Structured Programming using C

RCP2SFCES101

Unit-II

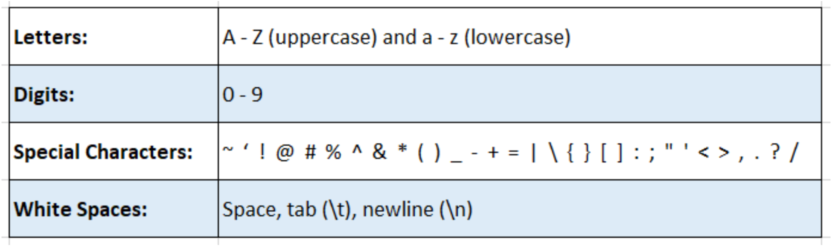
Fundamentals of C-Programming

Contents

Character Set, Identifiers and keywords, Data types, Constants, Variables

Character Set

- The character set in C consists of letters, digits, special characters, and white spaces that the language recognizes
- Character set are used to form tokens, which include keywords, identifiers, constants, string literals, and operators.
- Below are the valid alphabets, numbers and special symbols allowed in C



Letters:	A - Z (uppercase) and a - z (lowercase)
Digits:	0 - 9
Special Characters:	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /
White Spaces:	Space, tab (\t), newline (\n)

Figure: Character set in C

Identifiers in C

- Identifiers are user defined names given to various program elements like variables, functions, arrays, etc.
- **Examples:**

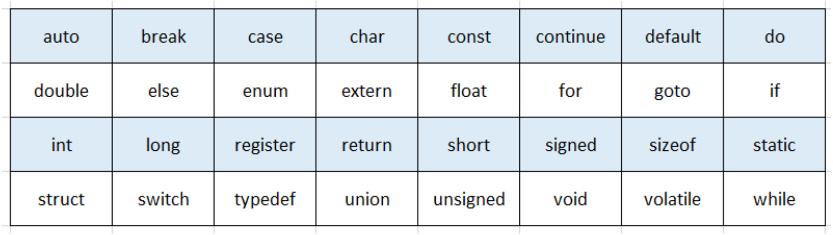
```
int total;
char studentNames[10];
void add(int a, int b)
{

}
```

Rules for naming Identifiers in C

- Identifier may consist of alphabets, digit, underscore
- The first character in the variable name must be an alphabet or underscore
- No commas or blanks are allowed within a variable name.
- No special symbol other than an underscore can be used in a identifier.
- Uppercase and lowercase letters are distinct (Ex. age and Age are two different identifiers)
- It should not be keyword.
- There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.

keywords in C

- Reserved words that have special meaning in C.
- Keywords are the words whose meaning has already been explained to the C compiler
- They **cannot** be used as identifiers (names for variables, functions, etc.).
- There are only 32 keywords available in C

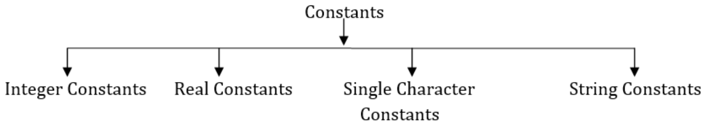


auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Figure: Keywords in C

Constants

- Constants are fixed values that do not change during the execution of a program.
- Following figure shows different types of constant supported by C



Types of Constants

1. **Integer Constant:** An integer constant refers to the sequence of digits.
   Examples:

   123

   -34

   976

   Types of Integer Constants:

   - **Decimal:** Sequence of digits from 0 through 9
     Examples: 10   255
   - **Octal:** Sequence of digits from 0 through 7 starting with 0
     Examples: 075   012
   - **Hexadecimal:** Sequence of digits from 0 through 9 starting with 0x or 0X
     Examples: 0x8A   0X23ab

Types of Constants

2. **Real Constants:** The numbers containing fractional part are called as real constants or floating point constants.
   Examples: 0.75
   -56.67
   .98
3. **Single Character Constants:** A single character constant contains a single character enclosed in single quotes.
   Examples: 'b'
   '*'
   '7'
4. **String Constants:** A sequence of characters enclosed in double quotes.
   Examples: "Hello World"
   "356"
   "5+4"

Variables

- A variable is an identifier(user defined names) for a memory location in which data (integer, real or character constants) can be stored and subsequently recalled.
- Variables are used for holding data values(integer, real or character constants) so that they can be utilized in various computations in a program.
- A variable may take different values at different times during the execution of program.
- Some examples of variable names are:

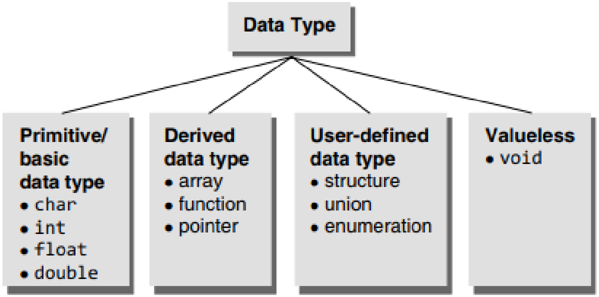
  average

  total

  studentName

Data Types in C

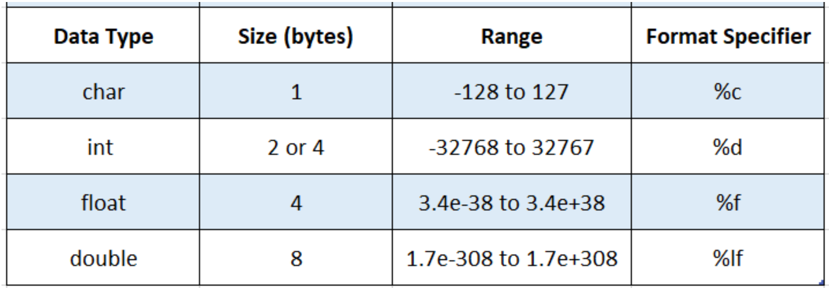
- The data type of a variable determines type of values that a variable might take and the set of operations that can be applied to those values.
- Every data type has its size in the form of the amount of memory.
- Based on the data types, specific amounts of memory and operations are assigned to the variables.
- Data types can be broadly classified as shown in Fig



