LIVEWIRE
FOR LIVE CAREERS

C Programming

LIVEWIRE
FOR LIVE CAREERS

Agenda of This Session

- Introduction to C:
- Introduction, Environment setup, Basic I/O
- Variables &Data types, Constants and Literals
- Storage classes , Type casting, operators
- Conditional Branching control statements: If-Else, switch-case,

A Brief History

- Developed in 1972 (traditional C – K&R C)
- By Dennis Ritchie
- AT At&T bell laboritories
- Influenced by Languages ALGOL, CPL, BCPL, B.
- Developed to make Unix more portable.
- 1989 –Standardised by ANSI (American National Standards Institute) – c89
- ISO (International Organization for standardization) adopted C89 with changes in 1990 – c90
- Later on updated in 1995, 1999, 2011 -C95, C99, c11

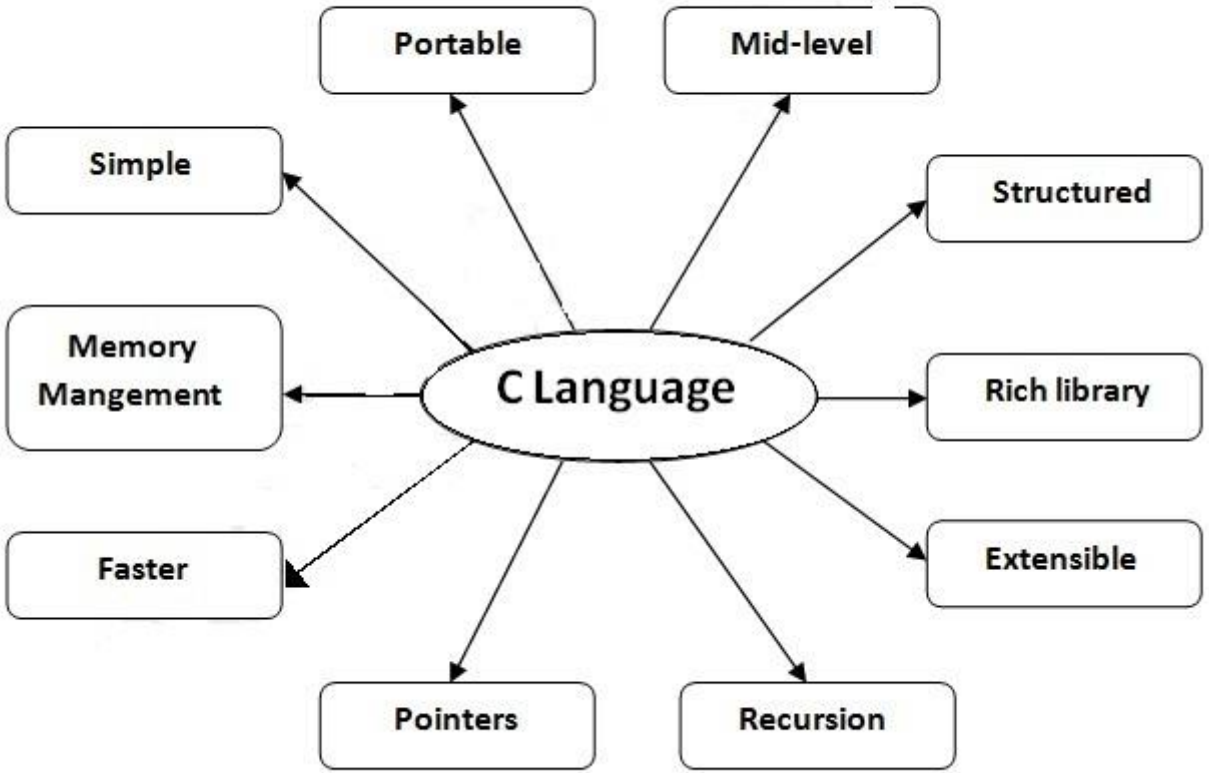
LIVEWIRE
FOR LIVE CAREERS

About C

- Mother language
- System programming language
- Procedure-oriented programming language
- Structured programming language
- Mid level programming language

