

## IT 112: Introduction to Programming

Dr. Manish Khare  
Dr. Bakul Gohel

Lecture 17,18,19  
(POINTER)



# Pointers in C

- In c programming language, we use normal variables to store user data values. When we declare a variable, the compiler allocates required memory with specified name.
- In c programming language, every variable has name, datatype, value, storage class, and address.
- We use a special type of variable called pointer to store the address of another variable with same datatype.



ptr \*

The diagram consists of two light gray rectangular boxes. The left box contains the text 'ptr \*'. A line extends from the bottom of this box, goes down, then right, then up again, ending in an arrowhead pointing at the bottom of the right box. The right box contains the text 'var'.

var

Pointers

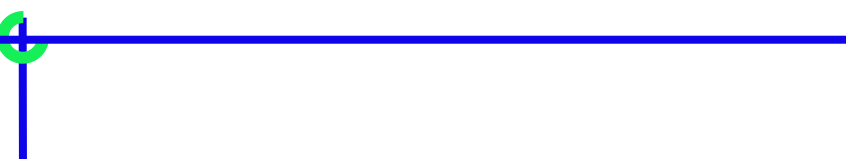


➤ Pointer is defined as follows...

- **Pointer is a special type of variable used to store the memory location address of a variable.**

➤ In c programming language, we can create pointer variables of any datatype. Every pointer stores the address of variable with same datatype only. That means, integer pointer is used store the address of integer variable only.

# Address Operation

- 
- There are two unary operations to consider.
    - The `*` operator: If `ptr` is a pointer variable, then `*ptr` gives you the content of the location pointed to by `ptr`.
    - The `&` operator: If `v` is a variable, then `&v` is the address of the variable.

# Accessing the Address of Variables

- In c programming language, we use the **reference operator** "&" to access the address of variable. For example, to access the address of a variable "**marks**". we use "**&marks**".
- We use the following printf statement to display memory location address of variable "**marks**"...
  - `printf("Address : %u", &marks);`



---

**`/* Example to demonstrate use of reference operator in C programming. */`**

`#include <stdio.h>`

`int main()`

`{`

`int var = 5;`

`printf("Value: %d\n", var);`

`printf("Address: %u", &var); //Notice, the ampersand(&) before var.`

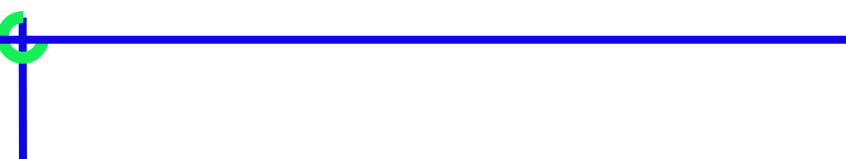
`return 0;`

`}`

## **Output**

Value: 5

Address: 2686778

- 
- **Note:** You may obtain different value of address while using this code.
  - In above source code, value 5 is stored in the memory location 2686778. var is just the name given to that location.




# Declaring Pointers (Creating Pointers)

- In c programming language, declaration of pointer variable is similar to the creation of normal variable but the name is prefixed with \* symbol.
- We use the following syntax to declare a pointer variable...
  - **datatype \*pointerName ;**
- A variable declaration prefixed with \* symbol becomes a pointer variable.
  - **int \*ptr;**
- In the above example declaration, the variable "**ptr**" is a pointer variable that can be used to store any integer variable address.

# Assigning Address to Pointer

- To assign address to a pointer variable we use assignment operator with the following syntax...
  - **pointerVariableName = & variableName ;**
- For example, consider the following variables declaration...
  - `int a, *ptr ;`
- In the above declaration, variable "**a**" is a normal integer variable and variable "**ptr**" is an integer pointer variable. If we want to assign the address of variable "**a**" to pointer variable "**ptr**" we use the following statement...
  - `ptr = &a ;`
- In the above statement, the address of variable "**a**" is assigned to pointer variable "**ptr**". Here we say that pointer variable **ptr** is pointing to variable **a**.

- 
- Declaring a pointer

`int *p;`

data type of pointer      name of the pointer variable  
recognize that p is pointer  
(space doesn't matter, position does)

- Assigning value to pointer

```
int *p;  
int j = 25;  
p = &j;
```

`p = &j;`


address-of operator

# Reference operator (&) and Dereference operator (\*)

- As discussed, & is called reference operator. It gives you the address of a variable.
- Likewise, there is another operator that gets you the value from the address, it is called a dereference operator (\*).
- Below example clearly demonstrates the use of pointers, reference operator and dereference operator.