Data Types in C

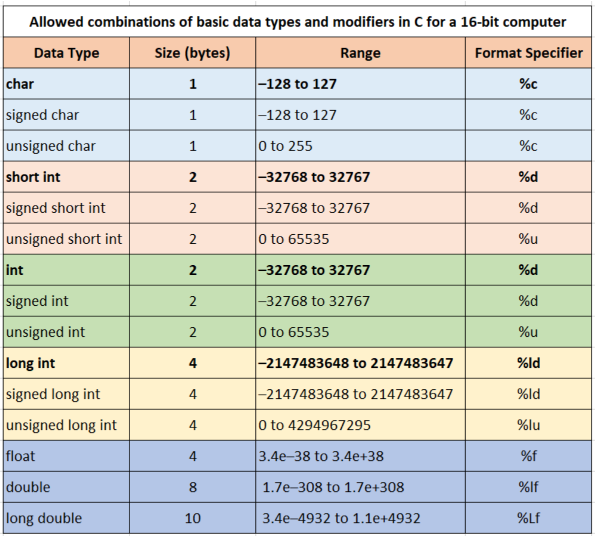
Basic Data Types (Primitive Data Types)

- **char:** It is used to store one character
- **int:** It is used to store integer types of data.
- **float:** It is used to store fractional numbers.
- **double:** It is used to store fractional numbers.
- **void:** It shows no data types. It means no value. Functions that return nothing are usually specified using void data type.



Data Type	Size (bytes)	Range	Format Specifier
char	1	-128 to 127	%c
int	2 or 4	-32768 to 32767	%d
float	4	3.4e-38 to 3.4e+38	%f
double	8	1.7e-308 to 1.7e+308	%lf

Data Types in C



Allowed combinations of basic data types and modifiers in C for a 16-bit computer

Data Type	Size (bytes)	Range	Format Specifier
char	**1**	**–128 to 127**	**%c**
signed char	1	–128 to 127	%c
unsigned char	1	0 to 255	%c
short int	**2**	**–32768 to 32767**	**%d**
signed short int	2	–32768 to 32767	%d
unsigned short int	2	0 to 65535	%u
int	**2**	**–32768 to 32767**	**%d**
signed int	2	–32768 to 32767	%d
unsigned int	2	0 to 65535	%u
long int	**4**	**–2147483648 to 2147483647**	**%ld**
signed long int	4	–2147483648 to 2147483647	%ld
unsigned long int	4	0 to 4294967295	%lu
float	4	3.4e–38 to 3.4e+38	%f
double	8	1.7e–308 to 1.7e+308	%lf
long double	10	3.4e–4932 to 1.1e+4932	%Lf

Variable Declaration and Initialization

Variables Declaration

Syntax:

```
data_type variable1, variable2 . . . variableN;
```

Example:

```
int a, b;
float x, y;
char ch;
```

Variables Initialization

Syntax:

```
variableName = value;
```

Example:

```
a = 10;
x = 3.5f;
ch = ‘S’;
```

It is possible to assign value to the variable at the time of declaration:

Example: int a = 10;

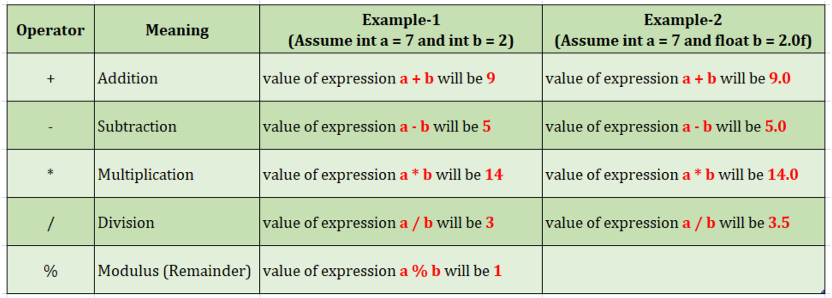
Operators and Expression, statements, Library Functions, Preprocessor

Operators

- Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- Operators are used in program to manipulate data and variables.
- Following are the different types of operators in C:
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Assignment Operator
  5. Unary Operators
  6. Conditional Operator
  7. Bitwise Operators

Arithmetic Operators

- Arithmetic operators are used to construct mathematical expressions.
- C provides five basic arithmetic binary operators.



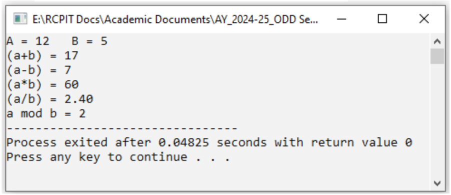
Operator	Meaning	Example-1 (Assume int a = 7 and int b = 2)	Example-2 (Assume int a = 7 and float b = 2.0f)
+	Addition	value of expression **a + b** will be **9**	value of expression **a + b** will be **9.0**
-	Subtraction	value of expression **a - b** will be **5**	value of expression **a - b** will be **5.0**
*	Multiplication	value of expression **a * b** will be **14**	value of expression **a * b** will be **14.0**
/	Division	value of expression **a / b** will be **3**	value of expression **a / b** will be **3.5**
%	Modulus (Remainder)	value of expression **a % b** will be **1**	

Figure: Arithmetic Operators

- In above examples a+b, a-b, a*b, a/b and a%b are arithmetic expressions
  Here a and b may be variables or constants known as operands