LIVEWIRE
FOR LIVE CAREERS

Features of C Language

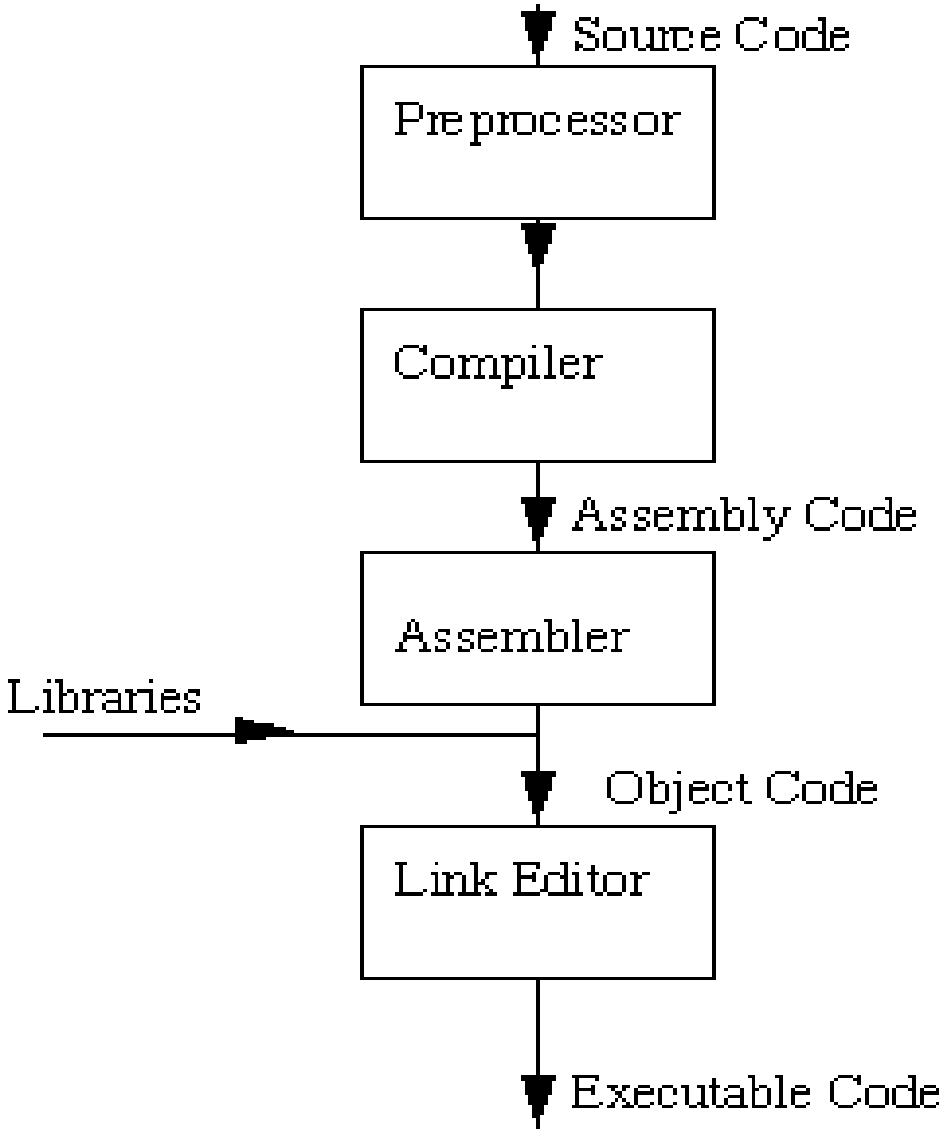


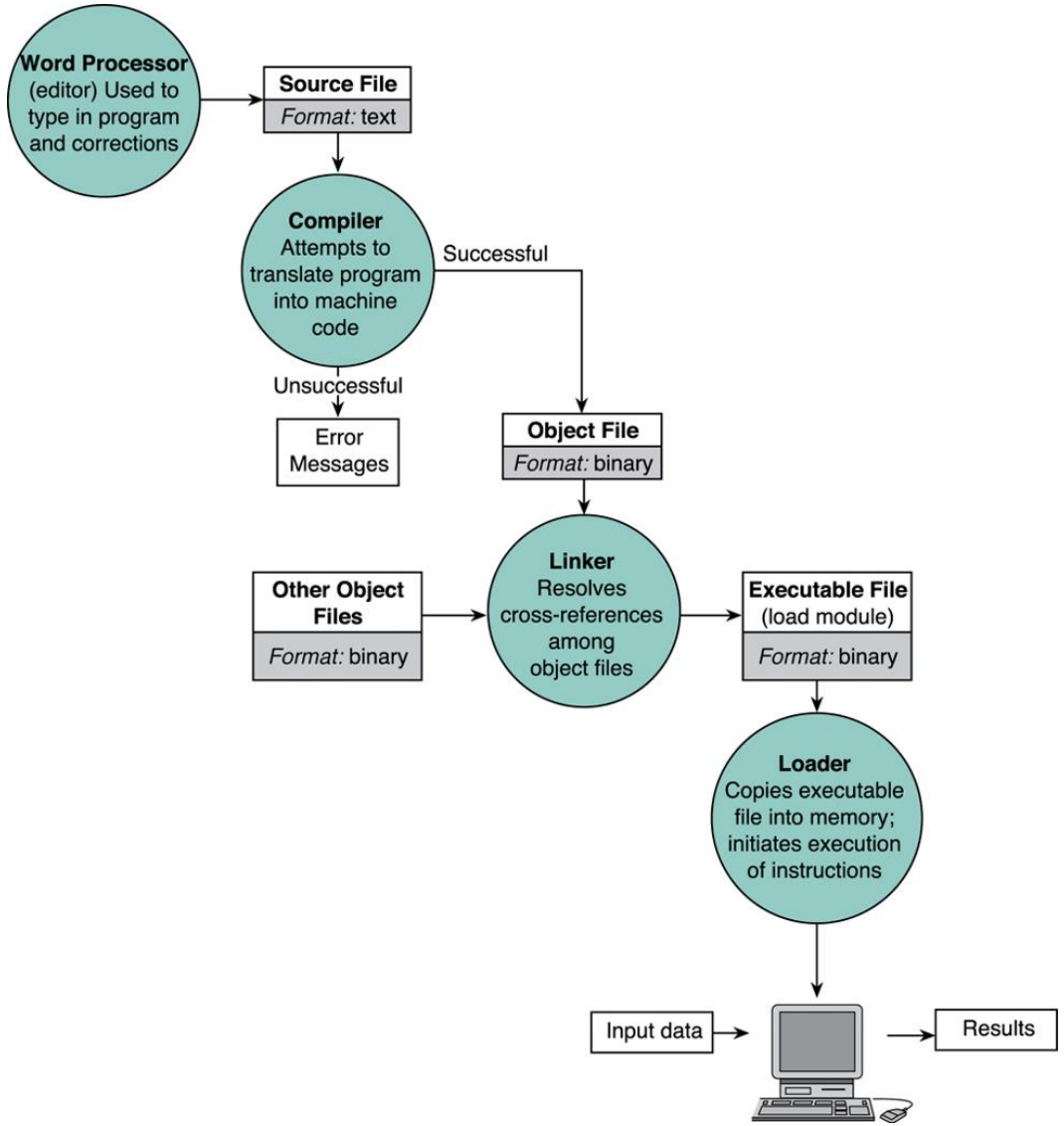
Compiler

- Program that converts Source code to Executable machine language Programs.
- C is a compiled Language.
- Executable machine language program are self contained and runs very quickly.
- Programs written once and run Multiple times, without carrying the source code or compiler around.

