POINTERS

INTRODUCTION

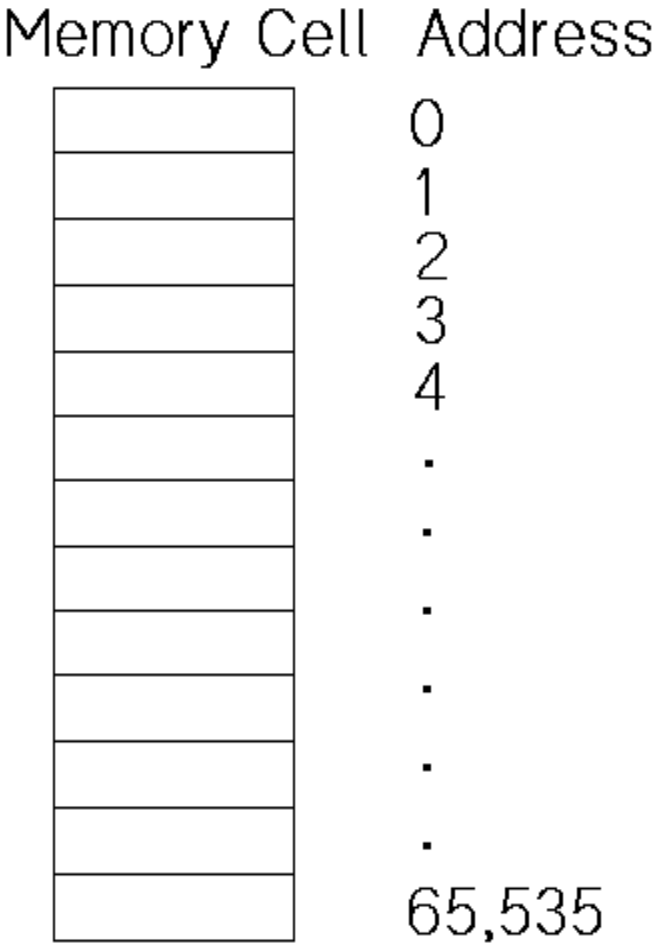
- A pointer is a derived data type in C.
- Pointers contain memory addresses as their values.
- Since these memory addresses are the locations in the computer memory where program instructions and data are stored, pointers can be used to access and manipulate data stored in the memory.

POINTERS BENEFITS TO THE PROGRAMMER

- Pointers are more efficient in handling arrays and data tables.
- Pointers can be used to return multiple values from a function via function arguments.
- Pointers permit references to functions and thereby facilitating passing of function as arguments to other functions.
- The use of pointers arrays to character stings results in saving of data storage space in memory.
- Pointers allow C to support dynamic memory management.
- Pointers provide an efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.
- Pointers reduce length and complexity of programs.
- They increase the execution speed and thus reduce the program execution time.