Example: Arithmetic Operators

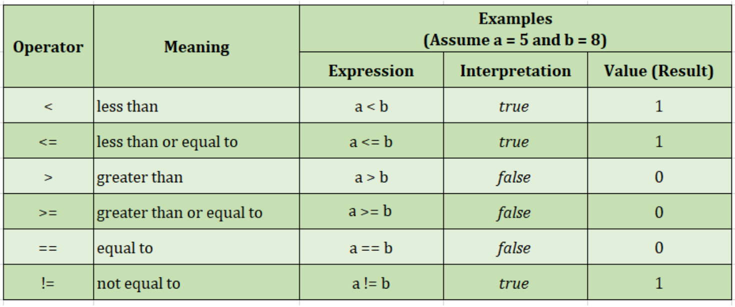
```c
#include<stdio.h>
int main()
{
    int a=12, b=5, c;
    float d;

    printf("A = %d\t B = %d",a,b);

    c = a + b;
    printf("\n(a+b) = %d",c);

    c = a - b;
    printf("\n(a-b) = %d",c);

    c = a * b;
    printf("\n(a*b) = %d",c);

    d = a / (float)b;
    printf("\n(a/b) = %.2f",d);

    c = a % b;
    printf("\na mod b = %d",c);

    return 0;
}
```



```
A = 12   B = 5
(a+b) = 17
(a-b) = 7
(a*b) = 60
(a/b) = 2.40
a mod b = 2
--------------------------------
Process exited after 0.04825 seconds with return value 0
Press any key to continue . . .
```

Relational Operators

- C provides six relational operators for comparing numeric quantities.
- These operators show the relation among the two operands.
- They are also used for decision-making tasks.
- Relational operators check the condition; if it is true, it returns 1, otherwise returns 0

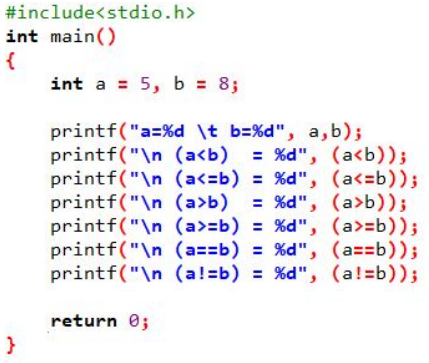


Operator	Meaning	Examples (Assume a = 5 and b = 8)		
		Expression	Interpretation	Value (Result)
<	less than	a < b	*true*	1
<=	less than or equal to	a <= b	*true*	1
>	greater than	a > b	*false*	0
>=	greater than or equal to	a >= b	*false*	0
==	equal to	a == b	*false*	0
!=	not equal to	a != b	*true*	1

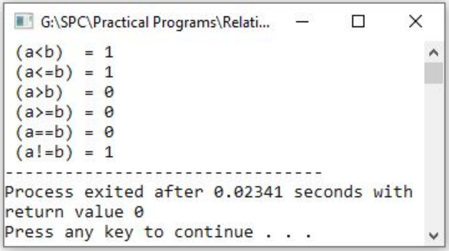
Figure: Relational Operators

Example: Relational Operators

//Program to demonstrate relational operators



```c
#include<stdio.h>
int main()
{
    int a = 5, b = 8;

    printf("a=%d \t b=%d", a,b);
    printf("\n (a<b)  = %d", (a<b));
    printf("\n (a<=b) = %d", (a<=b));
    printf("\n (a>b)  = %d", (a>b));
    printf("\n (a>=b) = %d", (a>=b));
    printf("\n (a==b) = %d", (a==b));
    printf("\n (a!=b) = %d", (a!=b));

    return 0;
}
```

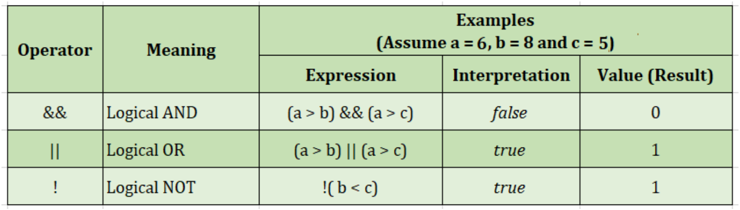


G:\SPC\Practical Programs\Relati...

```
 (a<b)  = 1
 (a<=b) = 1
 (a>b)  = 0
 (a>=b) = 0
 (a==b) = 0
 (a!=b) = 1
--------------------------------
Process exited after 0.02341 seconds with
return value 0
Press any key to continue . . .
```

Logical Operators

- Generally, logical operators are used with relational operators.
- Logical operators are used for combining two statements.
- Logical operators also return the results as true (1) or false (0).



Operator	Meaning	Examples (Assume a = 6, b = 8 and c = 5)		
		Expression	**Interpretation**	**Value (Result)**
&&	Logical AND	(a > b) && (a > c)	*false*	0
||	Logical OR	(a > b) || (a > c)	*true*	1
!	Logical NOT	!(b < c)	*true*	1

Figure: Logical Operators

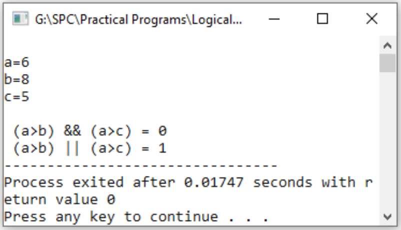
Logical Operators contd...

