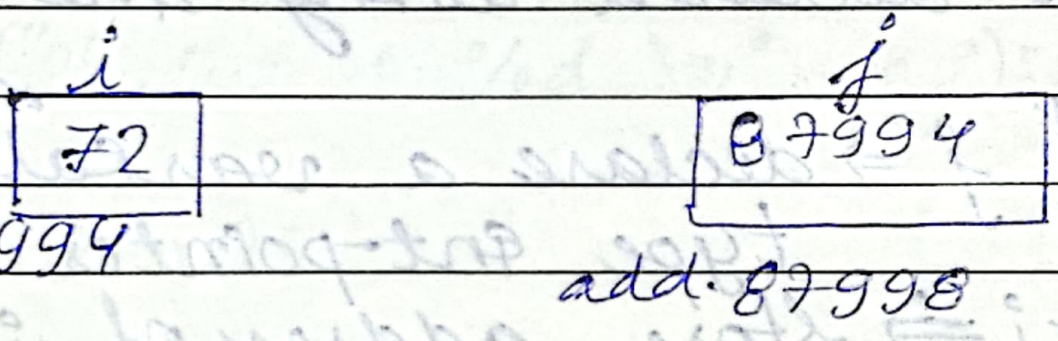
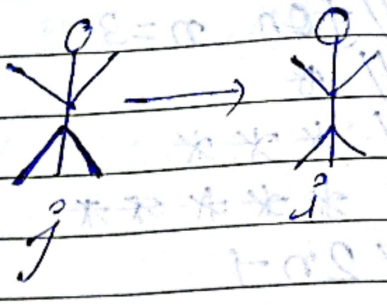
Ch·6 → Pointers

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 dig. above

&i ⇒ 87994
&j ⇒ 87998

Format specifier for printing pointer address is '%u'.

The 'value at address' operator (*)
The value of 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 a pointer of type integer, we also have pointers to char, float etc.

int * ch_ptr; → Pointer to integer

char * ch_ptr; → Pointer to characters

float * ch_ptr; → Pointer to float

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;
}
```

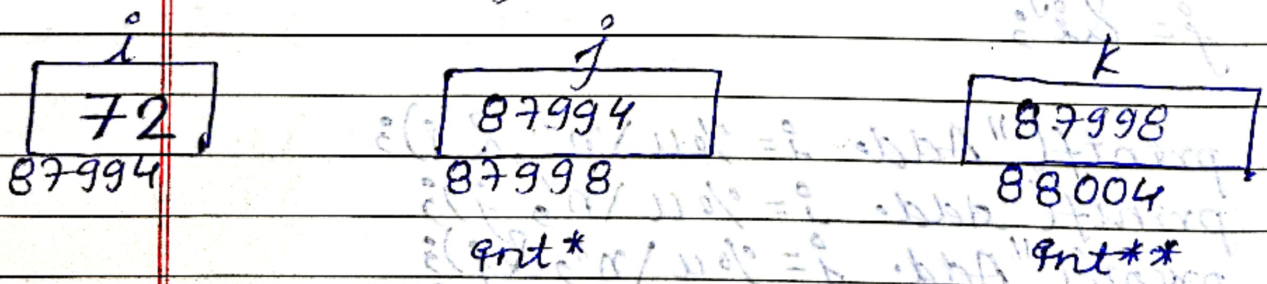
Output
add. p = 87994
add. i = 87994
add. j = 87998
Value j = 8
Value i = 8
Value i = 8

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 & create a variable & of type int*** to store the address of k. We mostly use int* & int** sometimes

Types of Function Calls

Based on the way we pass arguments to the function, function calls are of 2 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 the example

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

If sum is defined as sum(int a, int b), the values 3 & 4 are copied to a & b. Now even if we change a & b, nothing happens to the variables x & y.

In C we usually make a call by value.

Code:

```
#include<stdio.h>
int sum(int a, int b);
int main(){
    int a=4, b=7;
printf("The value of a & b is %d \n", a, b);
printf("The value of 4+7 is %d \n", sum(a,b));
printf("The value of a & b after function
        call is %d & %d \n", a, b);
return 0;
}
```

```
int sum (int a, int b) {
    int c;
    c = a + b;
    b = 3434;
    a = 23432;
    return c;
}
```

Output

2) The value of a & b is 4 & 7.
The value of 4+7 is 11
The value of a & b after function call is 4 & 7

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 * & & operators.

Code:

```
#include <stdio.h>
void wrong_swap (int a, int b);
void swap (int *a, int *b);

int main () {
    int x = 3, y = 4;
printf("The value of x & y before swap is %d & %d \n", x, y);
```

```
//wrong_swap(x,y); /* will not work due to call by value */
swap(&x, &y); /* will work due to call by reference */
printf("The value of x & y after swap is %d & %d\n", x,y);
return 0;
}
```

Output

The value of x & y before swap is 3 & 4

The value of x & y after swap is 4 & 3.

```
void wrong_swap(int a, int b){
    int temp;
    temp=a;
    a=b;
    b=temp;
}

void swap(int *a, int *b){
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Output:
The value of x & y before swap is 3 & 4.
The value of x & y after swap is 4 & 3.

Function for swapping:

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

This function is capable of swapping the value passed to it. If a=3 & b=4 before a call to swap(a, b), a=4 & 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 & b is 3
}
```

Practice Set 6

Q1) Write a program to print the address of a variable. Use this address to get the value of this variable

```
#include <stdio.h>
int main() {
  int a=6;
```

```
int *ptr;
ptr = &a;

printf("The value of variable a is %d \n", a);
printf("The value of
printf("The address of variable a is %u \n", ptr);

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

    return 0;
}
```

Output

The value of variable a is 6

The value of

The address of variable a is 6422216.

The value of variable a is 6.

(9) Write a program having a variable i. Print the address of i. Pass this variable to a function & print its address. Are these addresses same? why?

```
#include <stdio.h>

void printAdd(int a) {
printf("The address of variable a is
                %u \n", &a);
        }
int main () {
        int i = 4;
```

printf("The value of variable i is %d\n", i);

printf(&d(i);
printf("The address of variable i is %u\n", &i);

return 0;
}

Output
The address of variable a is 6422192
The address of variable i is 6422220

Q3) Write a program to change the value of a variable to ten times of its current value. Write a function & pass the value by reference.

Q.4) Write a program using a function which calculates the sum & average of 2 numbers. Use pointers & print the values of sum & average in main().

Code:

```
#include <stdio.h>

void sumAndAvg (int a, int b, int* sum,
                float *avg)
{
    *sum = a+b;
    *avg = (float)(*sum)/2;
}
int main (){
    int i, j, sum;
    float avg;
    i=3;
    j=6;

sumAndAvg (i, j, & sum, & avg);

printf("The value of sum is %d \n", sum

printf("The value of avg is %f \n", avg

    return 0;
}
```

Output:

The value of sum is 9.

The value of avg is 4.5

Q.9 Write a program to print the value of a variable by using "pointer to pointer" type of variable.

Code: #include <stdio.h>

```
int main (){
    int i = 345;
    int* ptr = &i;
    int** ptr_ptr = &ptr;

    ptr = &i;
    ptr_ptr = &ptr;

    printf("The value of i is %d", **ptr_ptr);

    return 0;
}
```

Output

The value of i is 345