LIVEWIRE
FOR LIVE CAREERS

The C Compilation Model




Word Processor
(editor) Used to type in program and corrections
Source File
Format: text
Compiler
Attempts to translate program into machine code
Successful
Unsuccessful
Error Messages
Object File
Format: binary
Other Object Files
Format: binary
Linker
Resolves cross-references among object files
Executable File
(load module)
Format: binary
Loader
Copies executable file into memory; initiates execution of instructions
Input data
Results

RUNNING A C PROGRAM

- Compiler

Eg: gcc, cc, tcc etc

- Integrated Development Environment- IDE (editor, compiler, Debugger)

Eg: visual studio, code blocks, Clion etc

- Online compilers

Eg: codechef.com, jdoodle.com etc

C Program Structure

- Documentations
- Pre-processor Statements
- Global Declarations
- The main() Function
  - Local Declarations
  - Program Statements And expressions
- User Defined Functions

Program Example

Example:

/* description: writes the words "hello world" on the screen */

```
#include<stdio.h>

int main()
{
   /* first c program */
   printf("hello, world!\n");
   return 0;
}
```

Explanation

- /*……*/ - comments
- #include<stdio.h> - pre-processor directive.
- int/void – return value
- main() – main function
- {….} – block
- printf – function to print text to screen
- return 0 – returns value 0 at the end of the main function.

Note

- C requires a semicolon at the end of every statement.
- Printf() is a standard C function
- \n signifies newline.

LIVEWIRE
FOR LIVE CAREERS

Data Types

- Fundamental Datatypes
  - Integer (int)
  - Character (char)
  - Floating Point (float, double)
- User Defined
  - TypeDef
  - Stucture
  - Union
  - Enumerated Datatype

Data types Cont'd

❖Integer

- used to declare numeric program variables of integer type
- whole numbers, positive and negative
- keyword: int
- Example:

```
int number;
number = 12;
```

❖Character

- equivalent to 'letters' in English language
- Numeric digits: 0 – 9
- Lowercase/uppercase letters: a – z and A – Z
- Space (blank)
- Special characters: , . ; ? " / ( ) [ ] { } * & % ^ < > etc
- single character
- keyword: char
- Example:

```
char my_letter;
my_letter = 'U';
```

Data types Cont'd

❖Float

- fractional parts, positive and negative
- keyword: float
- Example:

```
float height;
height = 1.72;
```

❖Double

- used to declare floating point variable of higher precision or higher range of numbers
- exponential numbers, positive and negative
- keyword: double
- Example:

```
double valuebig;
valuebig = 12E-3;
```

Type Modifiers

- Size qualifiers
  - long
  - Short
- Sign qualifiers
  - Signed
  - Unsigned

LIVEWIRE
FOR LIVE CAREERS

Size & Range

Type	Size (bytes)	Range	Format Specifier
Char Signed char	1	-127 to +127	%c
Unsigned char	1	0 to 255	%c
Short Short int Signed short Signed short int	2	−32,767 to +32,767	%hi
Int signed Signed int	2	−32,767 to +32,767	%i or %d
unsigned short unsigned short int	2	0 to 65536	%hu
unsigned unsigned int	2	0 to 65536	%u

Size & Range cont'd

long long int signed long signed long int	4	−2,147,483,647 to +2,147,483,647	%li
unsigned long unsigned long int	4	0 to 4,294,967,295	%lu
long long (since c99) long long int signed long long signed long long int	8	−9,223,372,036,854,775,807 to +9,223,372,036,854,775,807	%lli
unsigned long long (since c99) unsigned long long int	8	0 to 18446744073709551615	%llu

Size & Range cont'd

float	4	3.4E - 38 to 3.4E + 38	%f (promoted automatically to double for printf())
double	8	1.7E - 308 to 1.7E + 308	%f (%F) %g %G %e %E (for scientific notation)
long double	10	3.4E _ 4932 to 1.1E + 4932	%Lf %LF %Lg %LG %Le %LE
_Bool (since c99)	1	0 (false) or 1(true)	

Example Program

```
#include <stdio.h>
int main()
{
   int a = 4000; // positive integer data type
   float b = 5.2324; // float data type
   char c = 'Z'; // char data type
   long d = 41657; // long positive integer data type
   long e = -21556; // long -ve integer data type
   int f = -185; // -ve integer data type
   short g = 130; // short +ve integer data type
   short h = -130; // short -ve integer data type
   double i = 4.1234567890; // double float data type
   float j = -3.55; // float data type
}
```

Variables

- Variables are Names (identifiers) Associated with locations in memory for storing Variable data.
- Every variable has
  - Name
  - Type
  - Size
  - Value