1. **Logical AND (&&):**
   - The && operator is used to combine two statements (operands)
   - If both statements are true, then AND operator returns 1 (true) else, it returns 0 (false)
2. **Logical OR (||):**
   - The || operator is used to combine two statements (operands)
   - If any one of the statements is true, then the OR operator returns 1 (true) else, it returns 0 (false)
3. **Logical NOT (!):**
   - It is an unary operator. It applied to a single statement.
   - The logical NOT operator complements the results of the statement.
   - It means if the statement is true, it returns the 0 (false) and if the statement is false, it returns the 1 (true)

Example: Logical Operators

//Program to demonstrate logical operators

```
#include<stdio.h>
int main()
{
    int a = 6, b = 8, c = 5, result;

    printf("\na=%d\nb=%d\nc=%d\n", a,b,c);

    result = (a>b) && (a>c);
    printf("\n (a>b) && (a>c) = %d",result);

    result = (a>b) || (a>c);
    printf("\n (a>b) || (a>c) = %d",result);


    return 0;
}
```



G:\SPC\Practical Programs\Logical...

```

a=6
b=8
c=5

 (a>b) && (a>c) = 0
 (a>b) || (a>c) = 1
--------------------------------
Process exited after 0.01747 seconds with r
eturn value 0
Press any key to continue . . .
```

Assignment Operator

- Assignment operator is used to assign value of an expression to the variable.
- When this operator executes, it assigns the value of right operand to the left operand.
- The right operand must be either a constant or a variable, but the left operand must be a variable only
- The symbol of assignment operator is '='.
- **Examples:**

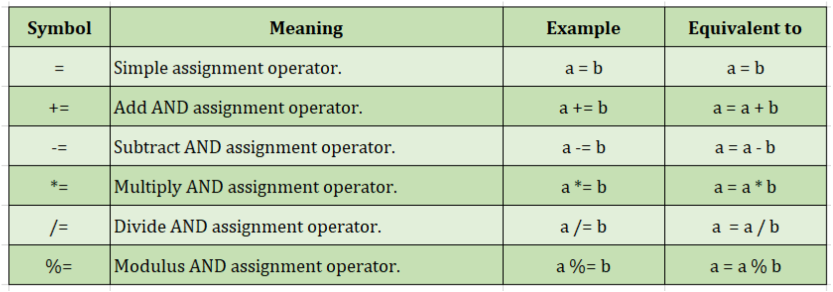
  a = 3

  x = y

  sum = a + b

Assignment Operator contd...

Compound Assignment Operators



Symbol	Meaning	Example	Equivalent to
=	Simple assignment operator.	a = b	a = b
+=	Add AND assignment operator.	a += b	a = a + b
-=	Subtract AND assignment operator.	a -= b	a = a - b
*=	Multiply AND assignment operator.	a *= b	a = a * b
/=	Divide AND assignment operator.	a /= b	a = a / b
%=	Modulus AND assignment operator.	a %= b	a = a % b

Unary Operators

The unary operator requires only one operand or the arguments. Some basic unary operators are as given below.

- Unary minus (-)
- Unary plus (+)
- sizeof() operator
- Addressof operator (&)
- Increment operator (++)
- Decrement operator (- -)

Unary Operators contd...

- **Unary minus (-):** It changes the sign of any operands. It means negative becomes positive or positive becomes negative.

  Example:

  ```
  a = 5;
  b = - a;          it means b = -5
  ```

- **Unary plus (+):** It represents the positive sign of the number.

  Example:

  ```
  a = +5;           It means positive 5 is assigned to variable a
  ```

Unary Operators contd...

- **sizeof() operator:** This operator returns the size of the operand in bytes.

  Example:

  ```
  int a;
  int b = sizeof(a);      Here b is assigned the value 4 (assuming int is 4 bytes)
  ```

- **Addressof operator (&):** Returns the memory address of the operand.

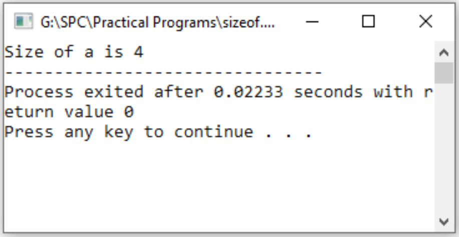
  Example:

  ```
  int a = 5;
  int *p = &a;      Here p is assigned the address of a
  ```

Example: sizeof Operator

//Program to demonstrate sizeof operator

```c
#include<stdio.h>
int main()
{
    int a = 10, b;

    b = sizeof(a);
    printf("Size of a is %d", b);

    return 0;
}
```



```
Size of a is 4
--------------------------------
Process exited after 0.02233 seconds with r
eturn value 0
Press any key to continue . . .
```

Increment and Decrement operator

Increment Operator (++): The operator ++ adds 1 to the operand value

Decrement Operator (- -): The operator – subtracts 1 from the operand value

Both are unary operators and are used in following forms:

Prefix form	Postfix form
++m	m++
--m	m--

Conditional Operator

- The character pair ?: is a conditional operator.
- This operator is used to construct conditional expression of the form:

  v = exp1 ? exp2 : exp3

  Here,

  v is variable

  exp1, exp2 and exp3 are expressions
- **The working of ?: is as follows:** exp1 is evaluated first. If it is true, then expression exp2 is evaluated and its value becomes the value of conditional expression. If exp1 is false, then exp3 is evaluated and its value becomes the value of conditional expression. Either exp2 or exp3 is evaluated.
- **Example:**

  a = 10;

  b = 5;

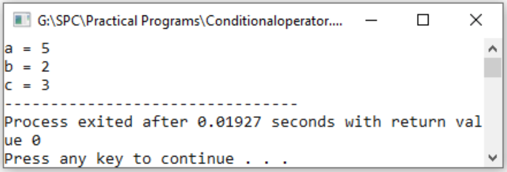
  c = (a > b) ? (a - b) : (a + b);

  After execution of above statement value of variable c will be 5.

