PROGRAMMING IN C

Data Types and Qualifiers

Receiving Input

This can be achieved using a function called scanf( ). This function is a counter-part of the printf( ) function.

printf( ) outputs the values to the screen whereas scanf( ) receives them from the keyboard.

```
/* Calculation of simple interest */
/* Author gekay Date 25/06/2016 */
# include <stdio.h>
int main( )
{
int p, n ;
float r, si ;
printf ( "Enter values of p, n, r" ) ;
scanf ( "%d %d %f", &p, &n, &r ) ;
si = p * n * r / 100 ;
printf ( "%f\n" , si ) ;
return 0 ;
}
```

Receiving Input

- Note the use of ampersand (**&**) before the variables in the **scanf( )** function is a must.
- **&** is an ‘Address of’ operator.
- It gives the location number (address) used by the variable in memory.
- When we say **&a**, we are telling **scanf( )** at which memory location should it store the value supplied by the user from the keyboard.

Comments in a C Program

a) Comment about the program should be enclosed within /* */ Thus, the first two statements in our program are comments.

(b) Sometimes it is not very obvious as to what a particular statement in a program accomplishes. At such times it is worthwhile mentioning the purpose of the statement (or a set of statements) using a comment. For example:

```
/* formula for simple interest */
si = p * n * r / 100 ;
```

Comments in a C Program

(c) Any number of comments can be written at any place in the program. For example, a comment can be written before the statement, after the statement or within the statement as shown below.

```
/* formula */ si = p * n * r / 100 ;
si = p * n * r / 100 ; /* formula */
si = p * n * r / /* formula */ 100 ;
```

(d) Comments cannot be nested. This means one comment cannot be written inside another comment. For example,

```
/* Cal of SI /* Author: gekay date: 25/06/2016 */ */
```

is invalid.

Comments in a C Program

(e) A comment can be split over more than one line, as in,

```
/* This comment has
three lines
in it */
```

Such a comment is often called a multi-line comment.

What is main( )

(a) **main( )** is a function. A function is nothing but a container for a set of statements. In a C program there can be multiple functions. To begin with, we would concentrate only on those programs which have only one function. The name of this function has to be **main( )**, it cannot be anything else. All statements that belong to **main( )** are enclosed within a pair of braces { } as shown below.

```
int main( )
{
statement 1 ;
statement 2 ;
statement 3 ;
}
```

What is main( )

(b) The way functions in a calculator return a value, similarly, functions in C also return a value. **main( )** function always returns an integer value, hence there is an **int** before **main( )**.

Formatted I/O Functions

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

List of some format specifiers-

S NO.	**Format Specifier**	**Type**	**Description**
1	%d	int/signed int	used for I/O signed integer value
2	%c	char	Used for I/O character value
3	%f	float	Used for I/O decimal floating-point value

Formatted I/O Functions

- printf()   scanf()   sprintf()   sscanf()
- printf() function is used in a C program to display any value like float, integer, character, string, etc on the console screen. It is a pre-defined function that is already declared in the stdio.h(header file)
- scanf() function is used in the C program for reading or taking any value from the keyboard by the user, these values can be of any data type like integer, float, character, string, and many more. This function is declared in stdio.h(header file), that’s why it is also a pre-defined function.

Formatted I/O Functions

- printf()   scanf()   sprintf() sscanf()
- **sprintf()** stands for **“string print”**. This function is similar to printf() function but this function prints the string into a character array instead of printing it on the console screen

```
#include <stdio.h>
int main()
{
   char str[50];
   int a = 2, b = 8;
    // The string "2 and 8 are even number"
   // is now stored into str
   sprintf(str, "%d and %d are even number",
          a, b);
   // Displays the string
   printf("%s", str);
   return 0;
}
```

Formatted I/O Functions

- printf() scanf() sprintf() sscanf()

- **sscanf():**
- sscanf stands for **“string scanf”.** This function is similar to scanf() function but this function reads data from the string or character array instead of the console screen.

Unformatted Input/Output functions

Unformatted I/O functions are used only for character data type or character array/string and cannot be used for any other datatype. These functions are used to read single input from the user at the console and it allows to display the value at the console

These functions are called unformatted I/O functions because we cannot use format specifiers in these functions and hence, cannot format these functions according to our needs.