# Accessing Variable Value Using Pointer

- Pointer variables are used to store the address of other variables. We can use this address to access the value of the variable through its pointer.
- We use the symbol "\*" in front of pointer variable name to access the value of variable to which the pointer is pointing.
- We use the following general syntax...
  - **\*pointerVariableName**



---

/\* Source code to demonstrate, handling of  
pointers in C program \*/

#include <stdio.h>

int main(){

int\* pc;

int c;

c=22;

printf("Address of c:%u\n",&c);

printf("Value of c:%d\n\n",c);

pc=&c;

printf("Address of pointer pc:%u\n",pc);

printf("Content of pointer pc:%d\n\n",\*pc);

c=11;

printf("Address of pointer pc:%u\n",pc);

printf("Content of pointer pc:%d\n\n",\*pc);


\*pc=2;

printf("Address of c:%u\n",&c);

printf("Value of c:%d\n\n",c);

return 0;

}



---

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2

# Common mistakes when working with pointers

➤ Suppose, you want pointer `pc` to point to the address of `c`. Then,

- `int c, *pc;`
- `pc = c;`      // Wrong! `pc` is address whereas, `c` is not an address.
- `*pc = c;`    // Correct! `*pc` is the value pointed by address and, `c` is also a value.
- `*pc = &c;`    // Wrong! `*pc` is the value pointed by address whereas, `c` is an address.
- `pc = &c;`      // Correct! `pc` is an address and, `c` is also an address.



# NULL Pointers

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.
- The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program

—



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

```
int main () {
```

```
    int *ptr = NULL;
```

```
    printf("The value of ptr is : %x\n", ptr );
```

```
    return 0;
```

```
}
```


## **Output**

The value of ptr is 0

# Size of basic data types


- All data is stored in memory. But different data types occupy different amount of memory.
- The sizeof() operator in C can be used to determine the **number of bytes** occupied by each data type.
  - For example, on some machine you may have
    - sizeof(int) = 4
    - sizeof(float) = 4
    - sizeof(double) = 8
- These numbers are Not the same for all machines. You should use the sizeof() operator instead of assuming the value.

# Pointers and Functions in C



➤ In c programming language, there are two ways to pass parameters to functions. They are as follows...

- Sending the values of the arguments (Call by Value)
- Sending the addresses of the arguments (Call By Reference)

- 
- In the first method the ‘value’ of each actual arguments in the calling function is copied into corresponding formal argument of the called function.
  - With this method, changes made to the formal arguments in the called function have no effect on the values of the actual arguments in the calling function.

# Example

```
main()
```

```
{
```

```
int a = 10;
```

```
int b = 20;
```

```
swapv(a,b)
```

```
printf("\na=%d", a);
```

```
printf("\nb=%d", b);
```

```
}
```

```
swapv(int x, int y)
```

```
{
```

```
int t;
```

```
t=x;
```

```
x=y;
```

```
y=t;
```

```
printf("\n x=%d", x);
```

```
printf("\n y=%d", y);
```

```
}
```


## Output:

```
x=20
```

```
y=10
```

```
a=10
```

```
b=20
```

- 
- The second method (call by reference) the addresses of actual arguments in the calling function are copied into formal arguments of the called function.
  - Means by using the formal arguments in the called function we can make changes in the actual arguments of the calling function

# Example

```
main()
{
    int a = 10;
    int b = 20;
    swapr(&a, &b)
    printf ("\n a = %d", a);
    printf ("\n b = %d", b);
}
```


```
swapr(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}
```

## Output

a = 20

b = 10





➤ Using ‘call by reference’ intelligently we can make a function return more than one value at a time, which is not possible ordinarily.

➤ Example

# Example

```
main()
{
    int radius;
    float area, perimeter;
    printf("\n Enter radius of a circle");
    scanf("%d", &radius);
    areapri(radius, &area, &perimeter);
    printf("Area=%f", area);
    printf("\nPerimeter = %f", perimeter);
}
```

```
areaperi(int r, float *a, float *p)
{
    *a=3.14*r*r;
    *p=2*3.14*r;
}
```

## Output

Enter radius of a circle 5

Area = 78.500000

Perimeter = 31.400000

# Pointers Arithmetic Operations in C

- Pointer variables are used to store address of variables. Address of any variable is an unsigned integer value i.e., it is a numerical value. So we can perform arithmetic operations on pointer values. But when we perform arithmetic operations on pointer variable, result depends on the amount of memory required by the variable to which the pointer is pointing.
- In c programming language, we can perform the following arithmetic operations on pointers...
  - Addition
  - Subtraction
  - Increment
  - Decrement
  - Comparison

# Addition and Subtraction Operation on Pointer

➤ In C programming language, the addition operation on pointer variables is calculated using the following formula...

- **$\text{AddressAtPointer} + (\text{NumberToBeAdd} * \text{BytesOfMemoryRequiredByDatatype})$**

➤ the subtraction operation on pointer variables is calculated using the following formula...