Example: Conditional Operators

//Program to demonstrate conditional operator

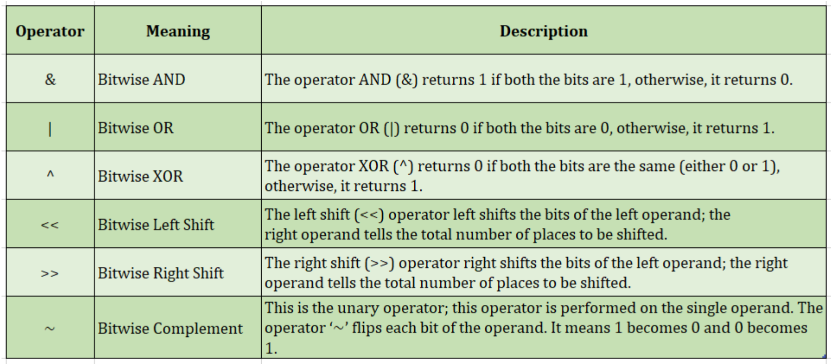
```
#include<stdio.h>
int main()
{
    int a = 5, b = 2, c;

    printf("a = %d \nb = %d",a,b);

    c = (a>b)?(a-b):(a+b);
    printf("\nc = %d", c);

    return 0;
}
```



```
a = 5
b = 2
c = 3
--------------------------------
Process exited after 0.01927 seconds with return val
ue 0
Press any key to continue . . .
```

Bitwise Operator

- The bitwise operators deal with the bit.
- It means that given operands are converted into the binary form and then manipulated bits according to the operations.



Operator	Meaning	Description
&	Bitwise AND	The operator AND (&) returns 1 if both the bits are 1, otherwise, it returns 0.
|	Bitwise OR	The operator OR (|) returns 0 if both the bits are 0, otherwise, it returns 1.
^	Bitwise XOR	The operator XOR (^) returns 0 if both the bits are the same (either 0 or 1), otherwise, it returns 1.
<<	Bitwise Left Shift	The left shift (<<) operator left shifts the bits of the left operand; the right operand tells the total number of places to be shifted.
>>	Bitwise Right Shift	The right shift (>>) operator right shifts the bits of the left operand; the right operand tells the total number of places to be shifted.
~	Bitwise Complement	This is the unary operator; this operator is performed on the single operand. The operator '~' flips each bit of the operand. It means 1 becomes 0 and 0 becomes 1.

Bitwise Operator contd...

Example- Bitwise AND(&):

```
x = 15      : 0000 0000 0000 1111
y = 28      : 0000 0000 0001 1100
z = x & y   : 0000 0000 0000 1100
```

After evaluation of expression z = x & y; the value of variable z will be 12

Example- Bitwise OR(|):

```
x = 15      : 0000 0000 0000 1111
y = 28      : 0000 0000 0001 1100
z = x | y   : 0000 0000 0001 1111
```

After evaluation of expression z = x | y; the value of variable z will be 31

Bitwise Operator contd...

Example- Bitwise XOR(^):

$x = 15$: 0000 0000 0000 1111

$y = 28$: 0000 0000 0001 1100

$z = x \,\hat{}\, y$: 0000 0000 0001 0011

After evaluation of expression $z = x \,\hat{}\, y$; the value of variable z will be 19

Example- Bitwise complement $\sim$:

$x = 88$: 0000 0000 0101 1000

$z = \sim x$: 1111 1111 1010 0111

Bitwise Operator contd...

Example- Bitwise Left Shift (<<):

a = 12 : 0000 0000 0000 1100

b = a << 2 : 0000 0000 0011 00**00**

Vacated positions

In above example, after evaluation of statement b = a << 2; the value of b will be 48

Example- Bitwise Right Shift (>>):

a = 12 : 0000 0000 0000 1100

b = a >> 2 : **00**00 0000 0000 0011

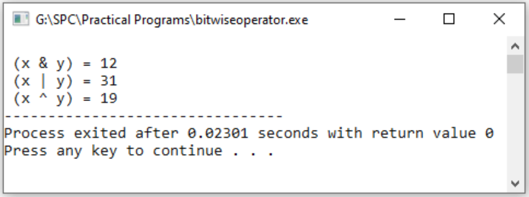
Vacated positions

In above example, after evaluation of statement b = a >> 2; the value of b will be 3

Example: Bitwise Operators

//Program to demonstrate bitwise operators

```
#include<stdio.h>
int main()
{
    int x = 15, y = 28, z;

    z = x & y;
    printf("\n (x & y) = %d", z);

    z = x | y;
    printf("\n (x | y) = %d", z);

    z = x ^ y;
    printf("\n (x ^ y) = %d", z);

    return 0;
}
```

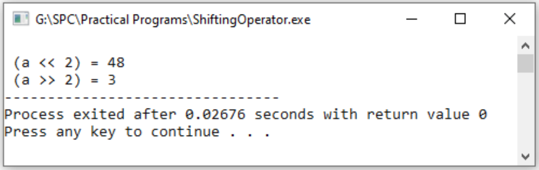

G:\SPC\Practical Programs\bitwiseoperator.exe

```

 (x & y) = 12
 (x | y) = 31
 (x ^ y) = 19
--------------------------------
Process exited after 0.02301 seconds with return value 0
Press any key to continue . . .
```

Example: Bitwise Operators

//Program to demonstrate bitwise shifting operators

```c
#include<stdio.h>
int main()
{
    int a = 12, b;

    b = a << 2;
    printf("\n (a << 2) = %d", b);

    b = a >> 2;
    printf("\n (a >> 2) = %d", b);

    return 0;
}
```



```
 (a << 2) = 48
 (a >> 2) = 3
--------------------------------
Process exited after 0.02676 seconds with return value 0
Press any key to continue . . .
```

Precedence and associativity of C operators

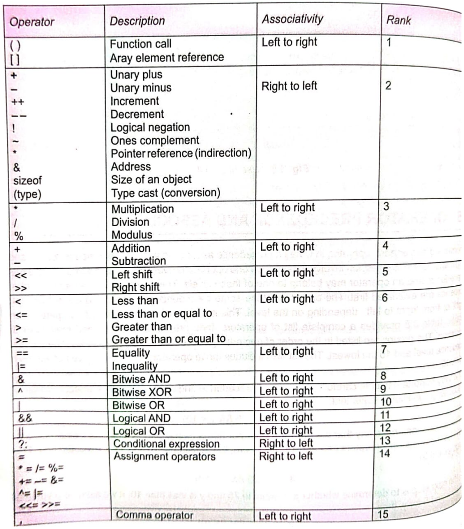
Operator Precedence:

- Operator precedence defines the order in which operators are evaluated in expressions.
- Operators with higher precedence are evaluated before operators with lower precedence.
- If operators have the same precedence, associativity determines the order of evaluation.

