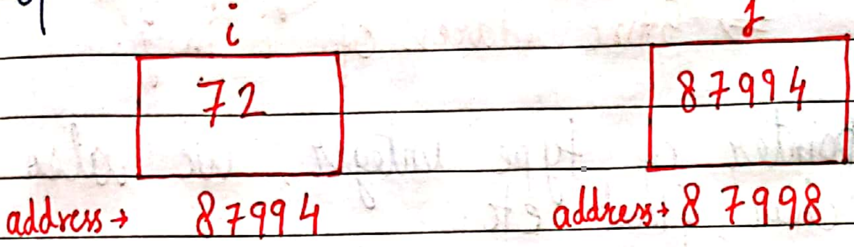
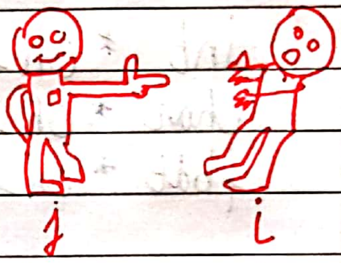
Chapter 6 - Pointers

A pointer is a variable which 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

Format specifier for printing pointer address is '%u'

The 'value at 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

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
```

Just like pointer of 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
```

Although its a good practice to use meaningful variable names, we should be very careful while reading & working on porgrams 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);
   print f (" 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;
}
```

Output:

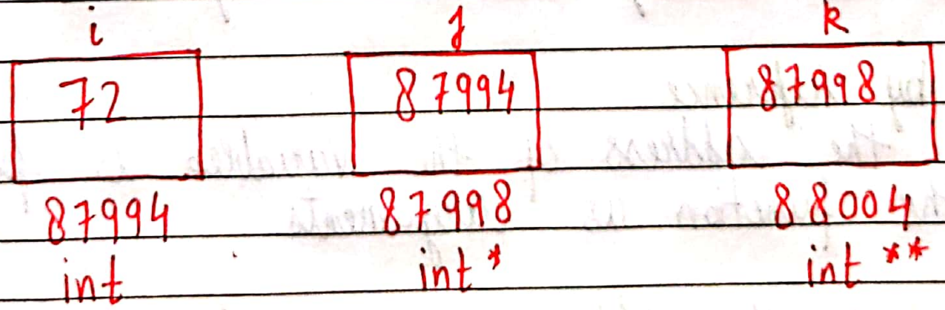
```
Add i = 87994
Add i = 87994
Add j = 87998
Value i = 8
Value i = 8
Value i = 8
```

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

Pointer to a pointer

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

```
int **k;
k = &j;
```



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

int c = sum($\underset{x}{3}$, $\underset{y}{4}$); ⇒ assume x=3 and y=4

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 variables 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;
}
```

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
int b = 4        => a is 3 and b is 4
Swap(a, b)
return 0;        => Now a is 4 and b is 3
}
```