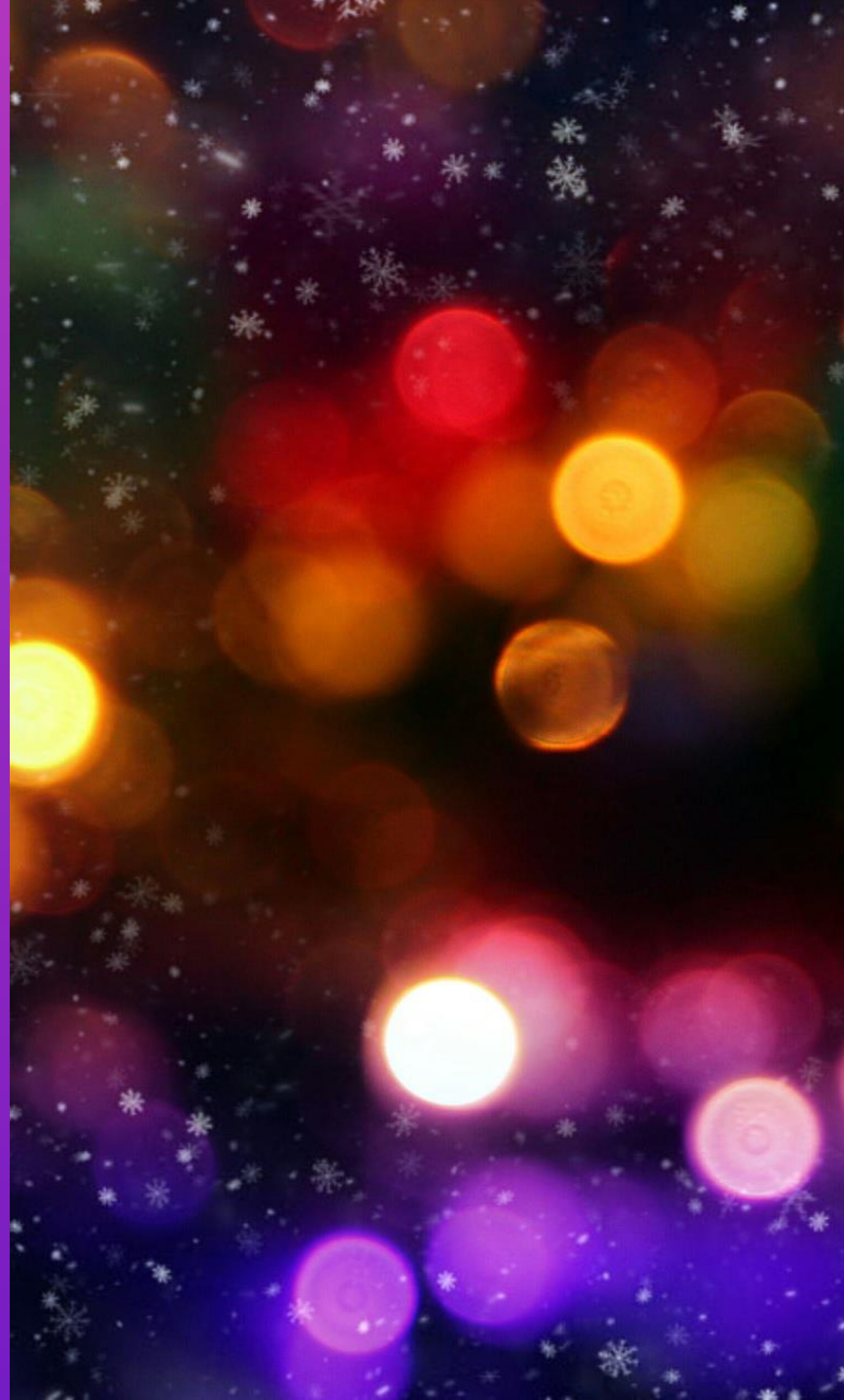
MNNIT COMPUTER CODING CLUB

CLASS-3

BASICS OF C



OPERATORS

Arithmetic	+, -, *, /, %
Assignment	=
Increment/Decrement	++, --
Relational	<, >, <=, >=, ==, !=
Logical	&&, ||, !
Conditional	? :
Comma	,
Bitwise	&, |, ~, <<, >>, ^