Operator Associativity:

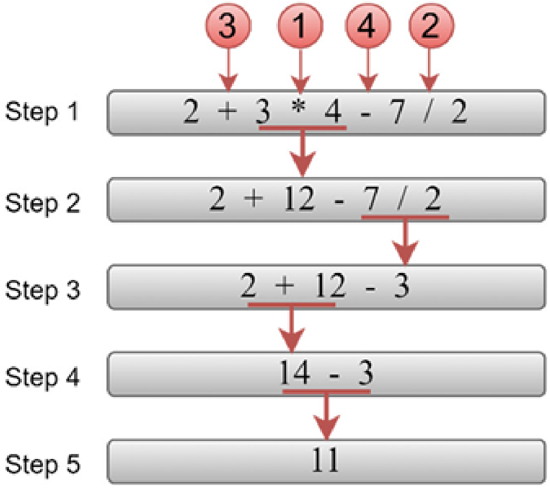
- Associativity defines the direction in which operators of the same precedence are evaluated.
- Associativity can be either:
  1. Left to Right (left-associative)
  2. Right to Left (right-associative)

Precedence and associativity of C operators



Operator	Description	Associativity	Rank
( ) [ ]	Function call Aray element reference	Left to right	1
+ – ++ –– ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference (indirection) Address Size of an object Type cast (conversion)	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ –	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6
== !=	Equality Inequality	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expression	Right to left	13
= *= /= %= += –= &= ^= |= <<= >>=	Assignment operators	Right to left	14
,	Comma operator	Left to right	15

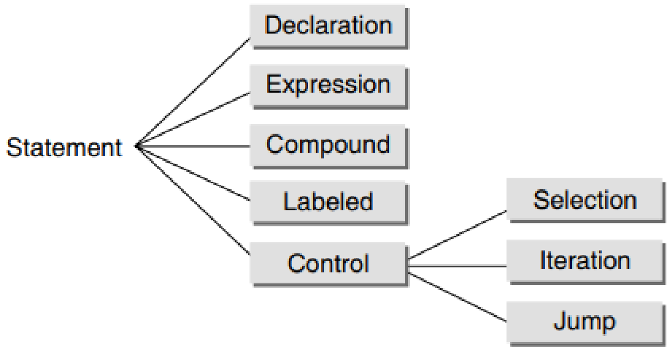
Precedence and associativity of C operators



Evaluation of $x=2+3*4-7/2$

Program Statement

- A statement is a syntactic construction that performs an action when a program is executed.
- All C program statements are terminated with a semicolon (;).
- A program statement in C can be classified as shown in Fig:



Program Statement contd...

1. **Declaration Statements:** is a program statement that serves to communicate to the language translator information about the name and type of the data objects needed during program execution.
   Example:
   int a;

2. **Expression Statements:** These type of statements include any valid C expression followed by a semicolon. They usually involve assignments or function calls.
   Examples:
   int x = 10;
   x = x + 5;
   printf(”Value of x: %d”, x);

Program Statement contd...

3. **Compound Statements:** A compound statement groups multiple statements into a single block, enclosed within curly braces {}.

   Example:

   ```
   {
           int a = 5;
           int b = 10;
           printf("Sum: %d", a + b);
   }
   ```

4. **Labeled Statements:** These provide a label to which the **goto** statement can jump.

   Example:

   ```
   label:
   printf("This is a labeled statement");
   ```

Program Statement contd...

5. **Control Statements:** The control statements determine the 'flow of execution' in a program. Below are different types of Control Statements:

   Selection Statements: allow a program to select a particular execution path from a set of one or more alternatives

   Examples:

   if statement and its different forms

   switch statement

   Iteration Statements: are used to execute a group of one or more statements(block) repeatedly.

   Examples:

   while loop

   do while loop

   for loop

Program Statement contd...

Jump Statements: These statements alter the flow of control in a program by jumping to another part of the program.

Examples:

break

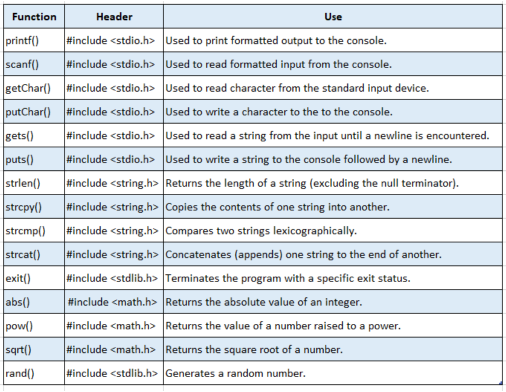
continue

return

goto

Library Functions

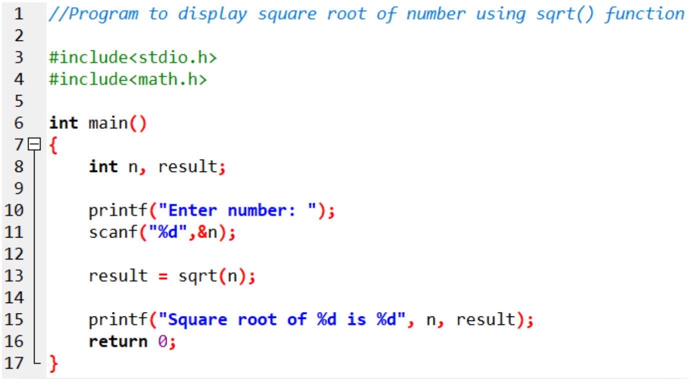
The C language is accompanied by a number of library functions that carry out various commonly used operations or calculations.