UNDERSTANDING POINTER

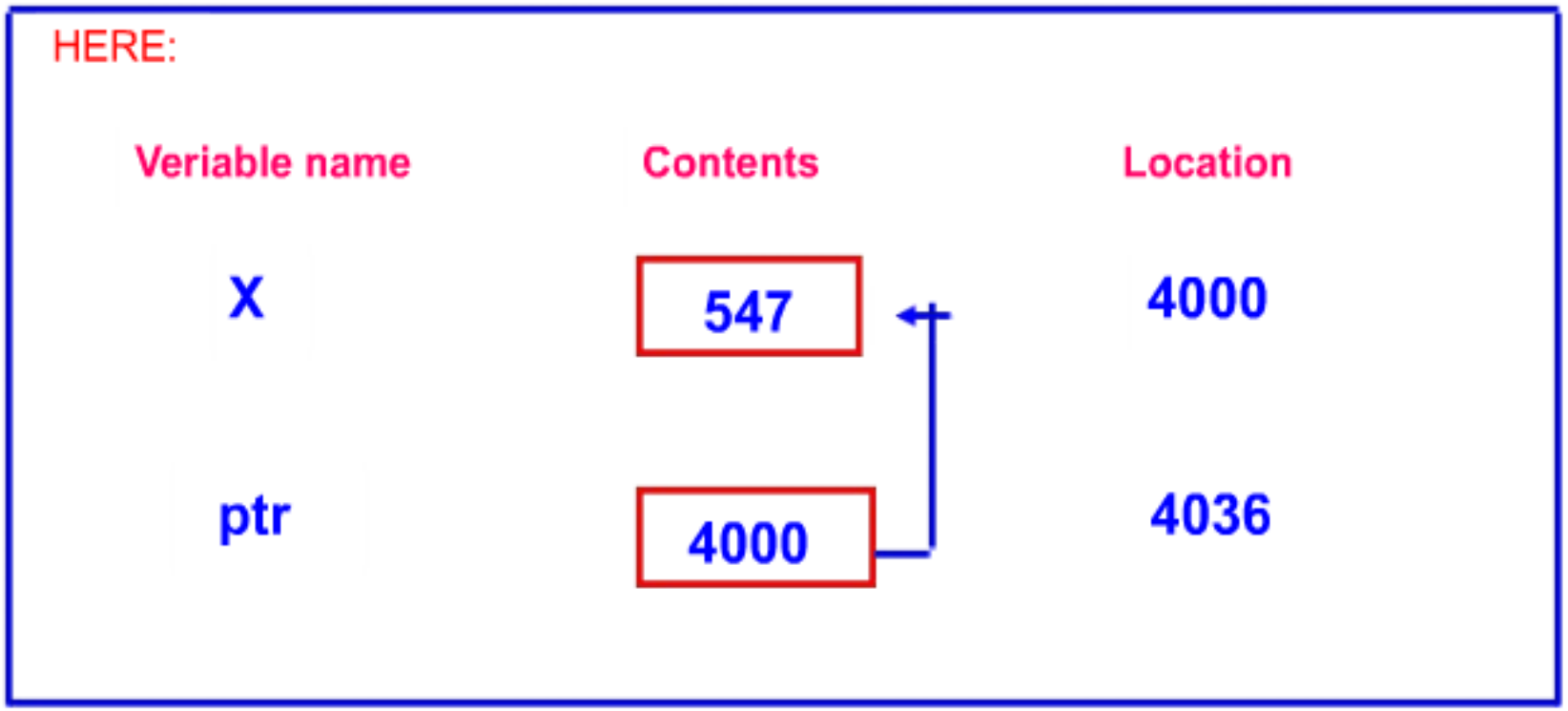
- The computer's memory is a sequential collection of storage cells
- Each cell. Commonly known as a byte, has a number called address associated with it
- The address are numbers consecutively, starting from zero
- The last address depends on the memory size
- A computer system having 64K memory will have its last address as 65,535



EXAMPLE FOR POINTER:

```
Int X = 547;
```

HERE:



According to above figure, By the help of ptr variable we stored the address of variable X in address 4036

POINTER VARIABLE

- We may access the value 547 by using either the name X or the address 4000.
- Since memory addresses are simply numbers, they can be assigned to some variables, that can be stored in memory, like any other variable.
- Such variables that hold memory addresses are called pointer variables.
- A pointer variable is, nothing but a variable that contains an address, which is a location of another variable in memory.