- **$\text{AddressAtPointer} - (\text{NumberToBeAdd} * \text{BytesOfMemoryRequiredByDatatype})$**

# Pointers Arithmetic Operations in C

- Pointer variables are used to store address of variables. Address of any variable is an unsigned integer value i.e., it is a numerical value. So we can perform arithmetic operations on pointer values. But when we perform arithmetic operations on pointer variable, result depends on the amount of memory required by the variable to which the pointer is pointing.
- In c programming language, we can perform the following arithmetic operations on pointers...
  - Addition
  - Subtraction
  - Increment
  - Decrement
  - Comparison

# Addition and Subtraction Operation on Pointer

➤ In C programming language, the addition operation on pointer variables is calculated using the following formula...

- **$\text{AddressAtPointer} + (\text{NumberToBeAdd} * \text{BytesOfMemoryRequiredByDatatype})$**

➤ the subtraction operation on pointer variables is calculated using the following formula...

- **$\text{AddressAtPointer} - (\text{NumberToBeAdd} * \text{BytesOfMemoryRequiredByDatatype})$**

# Addition Example

```
void main()
{
    int a, *intPtr ;
    float b, *floatPtr ;
    double c, *doublePtr ;
    clrscr() ;
    intPtr = &a ; // Asume address of a is 1000
    floatPtr = &b ; // Asume address of b is 2000
    doublePtr = &c ; // Asume address of c is 3000

    intPtr = intPtr + 3 ; // intPtr = 1000 + ( 3 * 4 )
    floatPtr = floatPtr + 2 ; // floatPtr = 2000 + ( 2 * 4 )
    doublePtr = doublePtr + 5 ; // doublePtr = 3000 + ( 5 * 8 )
```

```
printf("intPtr value : %u\n", intPtr) ;
printf("floatPtr value : %u\n", floatPtr) ;
printf("doublePtr value : %u", doublePtr) ;
getch() ;
}
```

intPtr value : 1012

floatPtr value : 2008

doublePtr value : 3040

# Subtraction Example

```
void main()
{
    int a, *intPtr ;
    float b, *floatPtr ;
    double c, *doublePtr ;
    clrscr() ;
    intPtr = &a ; // Asume address of a is 1000
    floatPtr = &b ; // Asume address of b is 2000
    doublePtr = &c ; // Asume address of c is 3000

    intPtr = intPtr - 3 ; // intPtr = 1000 - ( 3 * 4 )
    floatPtr = floatPtr - 2 ; // floatPtr = 2000 - ( 2 * 4 )
    doublePtr = doublePtr - 5 ; // doublePtr = 3000 - ( 5 * 8 )
```

```
printf("intPtr value : %u\n", intPtr) ;
printf("floatPtr value : %u\n", floatPtr) ;
printf("doublePtr value : %u", doublePtr) ;
getch() ;
}
```

intPtr value : 988

floatPtr value : 1992

doublePtr value : 2960



# Increment & Decrement Operation on Pointer

➤ The increment operation on pointer variable is calculated as follows...

$$\blacksquare \text{ AddressAtPointer} + \text{NumberOfBytesRequiresByDatatype}$$

➤ The decrement operation on pointer variable is calculated as follows...

$$\blacksquare \text{ AddressAtPointer} - \text{NumberOfBytesRequiresByDatatype}$$

# Increment Example

```
void main()
{
    int a, *intPtr ;
    float b, *floatPtr ;
    double c, *doublePtr ;
    clrscr() ;
    intPtr = &a ; // Asume address of a is 1000
    floatPtr = &b ; // Asume address of b is 2000
    doublePtr = &c ; // Asume address of c is 3000

    intPtr++ ; // intPtr = 1000 + 4
    floatPtr++ ; // floatPtr = 2000 + 4
    doublePtr++ ; // doublePtr = 3000 + 8
```

```
printf("intPtr value : %u\n", intPtr) ;
printf("floatPtr value : %u\n", floatPtr) ;
printf("doublePtr value : %u", doublePtr) ;
    getch() ;
}
```

```
intPtr value : 1004
floatPtr value : 2004
doublePtr value : 3008
```

# Decrement Example

```
void main()
{
    int a, *intPtr ;
    float b, *floatPtr ;
    double c, *doublePtr ;
    clrscr() ;
    intPtr = &a ; // Asume address of a is 1000
    floatPtr = &b ; // Asume address of b is 2000
    doublePtr = &c ; // Asume address of c is 3000

    intPtr++ ; // intPtr = 1000 - 4
    floatPtr++ ; // floatPtr = 2000 - 4
    doublePtr++ ; // doublePtr = 3000 - 8
```

```
printf("intPtr value : %u\n", intPtr) ;
printf("floatPtr value : %u\n", floatPtr) ;
printf("doublePtr value : %u", doublePtr) ;
    getch() ;
}
```

```
intPtr value : 996
floatPtr value : 1996
doublePtr value : 2992
```