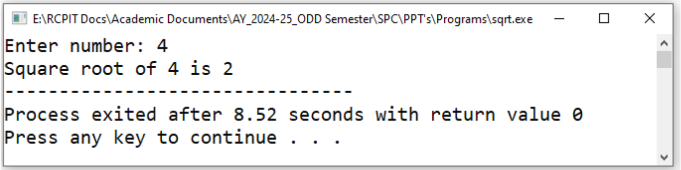
Function	Header	Use
printf()	#include <stdio.h>	Used to print formatted output to the console.
scanf()	#include <stdio.h>	Used to read formatted input from the console.
getChar()	#include <stdio.h>	Used to read character from the standard input device.
putChar()	#include <stdio.h>	Used to write a character to the to the console.
gets()	#include <stdio.h>	Used to read a string from the input until a newline is encountered.
puts()	#include <stdio.h>	Used to write a string to the console followed by a newline.
strlen()	#include <string.h>	Returns the length of a string (excluding the null terminator).
strcpy()	#include <string.h>	Copies the contents of one string into another.
strcmp()	#include <string.h>	Compares two strings lexicographically.
strcat()	#include <string.h>	Concatenates (appends) one string to the end of another.
exit()	#include <stdlib.h>	Terminates the program with a specific exit status.
abs()	#include <math.h>	Returns the absolute value of an integer.
pow()	#include <math.h>	Returns the value of a number raised to a power.
sqrt()	#include <math.h>	Returns the square root of a number.
rand()	#include <stdlib.h>	Generates a random number.

Figure: Commonly used Library Functions

Library Functions Example



```c
//Program to display square root of number using sqrt() function

#include<stdio.h>
#include<math.h>

int main()
{
    int n, result;

    printf("Enter number: ");
    scanf("%d",&n);

    result = sqrt(n);

    printf("Square root of %d is %d", n, result);
    return 0;
}
```



E:\RCPIT Docs\Academic Documents\AY_2024-25_ODD Semester\SPC\PPT's\Programs\sqrt.exe

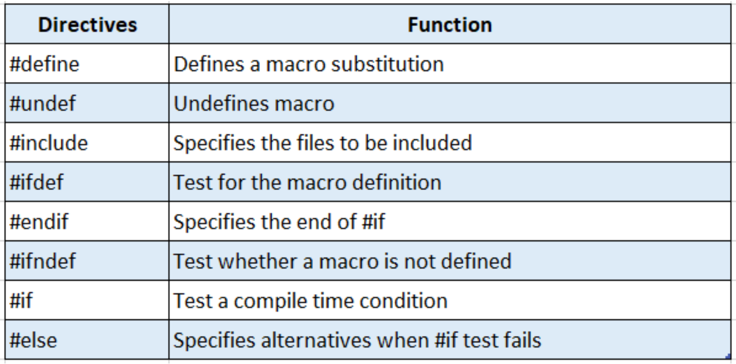
```
Enter number: 4
Square root of 4 is 2
--------------------------------
Process exited after 8.52 seconds with return value 0
Press any key to continue . . .
```

Preprocessor

- A preprocessor is a program that operates on the source code, making modifications or substitutions before it is passed to the compiler.
- It operates under the control of preprocessor directives
- Preprocessor directives are instructions that are processed before the actual compilation of the code begins.
- Preprocessor directives always begin with symbol # (Hash) and are handled by the C preprocessor.
- Preprocessor directives are placed in the source program before the main() line

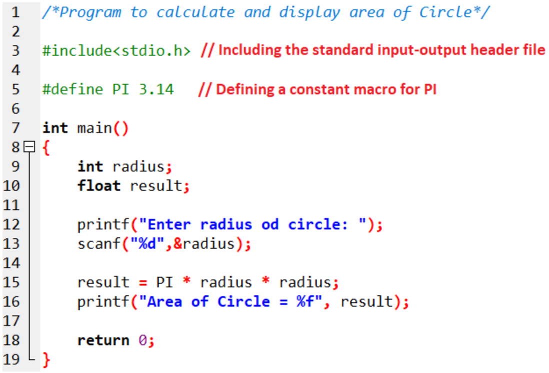
Preprocessor Directives

Below are commonly used preprocessor directives and their functions:



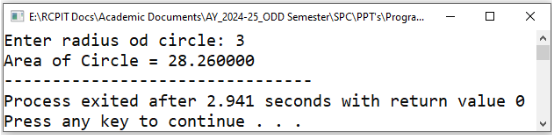
Directives	Function
#define	Defines a macro substitution
#undef	Undefines macro
#include	Specifies the files to be included
#ifdef	Test for the macro definition
#endif	Specifies the end of #if
#ifndef	Test whether a macro is not defined
#if	Test a compile time condition
#else	Specifies alternatives when #if test fails

Preprocessor directive Example



```c
/*Program to calculate and display area of Circle*/

#include<stdio.h>  // Including the standard input-output header file

#define PI 3.14    // Defining a constant macro for PI

int main()
{
    int radius;
    float result;

    printf("Enter radius od circle: ");
    scanf("%d",&radius);

    result = PI * radius * radius;
    printf("Area of Circle = %f", result);

    return 0;
}
```

Output:



E:\RCPIT Docs\Academic Documents\AY_2024-25_ODD Semester\SPC\PPT's\Progra...