ACCESSING THE ADDRESS OF A VARIABLE

- The actual location of a variable in the memory is system dependent.
- We can determine the address of a variable with the help of the operator & available in C.
- The operator & immediately preceding a variable returns the address of the variable associated with it.

p = &quantity

- Would assign the address 5000 to the variable p

POINTER DECLARATION

In 'c' Every Variable must be Declared its type ,since pointer variables contain address of separate Data type , They must be Declared before use them the declaration of pointer variable takes the following form

syntax:

```
data _type * pt_ name
```

EXAMPLE:

1. Int * p ; / * integer pointer */ & 2. float *p; /* float pointer */

HERE:

1. Declares the variable 'p' as a pointer variable that points to an integer data type

2. Declares the variable 'p' as a pointer variable that points to an 'float' data type

POINTER DECLARATION STYLES

```
int*   p ;   // * style 1*//
int  *p ;  //* style 2*//
int *  p ; //*style 3*//
```

However ,the style 2 is more popular due to following reasons;

1. This style is convenient to have multiple declaration in the same statement

Ex:

```
int *p, x,*q ;
```

2. This style matches with the format used for accessing the target values.

Ex:

```
int x, *p , y;
  x=10;
  y=*p;
 *p = 20;
```

POINTER INITIALIZATION

As ordinary variables are Initialized with in Declaration part of the program , The Pointer variables can initialized by Accessing address of the variable that are Used in the program .

Syntax :

```
Data_type *ptr = expression
```

Where:

data_type = Any Basic Data type

*ptr = pointer variable

expression = another variable or may be constant

Example for pointer initialization

```
Int quantity ;
Int *p ;                    // * Declaration *//
p = & quantity ;  //  * initialization*//
```

Here;

the third line assign the address of ‘quantity’ to pointer variable ‘p’

Invalid initializations

1.
```
float a , b ;
int  x ,  * p ;
p = & a ;          // * wrong *//
b =    * p ;
```

Here;

we will get erroneous output because we are trying to assign the address of float variable to an integer variable it is not allowed in pointer assignments.

2.
```
Int * p = & x , x ;    // * wrong * //
```

Here;

it declares ‘ x ’ as integer variable, ‘p’ as pointer variable then assign ‘ p’ to the address of ‘x’ . first we must be declare the ‘x’ before initialization hence above statement gives the erroneous output

ACCESSING A VARIABLE THROUGH ITS POINTER

We access the value of the variable by help of 'pointer' this is done by using another unary operator * (asterisk) usually known as " Indirection operator "

consider following statements

```
int quantity, *p ,n;      ( first line )
quantity = 179;          (second line)
p = & quantity;          (third line)
n = *p;                  (fourth line)
```

EXPLANATION:

The First line declares 'quantity' & 'n' as integer variable and 'p' as pointer variable

The second line Assigns value 179 to variable quantity

The third line assigns address of variable quantity to the pointer variable 'p'

the forth line contains ' * ' operator in this case *p returns value of variable quantity Now value of 'n' would be 179

the Two statements

```
p = & quantity;
n = * p;
```

Are Equivalent to

```
n = * & quantity;
```

which in turns is equivalent to

```
n = quantity;
```

PROGRAM EXAMPLE FOR POINTER INITIALIZATION

```
# include < stdio.h>
Void main ()
int m,*ptr ;
m = 270;
ptr = & m;
printf (" The content of variable 'm' = %d\n",*ptr);
*ptr= 434 ;   /* pointer initialization*/
Printf(" the content of the variable 'm' after initialization = %d\n", *ptr);
```

OUT PUT:

The content of the variable 'm' = 270

The content of the variable 'm' after initialization = 434

Pointers are used in following variables

- Variables of any basic data type
- An Array
- Functions
- Structures, and
- Unions

CHAIN OF POINTERS

- It is possible to make a pointer to point to another pointer, thus creating a chain of pointers.

- The pointer variable p2 contains the address of the pointer variable p1, which points to the location that contains the desired value. This is known as multiple indirections.
- A variable that is a pointer to a pointer must be declared using additional indirection operator symbols in front of the name.
- This declaration tells the compiler that p2 is a pointer to a pointer of int type.

CHAIN OF POINTERS

- We can access the target value indirectly pointed to by pointer to a pointer by applying the indirection operator twice.

```
main()
{
 int x, *p1, **p2;
 x = 100;
 p1 = &x;              /* address of x */
 p2 = &p1;             /* address of p1 */
 printf("%d", **p2);
}
```