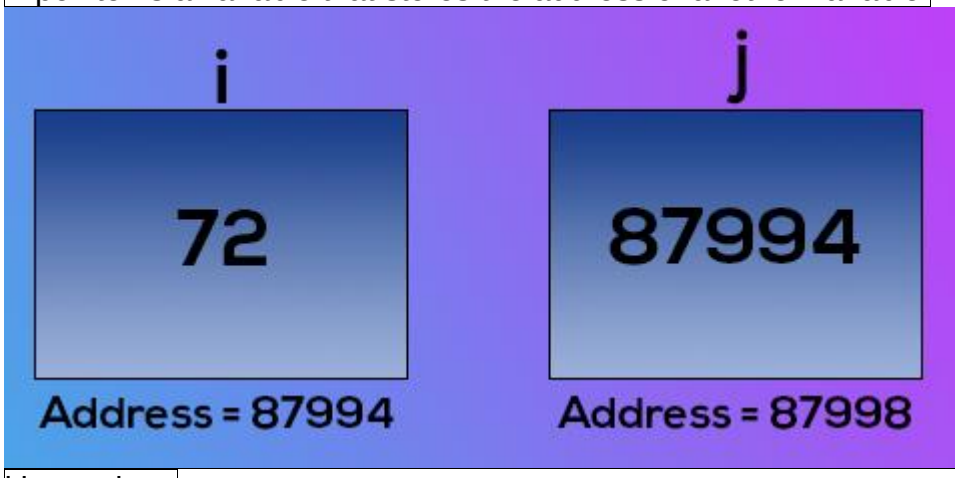
Chapter 6 - Pointers

A pointer is a variable that stores the address of another variable.



j is a pointer
j points to i.

The "address of" (&) operator

The address of operator is used to obtain the address of a given variable
If you refer to the diagrams above

```
&i=> 87994


&j=>87998
```

Copy

Format specifier for printing pointer address is '%u'

The "value of address" operator (*)

The value at address or * operator is used to obtain the value present at a given memory address. It is denoted by *

```
*(&i) = 72


*(&j) = 87994
```

Copy

How to declare a pointer?

A pointer is declared using the following syntax,

```
int *j;  => declare a variable j of type int-pointer
```

```
j=&i     =>store address of i in j
```

Copy

Just like pointer type integer, we also have pointers to char, float, etc.

```
int *ch_ptr;        -> pointer to integer

char *ch_ptr;       -> pointer to character

float *ch_ptr       -> pointer to float
```

Copy

Although it's a good practice to use meaningful variable names, we should be very careful while reading & working on programs from fellow programmers.

A Program to demonstrate Pointers:

```
#include<stdio.h>
int main()
{
int i=8;
int *j;
j=&i;
printf("Add i=%u\n",&i);
printf("Add i=%u\n",j);
printf("Add j=%u\n",&j);
printf("Value i=%d\n",i);
printf("Value i=%d\n",*(&i));
printf("Value i=%d\n",*j);
return 0;
}
```

Copy

Output:

```
Add i=87994
```

```
Add i=87994

Add j=87998

Value i=8

Value i=8

Value i=8
```

Copy

This program sums it all. If you understand it, you have got the idea of pointers.

Pointers to a pointer:

Just like j is pointing to i or storing the address of i, we can have another variable, k which can store the address of j. What will be the type of k?

```
int **k;

k= &j;
```

Copy

We can even go further one level and create a variable l of type int*** to store the address of k. We mostly use int* and int** sometimes in real-world programs.

Types of function calls

Based on the way we pass arguments to the function, function calls are of two types.

1. Call by value -> sending the values of arguments.
2. Call by reference -> sending the address of arguments

Call by value:

Here the values of the arguments are passed to the function. Consider this example:

```
int c = sum( 3 , 4 );    =>  Assume x=3 and y=4
```

Copy

If sum is defined as sum(int a, int b), the values 3 and 4 are copied to a and b. Now even if we change a and b, nothing happens to the variables x and y.
This is **call by value.**
In C, we usually make a call by value.
Call by reference:
Here the address of the variable is passed to the function as arguments.
Now since the addresses are passed to the function, the function can now modify the value of a variable in calling function using * and & operators. Example:

```
void swap(int *x, int *y)
{
int temp;
temp= *x;
*x = *y;
*y = temp;
}
```

Copy

This function is capable of swapping the values passed to it. If a=3 and b=4 before a call to swap(a,b), a=4 and b=3 after calling swap.

```
int main()
{
int a=3;     // a is 3 and b is 4
int b=4;
swap(a,b)
return 0;    // now a is 4 and b is 3
}
```

Copy

Chapter 6 - Practice Set

1. Write a program to print the address of a variable. Use this address to get the value of this variable.
2. Write a program having a variable i. Print the address of i. Pass this variable to a function and print its address. Are these addresses same? Why?
3. Write a program to change the value of a variable to ten times its current value. Write a function and pass the value by reference.
4. Write a program using a function that calculates the sum and average of two numbers. Use pointers and print the values of sum and average in main().
5. Write a program to print the value of a variable i by using the "pointer to pointer" type of variable.
6. Try problem 3 using call by value and verify that it doesn't change the value of the said variable.

KEEP LEARNING & KEEP PRACTICING :)

Nitin-787