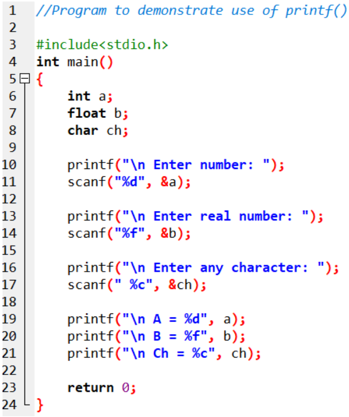
```
Enter radius od circle: 3
Area of Circle = 28.260000
--------------------------------
Process exited after 2.941 seconds with return value 0
Press any key to continue . . .
```

Data Input and Output

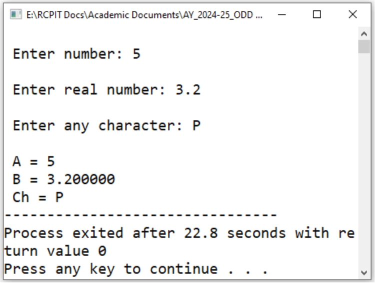
Data Input and Output

- C language provide several built-in functions for performing input/output operations, mainly through the stdio.h library.
- Following are the functions used for standard input and output:
  - **scanf() :** Used to read formatted input from the console.
  - **printf() :** Used to print formatted output to the console.
  - **getchar() :** Used to read character from the standard input device.
  - **putchar() :** Used to write a character to the to the console.
  - **gets() :** Used to read a string from the input until a newline is encountered.
  - **puts() :** Used to write a string to the console followed by a newline.

Example: printf() and scanf() function

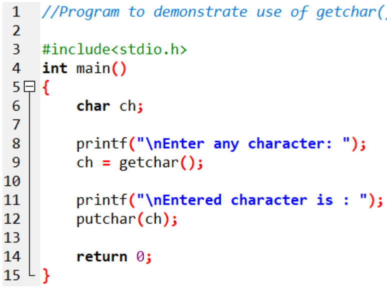


```c
//Program to demonstrate use of printf() and scanf() functions

#include<stdio.h>
int main()
{
    int a;
    float b;
    char ch;

    printf("\n Enter number: ");
    scanf("%d", &a);

    printf("\n Enter real number: ");
    scanf("%f", &b);

    printf("\n Enter any character: ");
    scanf(" %c", &ch);

    printf("\n A = %d", a);
    printf("\n B = %f", b);
    printf("\n Ch = %c", ch);

    return 0;
}
```

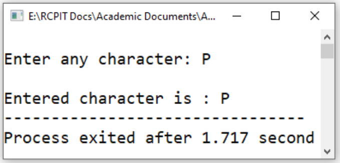


```

 Enter number: 5

 Enter real number: 3.2

 Enter any character: P

 A = 5
 B = 3.200000
 Ch = P
--------------------------------
Process exited after 22.8 seconds with re
turn value 0
Press any key to continue . . .
```

Example: getchar() and putchar() function

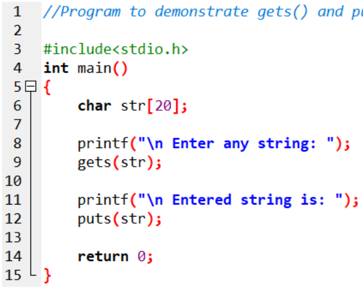


```c
//Program to demonstrate use of getchar() and putchar() functions

#include<stdio.h>
int main()
{
    char ch;

    printf("\nEnter any character: ");
    ch = getchar();

    printf("\nEntered character is : ");
    putchar(ch);

    return 0;
}
```

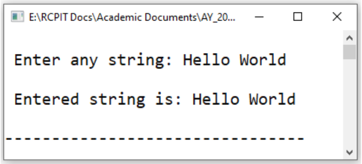


```
Enter any character: P

Entered character is : P
--------------------------------
Process exited after 1.717 second
```

Example: gets() and puts() function



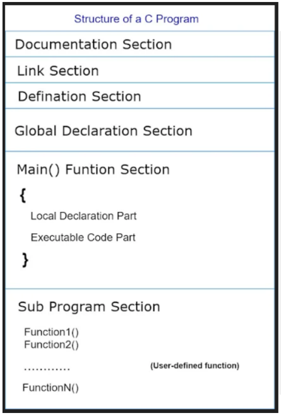
```c
//Program to demonstrate gets() and puts() functions

#include<stdio.h>
int main()
{
    char str[20];

    printf("\n Enter any string: ");
    gets(str);

    printf("\n Entered string is: ");
    puts(str);

    return 0;
}
```

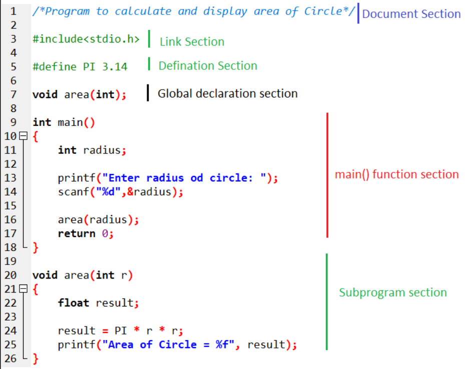


```
Enter any string: Hello World

Entered string is: Hello World

--------------------------------
```

Structure of C program

Structure of C program





```c
/*Program to calculate and display area of Circle*/

#include<stdio.h>

#define PI 3.14

void area(int);

int main()
{
    int radius;

    printf("Enter radius od circle: ");
    scanf("%d",&radius);

    area(radius);
    return 0;
}

void area(int r)
{
    float result;

    result = PI * r * r;
    printf("Area of Circle = %f", result);
}
```

References

- ”MASTERING C” by K.R.Venugopal and Sudeep R.Prasad , Tata McGraw-Hill Publications
- “Programming in ANSI C”, by E. Balaguruswamy, Tata McGraw-Hill Education.
- “Let Us C”, by Yashwant Kanetkar, BPB Publication.
- “Programming in C”, by Pradeep Day and Manas Gosh, Oxford University Press.
- https://www.javatpoint.com/c-programming-language-tutorial
- https://www.tutorialspoint.com/cprogramming/index.htm
- https://www.geeksforgeeks.org/c-programming-language/
- https://www.youtube.com/playlist?list=PLdo5W4Nhv31a8UcMN9-35ghv8qyFWD9_S

Thank you