# Comparison of Pointers

- The comparison operation is performed between the pointers of same datatype only.
- In C programming language, we can use all comparison operators (relational operators) with pointers.
- **We can't perform multiplication and division operations on pointers.**

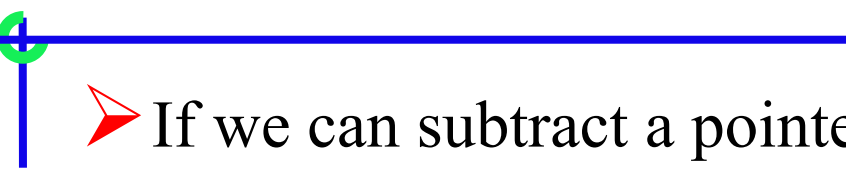
- 
- Consider two pointers ptr1 and ptr2 which point to the same type of data.

- `<datatype> *ptr1, *ptr2;`

- Adding two pointers together is not allowed!
- `ptr1 - ptr 2` is allowed, as long as they are pointing to elements of the same array. The result is

$$\frac{\text{ptr1} - \text{ptr2}}{\text{sizeof}(\text{datatype})}$$

- In other settings, this operation is undefined (may or may not give the correct answer).




➤ If we can subtract a pointer from another, all the relational operations can be supported!

➤ Logical Operations on Pointers

- $\text{ptr1} > \text{ptr2}$  is the same as  $\text{ptr1} - \text{ptr2} > 0$ ,
- $\text{ptr1} = \text{ptr2}$  is the same as  $\text{ptr1} - \text{ptr2} = 0$ ,
- $\text{ptr1} < \text{ptr2}$  is the same as  $\text{ptr1} - \text{ptr2} < 0$ ,

# Pointers to Pointers in C

- In c programming language, we have pointers to store the address of variables of any datatype. A pointer variable can store the address of normal variable.
- C programming language also provides pointer variable to store the address of another pointer variable. This type of pointer variable is called as pointer to pointer variable.
- Sometimes we also call it as double pointer. We use the following syntax for creating pointer to pointer...
  - **datatype \*\*pointerName ;**



---

## ➤ Example

- `int **ptr;`


Here, **ptr** is an integer pointer variable that stores the address of another integer pointer variable but does not store the normal integer variable address.

## ➤ Points to be Remembered

- To store the address of normal variable we use single pointer variable
- To store the address of single pointer variable we use double pointer variable
- To store the address of double pointer variable we use triple pointer variable
- Similarly the same for remaining pointer variables also...




# Example



```
void main()
{
    int a ;
    int *ptr1 ;
    int **ptr2 ;
    int ***ptr3 ;
    clrscr() ;
    ptr1 = &x ;
    ptr2 = &ptr1 ;
    ptr3 = &ptr2 ;
    printf("Address of 'x' = %u\n", ptr1) ;
    printf("Address of 'ptr1' = %u\n", ptr2) ;
    printf("Address of 'ptr2' = %u\n", ptr3) ;
    getch() ;
}
```

# Exercise




---

```
main()
{
    int i=-5; j=-2;
    junk(i, &j);
    printf("\n i=%d j=%d", i,j);
}
```

## Output

i=-5 j=4

```
junk(int i, int *j)
{
    i=i*i;
    *j=(*j)*(*j);
}
```



---

```
main()
{
char c, *cc;
int i;
long l;
float f;
c = 'Z';
i = 15;
l = 77777;
f = 3.14;
cc = &c;
printf("\n c = %c  cc = %u", *cc, cc);
cc=&i;
```

```
printf("\n i = %d  cc = %u", *cc, cc);
cc=&l;
printf("\n l = %ld  cc = %u", *cc, cc);
cc=&f;
printf("\n f = %f  cc = %u", *cc, cc);
}
```



---

main()

{

int a, \*b, \*\*c, \*\*\*d, \*\*\*\*e;

a = 10;

b = &a;

c = &b;


d = &c;

e = &d;

printf("\n a=%d b=%u c=%u d=%u e=%u", a,b,c,d,e);

printf("\n %d %d %d, a, a+\*b, \*\*c+\*\*\*d+\*\*\*\*e);

}



---

```
main()
{
char *c = 4000;
int *i = 4000;
long *l = 4000;
float *f = 4000;
double *d = 4000;
printf("\n c=%u, c+1=%u", c,c+1);
printf("\n i=%u, i+1=%u", i,i+1);
printf("\n l=%u, l+1=%u", l,l+1);
printf("\n f=%u, f+1=%u", f,i+1);
printf("\n d=%u, d+1=%u", d,d+1);
}
```