OPERATORS: Arithmetic, Assignment

Unary Operator: Single Operand

```c
#include<stdio.h>
int main()
{
    int a;
    a = 10;
    printf("Original Value: %d\n", a);
    a = -a;
    printf("After Applying Unary Operator: %d\n", a);
    a = -a;
    printf("After Applying Unary Operator Again: %d\n", a);
}
```

Binary Operator: Two Operand

- Modulo only with integers

```c
#include<stdio.h>
int main()
{
    int a, b, sum, diff, mul, quot, rem;
    scanf("%d %d", &a, &b);
    sum = a + b;
    diff = a - b;
    mul = a * b;
    quot = a / b;
    rem = a % b;
    printf("Sum of %d and %d is %d\n", a, b, sum);
    printf("Difference of %d and %d is %d\n", a, b, diff);
    printf("Multiplication of %d and %d is %d\n", a, b, mul);
    printf("Quotient of %d and %d is %d\n", a, b, quot);
    printf("Remainder of %d and %d is %d\n", a, b, rem);
}
```

INCREMENT/DECREMENT

Prefix: First operate then use

- y=++x;
  - x=x+1;
  - y=x;

- y=--x;
  - x=x-1;
  - y=x;

```c
#include<stdio.h>
int main()
{
    int x=8;
    printf("x = %d\n", x);
    printf("x = %d\n", ++x);
    printf("x = %d\n", x);
    printf("x = %d\n", --x);
}
```

Postfix: First use then operate

- y=x++;
  - y=x;
  - x=x+1;

- y=x--;
  - y=x;
  - x=x-1;

```c
#include<stdio.h>
int main()
{
    int x=8;
    printf("x = %d\n", x);
    printf("x = %d\n", x++);
    printf("x = %d\n", x);
    printf("x = %d\n", x--);
}
```

CONDITIONS(RELATIONAL OP.)

Operator	Meaning
<	less than
<=	less than or equal to
= =	equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to

Let's take a = 9 and b = 5

Expression	Relation	Value of Expression
a < b	False	0
a <= b	False	0
a= =b	False	0
a != b	True	1
a > b	True	1
a >= b	True	1
a = = 0	False	0
b!=0	True	1
a>8	True	1
2 > 4	False	0

LOGICAL OPERATORS

- When we need to combine two or more conditions

Operator	Meaning
&&	AND
||	OR
!	NOT

AND

Condition1	Condition2	Result
False	False	False
False	True	False
True	False	False
True	True	True

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num>0 && num%2==0){
        printf("it is positive even");
    }
    return 0;
}
```

OR

Condition1	Condition2	Result
False	False	False
False	True	True
True	False	True
True	True	True

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num==0 || num > 0){
        printf("it is not negative");
    }
    return 0;
}
```

NOT

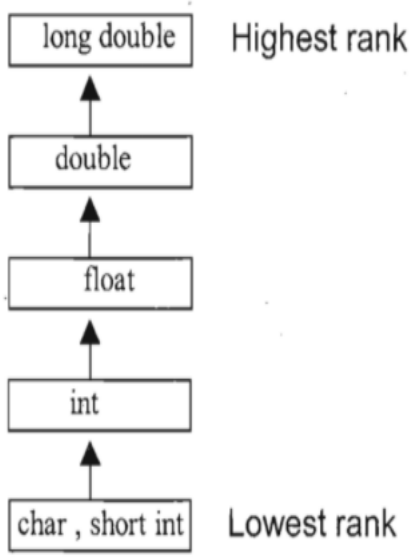
Condition	Result
False	True
True	False

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(!(num==0)){
        printf("it is not 0");
    }
    return 0;
}
```

TYPE CONVERSION

Implicit: Done by C compiler

- In case of operations done between different types of operators, lower rank is automatically converted into higher rank and result is also in higher rank
- In case of assignment LHS operator gets converted into data type of RHS




```c
#include<stdio.h>
int main()
{
    int i1 = 5, i2 = 3;
    float f1 = 2.5, f2 = 3.8;
    i1 = 80.56; //Demotion of LHS into RHS type
    printf("i1 = %d\n", i1);
    f1 = i1 + f2; // i2 promoted to float
    printf("f1 = %f\n", f1);
}
```

Explicit: Done by Programmer

- Also known as type casting or coercion
- Cast operator: Unary Operator
- Syntax: (datatype) expression

