CHAPTER - 7
POINTER DATA TYPE

CHAPTER 7

POINTER DATA TYPE

INTRODUCTION **(Reading)**

POINTER DECLARATION

ADDRESS OPERATOR

POINTER EXPRESSION AND POINTER ARITHMETIC

POINTER ON POINTER **(Reading)**

VOID REVISITED **(Reading)**

GENERIC, NULL AND INVALID POINTERS

POINTER DATA TYPE

POINTER DECLARATION

- **A pointer is a variable declared like any other data type and takes space in memory to hold the value**
- **Pointer contains the memory location of another variable**
- **Asterisk before the variable name and after the data type tells the complier that the pointer variable is pointing to that data type**

syntax:

data type * variable name

examples:

```
int *ptr;
float *string;
char *mycharp;
```

POINTER DATA TYPE

ADDRESS OPERATOR

- **The unary operator & returns a pointer to its operand**
- **The address operator applied to a function designator yields a pointer to the function**
- **A function pointer generated by the address operator is valid throughout the execution**
- **An object pointer generated by the address operator is valid as long as the object's storage remains allocated**

syntax:

Address expression: & unary-expression

examples:

```
int num = 5;
int *ptr = &num;;
extern int f ();
int (*fp) ();
fp = &f;    // & yields a pointer of f
```

POINTER DATA TYPE

CONTENTS OF POINTER DATA TYPE

```
/* display the contents of the variable, their address using pointer*/
include< stdio.h >
void main()
{
    int num, *intptr;
    float x, *floptr;
    char ch, *cptr;
    num = 123;
    /* warning C4305: '=' : truncation from 'const double' to 'float' */
    x = 12.34;
    ch = 'a';
    intptr = &num;
    cptr = &ch;
    floptr = &x;
    printf ("Num %d stored at address %u\n", *intptr, intptr);
    printf ("Value %f stored at address %u\n", *floptr, floptr);
    printf ("Character %c stored at address %u\n", *cptr, cptr);
}
```

POINTER DATA TYPE

POINTER EXPRESSION AND POINTER ARITHMETIC

- **Pointer variables can be used in expressions.**
- **C allows us to add integers to or subtract integers from pointers as well as to subtract one pointer from the other.**
- **Pointers can be compared by using relational operators**

examples:

```
y = *p1**p2;
sum = sum + *p1;
z =  5* - *p2/*p1;
*p2 = *p2  + 10;
```

POINTER DATA TYPE

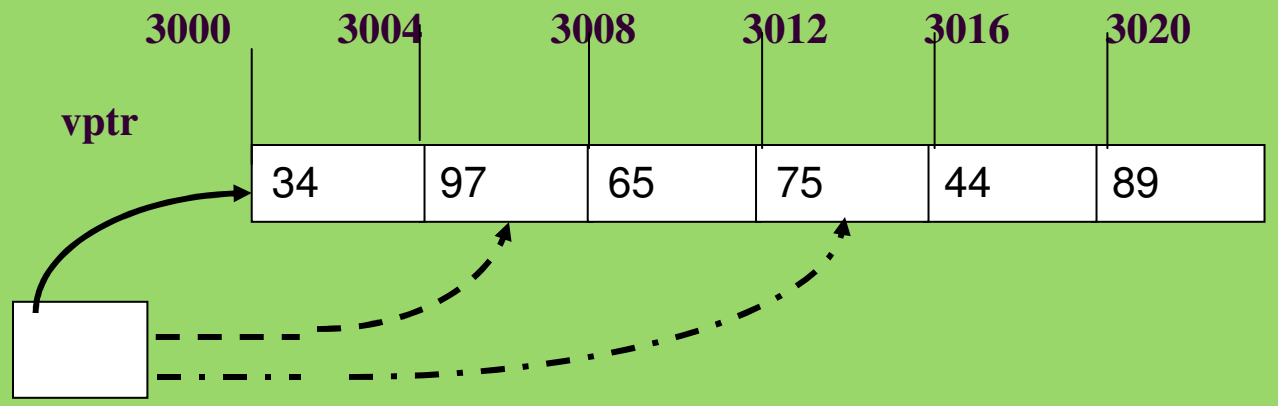
EXAMPLE OF POINTER ARITHMETIC

```
/*Program to illustrate the pointer expression and pointer arithmetic*/
#include< stdio.h >
void main()
{
        int * ptr1, * ptr2, a, b, x, y, z;
        a = 30; b = 6;
        ptr1 = &a;
        ptr2 = &b;
        x = *ptr1 + *ptr2  – 6;
        y = 6 - *ptr1 /  *ptr2 + 30;
        printf ("\n Address of a + %u", ptr1);
        printf ("\n Address of b %u", ptr2);
        printf ("\n a = %d, b = %d", a, b);
        printf ("\n x = %d, y = %d", x, y);
        ptr1 = ptr1 + 70;
        ptr2 = ptr1;
        printf("\n a = %d, b = %d", a, b);
}
```

POINTER DATA TYPE

POINTER EXPRESSION AND POINTER ARITHMETIC

int v[] = {34, 97, 65, 75, 44, 83};
int *vptr;
The following two statements achieve the same result:
vptr = v;
vptr = &vp[0];



vptr* is initialized to point to the array *v* or to *v[0]*. The value of *vptr* is *3000*, which is the location of *v[0].

vptr points to 34, vptr++ points to 97 and vptr + 2 points to 75

POINTER DATA TYPE

EXAMPLE USING POINTER

```
int main (void)
{
    int a = 7, *aptr;  // aptr is pointer to integer
    aptr = &a;   /* aptr is set address of a */
    printf ("\n The address of a is %p \n"
            "The value of aptr is %p \n\n", &a,
                                            aptr);

    printf ("\n The value  of a is %d \n"
            "The value of *aptr is %d \n\n", a,
                                             *aptr);

    printf ("\n Proving that * and & are
    complements of "
                    "each other.\n &*aptr = %p \n
        *&aptr = %p\n", &*aptr, *&aptr);
    return 0;
}
```

Output of the above program:

```
The address of a is 0012FF7C
The value of aptr is 0012FF7C
 The value  of a is 7
The value of *aptr is 7
 Proving that * and & are
    complements of each other.
 &*aptr = 0012FF7C
 *&aptr = 0012FF7C
```

POINTER DATA TYPE

GENERIC, NULL AND VOID POINTERS

- Pointers types are object pointers or function pointers depending on whether the pointer pointing is an object or a function
- C introduces the type void * as a “generic pointer”. Any pointer to an object can be converted to type void * and back without change
- When a function has a formal parameter that can accept a pointer of any type, the formal parameter should be declared to be of type void *
- C has a special “null pointer” value that explicitly points to no object or function. The null pointer may be written as the integer constant 0 or 0L or as (void *) 0
- It is also possible to create invalid pointers, that is, pointer values that are not null but also do not designate a proper object or function
- Invalid pointers can be created by casting arbitrary integer values to pointer type, by de-allocating the storage for an object to which a pointer refers, or by using pointer arithmetic to produce a pointer outside the range of an array