LIVEWIRE
FOR LIVE CAREERS

Rules for naming identifiers

Rules	Example
Can contain a mix of characters and numbers. However it cannot start with a number.	H2O
First character must be a letter or underscore	Temp1; _nummix
Can be of mixed cases including underscore character	Circle_Area; TrigCalc
Cannot contain any arithmetic operators	A/2; S*
Or any other punctuation marks	Num#123; power^2;
Cannot be a C keyword/reserved word	If, printf, while, struct
Cannot contain a space	Decimal Numbers
Identifiers are case sensitive	Temp != temp

Reserved Words

- C89 has 32 reserved words, also known as keywords, which are the words that cannot be used for any purposes other than those for which they are predefined:

Auto	Double	Int	Struct
Break	Else	Long	Switch
Case	Enum	Register	Typedef
Char	Extern	Return	Union
Const	Float	Short	Unsigned
Continue	For	Signed	Void
Default	Goto	Sizeof	Volatile
Do	If	Static	While

Reserved Words Cont'd

- C99 reserved five more words:

_Bool	_Imaginary	Restrict	_Complex	Inline

- C11 reserved seven more words:

_Alignas	_Atomic	_Noreturn	_Thread_local
_Alignof	_Generic	_Static_assert	

Constants and Literals

- Four basic types of constants in C
  - Integer constants
  - Floating-point constants
  - Character constants
  - String constants

Integer Constants

❖Integer constants of 3 different bases: Decimal (Base 10), Hexadecimal (Base 16) or Octal (base 8)

- For decimal literals, no prefix is used.
- Prefix used for hexadecimal: 0x / 0X
- Prefix used for octal: 0

❖Example:

- 123 /* decimal constant*/
- 0x9b /* hexadecimal constant*/
- 0X9c /* hexadecimal constant*/
- 0456 /* octal constant*/

Floating Point Constants

❖Floating point constant has a integer part, a decimal point, a fractional part and may contain an exponential part.

- Decimal point format or
- Exponent format

❖Example:

```
1234.5432       /* Decimal Form */
-100.001        /* signed Decimal Form */
2.37E-3                 /* Exponential Form */
3.14159         /* Legal */
314159E-5L              /* With long suffix */
```

Character Constant

- ❖Character constants hold a single character enclosed in single quotes. Different characters have associated integer values (like 'A' is 65, 'a' is 97 etc.)
- Character constants can hold escape sequences too

❖ Example:

'a'

'b'

1

2

'\n'

'\0'

LIVEWIRE
FOR LIVE CAREERS

Escape Sequence

Description	Escape Sequence
Bell	\a
Backspace	\b
Horizontal tab	\t
Vertical tab	\v
Newline	\n
Form feed	\f
Carriage Return	\r
Quotation mark	\”
Apostrophe/single quotes	\’
Question mark	\?
Backslash	\\
Null	\0

String Constants

❖set of characters enclosed between a pair of double quotes

- A string literal may contain any number of characters including alphanumeric characters, escape characters, graphical characters etc.

❖Example

```
""                  /* Null String. */
"A"                 /* String literal having single characters. */
"ABc12.iyu"         /* String literal with multiple characters. */
"ABd jjuh\n"        /* String with spaces and escape characters. */
```

Declaration of Constants in C

2 Ways:

- Using const keyword in variable declaration.
- Using #define preprocessor directives.

Declaration of Constants in C Cont

Const

- Syntax

const data_type variable_name = Constant;

- Example

const float PI = 3.141;

#define

- Syntax

#define Constant_Identifier Value

- Example:

#define PI 3.141

Storage Classes

- Every variable has a storage class and a scope
- For variables storage class determines
  - Part of memory
  - Scope
  - Visibility
  - Lifetime
- 4 storage classes
  - Automatic variables
  - External variables
  - Static variables
  - Register variables

Automatic Storage Class

- Keyword: auto
- Scope: local
- Visibility: function or block
- Lifetime: function or block
- Any variable declared within a function or block with or without the 'auto' Keyword is by default a variable of the automatic storage class.
- auto is the default storage class for local variables.
- Example:

```
{
        int Count;
        auto int Month;
}
```

External Variable

- Keyword: Extern
- Scope: Global
- Visibility: Entire duration of program execution
- Lifetime: Entire duration of program execution
- extern defines a global variable that is visable to ALL object modules.
- The extern keyword is used before a variable to inform the compiler that this variable is declared somewhere else.
- The extern declaration does not allocate storage for variables.