```c
#include<stdio.h>
int main()
{
    int x=5, y=2;
    float p, q;
    p = x/y;
    printf("p = %f\n", p);
    q = (float)x/y;
    printf("q = %f\n", q);
}
```

(int)20.3	constant 20.3 converted to integer type and fractional part is lost(Result 20)
(float)20/3	constant 20 converted to float type, and then divided by 3 (Result 6.66)
(float)(20/3)	First 20 divided by 3 and then result of whole expression converted to float type(Result 6.00)
(double)(x +y -z)	Result of expression x+y-z is converted to double
(double)x+y-z	First x is converted to double and then used in expression

PRECEDENCE AND ASSOCIATIVITY

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ - - ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

ESCAPE SEQUENCES

```c
#include<stdio.h>

int main()
{
    printf("Hello World\n");
    printf("Hello\tWorld\n");

    printf("\"");
    printf("\n");

    printf("\'");
    printf("\n");

    printf("\\");
    printf("\n");

    printf("\141");
    printf("\n");

    printf("\x61");
    printf("\n");

    return 0;
}
```

Output:

```
Hello World
Hello   World
"
'
\
a
a
```

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

CONTROL STATEMENTS

The control flow statement in a language specify the order in which computations are performed.

As discussed in the previous class, expressions such as

- x = 0;
- i++;
- printf(...);

become a statement when followed by a semicolon.

Braces { and } are used to group declarations and statements together into a compound statement, or block, so that they are syntactically equivalent to a single statement.

C provides two styles of flow control:

- Branching
- Looping

  Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

IF STATEMENT

- The syntax of the `if` statement in C programming is:

```
if (test expression)
{
   // statements to be executed if the test expression is true
}
```

Working

The `if` statement evaluates the test expression inside the parenthesis `()`.

- If the test expression is evaluated to true, statements inside the body of `if` are executed.
- If the test expression is evaluated to false, statements inside the body of `if` are not executed.

IF ELSE STATEMENT

The `if` statement may have an optional `else` block. The syntax of the `if..else` statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

Working

If the test expression is evaluated to true,

- statements inside the body of `if` are executed.
- statements inside the body of `else` are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of `else` are executed
- statements inside the body of `if` are skipped from execution.
- Conditional Operator

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num<0){
        printf("Number entered is negative");
    }else{
        printf("Number entered is non negative");
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int num;
    printf( "Enter a number\n");
    scanf("%d",&num);
    if(num<0){
        printf ("Number entered is negative");
    }
    return 0;
}
```

IF ELSE PROGRAMS

IF ELSE LADDER

When a choice has to be made from more than 2 possibilities, the `if...else` ladder is used. The `if...else` statement executes a block of code depending upon whether the test expression is true or false.

The if...else ladder allows you to check between multiple test expressions and execute different statements.

```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
else if (test expression3) {
   // statement(s)
}
.
.
else {
   // statement(s)
}
```

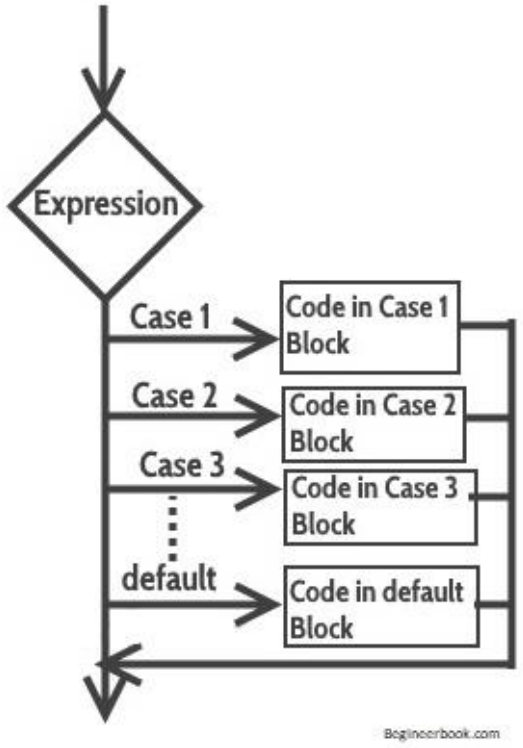
SWITCH CASE

The switch statement allows us to execute one code block among many alternatives.
You can do the same thing with the `if...else..if` ladder. However, the syntax of the `switch` statement is much easier to read and write.

Syntax:

```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
    .
    .
    default:
      // default
statements
}
```



Note:
The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.