y is not returned to the main() program. The pointer modifies the location of y and the value is changed. A pointer can be returned from the function as

given below.

```
#include <stdio.h>
int* min(int *n1, int *n2) {
 if (*n1 < *n2)
     return n1;
 return n2;
}

int main() {
 int x = 15, y = 100, *ptr;
 printf("Original Numbers: x = %d, y = %d",x , y);
 ptr = min(&x, &y);
 printf("\nMinimum of said two numbers: %d", *ptr);
 return 0;
}
Sample Input/Output:
Original Numbers: x = 15, y = 100
Minimum of said two numbers: 15
```

Program 8: Pointers returned from the Function

7.5. Dynamic Memory Allocation

Dynamic memory allocation is a technique where the size of the memory is allotted during runtime. There are four functions in the C programming language to allocate the memory dynamically. These functions are as follows.

`malloc()`: `malloc` stands for memory allocation, it is used to allocate a single block of memory of the size necessary for a specific data type. This function returns a pointer of the type of void and initially assigns a garbage value for each block. The use of malloc is given as follows.

General Syntax: ptr = (cast-type*) malloc(byte-size)

Example:

```
int *ptr;
ptr = (int*) malloc(sizeof(int));
```

The `malloc()` function shown above returns a pointer pointing to an integer data type. A block of 4 bytes is assigned to a ptr. The 4 blocks are determined using the sizeof operator. If there is no space allocated, then a NULL pointer is returned.

`calloc()`: `calloc` function is used for allocating a specific number of blocks in a contiguous manner. There are two arguments in this function: The first one refers to the number of arguments and the second returns number of bytes. The use of calloc is as given below.

```
int n, *ptr;
n = 2;
 ptr = (int*)calloc(n, sizeof(int));
```

`free()`: The `free` function is used to deallocate or delete the memory allocated by malloc or calloc. It is helpful to prevent the wastage of memory and make it available for the rest of the program. The `free` function is used as follows.

```
free ptr;
```

`realloc()`: `realloc`() function is used for changing the previous allocation in a dynamic method. If the memory created by `malloc()` or `calloc()` is not sufficient, then `realloc()` can be used. The `realloc` function is used as follows.

```
int *ptr;
ptr = realloc(ptr, 10 * sizeof(int));
```

A C program using all the dynamic memory allocation/deallocation functions is given in Program 8.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int n,*ptr;
    printf("Enter a number: ");
    scanf("%d",&n);
    ptr=(int*)malloc(sizeof(int));  //memory allocated using
      malloc
    if(ptr==NULL)
    {
        printf("Memory is not allocated");
        exit(0);
    }
    *ptr = n;
    if(*ptr%2==0)
    printf("Pointer is pointing to even number");
    else
    printf("Pointer is pointing to odd number");
    free(ptr);
return 0;
}
```

Sample Input/Output

```
Enter a number: 4
Pointer is pointing to even number
Enter a number: 7
Pointer is pointing to odd number
```

Program 8: malloc() and free() Functions

7.6. Self-Assessment Questions/Activities

Q1.Discuss the features of pointers with an example. [5 Marks, L1]

Q2. Explain the following with a suitable example. [6 Marks, L2]

1)Pointer Declaration

2) Pointer Initialization

Q3. Discuss the following related to pointers. [10 Marks, L2]

1) Address Operator
2) Dereferencing Operator

Q4. Compare static and dynamic memory allocation. [8 Marks, L2]

Q5. Explain the NULL pointer and void pointer with an example. [5 marks, L2]

Q6. Discuss examples using malloc() and free() functions. [8 Marks, L2]

Q7. What are realloc() and calloc() functions? [8 Marks, L2]

Q8. Discuss the advantages of pointers. [8 Marks, L2]

Q9. Explain with a suitable example how a pointer can be passed and returned to function with an example. [8 Marks, L1]

Q10. Write a C program to create memory locations for the data types int, float and char. Explain the functions of malloc() and free() functions. [8 Marks, L1]

7.7. Multiple-Choice Questions

Q1. What is the output of the following program? [1 Mark, L2]

```
#include <stdio.h>
int sum (int, int);
int main (void)
{
    int total;
    total = sum (10, 20);
    printf ("%d\n", total);
    return 0;
}

int sum (int a, int b)
{
      int s;
      s=a+b;
      return s;
}
```

A. 30
B. 20
C. 10
D. 50

Q2. Any C program must contain_____ [1 Marks, L1]

A. At least one function
B. Need not contain any function
C. Variables and constants
D. None of the above

Q3. _____ of the following gives return value. [1 Mark, L1]

A. static
B. for
C. return
D. const

Q4. The default return type of a function is_____ [1 Mark, L2]

A. float
B. void
C. int
D. automatic

Q5. Default parameter passing mechanism is_____ [1 mark, L3]

A. Call by Value
B. Call by Reference
C. Any of the technique
D. None of the above

Q6. Recursion is handled using _____ data structure. [1 Mark, L2]

A. Queue
B. Arrays
C. Stack
D. Structure

Q7. The recursive function can be terminated using____ [1 Mark, L3]

A. Function Call
B. Loop
C. Function Prototype
D. Base Condition

Q8. Determine the output of the following program [1 Mark, L3]

```
void print(int n)
{
if (n == 0)
return;
printf("%d", n%2);
print(n/2);
}
int main()
{
print(12);
return 0;
}
```

A. 0000
B. 0001

C. 1000
D. 0100

Q9. When a function is called from the main() program the memory is allocated using___[1 Mark, L2]

A. Random Access Memory
B. Stack
C. Arrays
D. Compiler

Q10. What is the output of the following program? [1 Mark, L3]

```
int main()
{
    printf("Happy Birthday");
    main();
    return 0;
}
```

A. Happy Birthday
B. Infinite Loop
C. Compilation Error
D. Garbage Value

7.8. Keys to Multiple-Choice Questions

Q1.30 (a)

Q2.At least one function (a)

Q3.return (c)

Q4.int (c)

Q5.Call by value

Q6.stack (c)

Q7.base condition (d)

Q8.0001 (b)

Q9.Stack (b)

Q10. Infinite loop (b)

7.9. Summary of the Unit

Functions are useful in supporting the divide and conquer concept and modular programming. A set of repetitive statements can be replaced by a user-defined function. Functions can be library or user-defined.

Function declaration specifies the return type, number, and type of arguments/parameters. A function can call itself till a bae condition is satisfied. This concept is known as recursion. Recursive functions make the program compact and reduce time and space efficiency.

7.10. Keywords:

- Library Functions
- User-defined Functions
- Function prototype, call, return
- Recursive Functions

7.11. Recommended Learning Resources

[1] Herbert Schildt. (2017). *C Programming: The Complete Reference*. 4th ed. USA: McGraw-Hill.