- Example:

```
#include <stdio.h>
extern int x;
int main()
{
  printf("x: %d\n", x);
}
int x = 10;
```

Static Variables

- Keyword: static
- Scope: global
- Visibility: function or block
- Lifetime: Entire duration of program execution
- A static variable tells the compiler to persist the variable until the end of program.
- Static variables have default initial value zero and initialized only once in their lifetime.

```
void test();   //Function declaration
main()
{
 test();
 test();
 test();
}
void test()
{
 static int a = 0;        //Static variable
 a = a+1;
 printf("%d\t",a);
}
output :
1          2          3
```

Register Variable

- Keyword: register
- Scope: local
- Visibility: function or block
- Lifetime: function or block
- Memory location: CPU registers
- Register should only be used for variables that require quick access – such as counters.

Example:

```
{
  register int  Miles;
}
```

LIVEWIRE
FOR LIVE CAREERS

Type Casting

- Process of converting an entity of one data type to another.
- 2 Types:
  - Implicit Type casting
  - Explicit Type casting

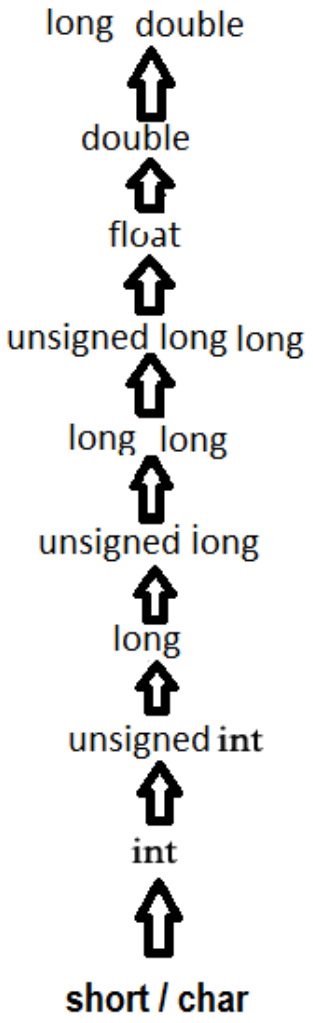
Implicit Type casting

- automatic type conversion by the compiler
- data of one or more subtypes can be converted to a supertype as needed at runtime
- The compiler converts all operands into the data type of the largest operand
- Example:

```
int i=20;

double p;

p=i; // implicit conversion
```



Explicit type conversion

- 2 types
  - Upcasting- lower to upper datatype
  - Downcasting- upper to lower datatype
- Upcast – no data loss
- Downcast – data loss
- Example:
  - float to int causes truncation, i.e., removal of the fractional part.
  - double to float causes rounding of digit.
  - long long to int causes dropping of excess higher order bits.
- Syntax:

(data_type)expression;

```
#include <stdio.h>
int main()
{
   int value1 = 10, value2 = 3;
   float result;

   result = (float)value1 / value2;
   printf("Result with type casting: %f",
result);
   return 0;
}
```

Operators

- Special characters used to perform particular operations in C
- Used to manipulate data and variables
- They are:
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Assignment Operators
  - Relational Operators
  - Logical Operators
  - Conditional Operators
  - Bitwise Operators
  - Special Operators

Operators Cont'd

Arithmetic Operators

Operator	Meaning of Operator
+	Addition
–	Subtraction
*	Multiplication
/	Division
%	Remainder

Relational Operators

Operator	Description
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Operators Cont'd

Assignment Operators

Operator	Description
=	Simple Assignment
+=	Assignment by sum
-=	Assignment by difference
*=	Assignment by product
/=	Assignment by quotient
%=	Assignment by remainder

<<=	Assignment by bitwise left shift
>>=	Assignment by bitwise right shift
&=	Assignment by bitwise AND
^=	Assignment by bitwise XOR
|	Assignment by bitwise OR

LIVEWIRE
FOR LIVE CAREERS

Operators Cont'd

Logical Operators

Operator	Description
&&	Logical AND. True only when all operands are true. (performs logical conjunction on 2 expressions)
||	Logical OR. True if either one operand is true.(Performs a logical disjunction on 2 expressions)
!	Logical NOT. True when operand is 0. (Performs logical negation on an expression)

Bitwise Operators

Operator	Description
<<	Binary Left Shift Operator
>>	Binary Right Shift Operator
~	Binary ones compliment operator
&	Binary AND operator
^	Binary XOR operator
|	Binary OR operator

Operators Cont'd

Increment and Decrement Operators

Operator	Description
++	Increment Operator
--	Decrement Operator

Conditional (ternary) operator

- Syntax :

Conditional Expression ? expression1 : expression2

  - If conditionalExpression is true, expression1 is evaluated.
  - If conditionalExpression is false, expression2 is evaluated.

- Example :

```
(A > 100 ? 0 : 1);
```

Special Operators

- Comma Operator
  - Link related expressions togather
  - Eg: int a, c = 5, d;

- Sizeof Operator
  - returns the size of data
  - Eg:

```
int a,b;
b=sizeof(a);
```

- &- Address of Operator

Eg: &a;

- * - Pointer to a variable

Eg: * a

Control Statements

Control the flow of program

3 types:

- Conditional branching control statement
- Conditional looping control statement
- Unconditional control statement

Branching control statement

- 2 types
- Decision Control Statements

Checks for one or more conditions and if the condition is satisfied i.e. true one or more statements are executed.

  - IF statement
  - IF– ELSE statement
  - IF–ELSE–IF Statement
  - Nested IF and IF–ELSE Statement

- Selection Control Statement

A switch statement decides which of several statements to execute.

  - Switch–Case Statement

IF Statement

- Most simple form of the branching statements.
- Syntax:

```
if (expression)
  statement;


or


if (expression)
  {
    Block of statements;
  }
```

```
#include <stdio.h>
int main()
{
    int x = 20;
    int y = 22;
    if (x<y)
    {
        printf("Variable x is less than y");
    }
    return 0;
}
```

IF-ELSE Statement

- Executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).
- Syntax:

```
if (expression)
  {
   Block of statements;
  }
else
  {
   Block of statements;
  }
```

```
#include <stdio.h>
int main()
{
   int num;
   printf("Enter an integer: ");
   scanf("%d",&num);
   // True if remainder is 0
   if( number%2 == 0 )
      printf("%d is an even integer.",num);
   else
      printf("%d is an odd integer.",num);
   return 0;
}
```

IF-ELSE-IF Statement

- if-else-if statement allows you to check for multiple test expressions and execute different codes for more than two conditions.
- Syntax:

```
if (expression)
  {
   Block of statements;
  }
else if(expression)
  {
   Block of statements;
  }
else
  {
   Block of statements;
  }
```

```
#include <stdio.h>
int main()
{
int var1, var2;
printf("Enter two integers: ");
   scanf("%d %d", &var1, &var2);
if (var1 >var2)
{
   printf("var1 is greater than var2");
}
else if (var2 > var1)
{
   printf("var2 is greater than var1");
}
else
{
   printf("var1 is equal to var2");
}
return 0;
}
```

Nested IF-ELSE Statement

- More than one conditions are checked one after another to execute a specific statement.
- Syntax:

```
If(Expression)
{
      If(Expression 2)
      {
        Block of statement
      }
      else
      {
        Block of statement
      }
}
```

```
#include <stdio.h>
int main()
{
   int n1, n2, n3;
   printf("Enter any three numbers :\n");
   scanf("%d %d %d", &n1, &n2, &n3);
   if(n1>n2)
   {
      if(n1>n3)
         printf("n1 = %d is max.", n1);
      else
         printf("n3 = %d is max.", n3);
   }
   else
   {
      if(n2>n3)
         printf("n2 = %d is max.", n2);
      else
         printf("n3 = %d is max.", n3);
   }
    return 0;
}
```