- getch()
- getche()
- getchar()
- putchar()
- gets()
- puts()
- putch()

Unformatted Input/Output functions

- **getch():**
- getch() function reads a single character from the keyboard by the user but doesn’t display that character on the console screen and immediately returned without pressing enter key. This function is declared in conio.h(header file). getch() is also used for hold the screen

```
#include <conio.h>
#include <stdio.h>
 int main()
{
   printf("Enter any character: ");
     // Reads a character but
   // not displays
   getch();
     return 0;
}
```

Unformatted Input/Output functions

- getche() function reads a single character from the keyboard by the user and displays it on the console screen and immediately returns without pressing the enter key. This function is declared in conio.h(header file).

```
#include <conio.h>
#include <stdio.h>
 int main()
{
   printf("Enter any character: ");
     // Reads a character but
   // displays immediately
   getche();
     return 0;
}
```

Unformatted Input/Output functions

- The getchar() function is used to read only a first single character from the keyboard whether multiple characters is typed by the user and this function reads one character at one time until and unless the enter key is pressed. This function is declared in stdio.h(header file)#include <conio.h>

```
#include <conio.h>
#include <stdio.h>
int main()
{
     char ch;
    printf("Enter the character: ");
    // Taking a character from keyboard
   ch = getchar();
    // Displays the value of ch
   printf("%c", ch);
   return 0;
}
```

Re-Cap

Which of the following are invalid C constants and why?

A.	’3.15’	A.	Invalid. A character constant can contain only 1 character
B.	35,550	B.	Invalid. An integer constant cannot contain a comma
C.	3.25e2	C.	Valid
D.	2e-3	D.	Valid
E.	‘eLearning’	E.	Invalid. A character constant can contain only 1 character
F.	"show“	F.	Invalid. A character constant can contain only 1 character
G.	‘Quest’	G.	Invalid. A character constant can contain only 1 character
H.	2^3	H.	Invalid. Number cannot be expressed in this form
I.	4 6 5 2	I.	Invalid. There cannot be a space within a constant

Which of the following are invalid variable names and why?

A.	B’day	A.	Invalid: No special symbols are allowed.
B.	int	B.	Invalid: keywords cannot be used as a variable name
C.	$hello	C.	Invalid: No special symbols are allowed.
D.	#HASH	D.	Invalid: No special symbols are allowed.
E.	dot.	E.	Invalid: No special symbols are allowed.
F.	number	F.	Valid
G.	totalArea	G.	Valid
H.	_main( )	H.	Invalid: No special symbols are allowed.
I.	temp_in_Deg	I.	Valid _ is allowed
J.	total%	J.	Invalid: No special symbols are allowed.
K.	1st	K.	Invalid: Variable name cannot start with a number
L.	stack-queue	L.	Invalid: No special symbols are allowed.
M.	variable name	M.	Invalid: No white space is allowed
N.	%name%	N.	Invalid: No special symbols are allowed
O.	salary	O.	Valid

State whether the following statements are True or False

(a) C language has been developed by Dennis Ritchie.

(b) Operating systems like Windows, UNIX, Linux and Android are written in C.

(c) C language programs can easily interact with hardware of a PC / Laptop.

(d) A real constant in C can be expressed in both Fractional and Exponential forms.

(e) A character variable can at a time store only one character.

(f) The maximum value that an integer constant can have varies from one compiler to another.

(g) Usually all C statements are written in small case letters.

(i) Spaces may be inserted between two words in a C statement.

(h) Spaces cannot be present within a variable name.

(j) C programs are converted into machine language with the help of a program called Editor. **False**

(k) Most development environments provide an Editor to type a C program and a Compiler to convert it into machine language.

(l) int, char, float, real, integer, character, char, main, printf and scanf all are keywords. **False**

Match the following

A. \n	1) Literal
B. 3.145	2) Statement terminator
C. -6513	3) Character constant
D. ’D’	4) Escape sequence
E. 4.25e-3	5) (e) Input function
F. main( )	6) (f) Function
G. %f, %d, %c	7) (g) Integer constant
H. ;	8) (h) Address of operator
I. Constant	9) Output function
J. Variable	10) Format specifier
K. &	11) Exponential form
L. printf( )	12) Real constant
M. scanf( )	13) Identifier

Match the following

(a) \n	(4) Escape sequence
(b) 3.145	(12) Real constant
(c) -6513	(7) Integer constant
(d) ’D’	(3) Character constant
(e) 4.25e-3	(11) Exponential form
(f) main( )	(6) Function
(g) %f, %d, %c	(10) Format specifier
(h) ;	(2) Statement terminator
(i) Constant	(1) Literal
(j) Variable	(13) Identifier
(k) &	(8) Address of operator
(l) printf( )	(9) Output function
(m) scanf( )	(5) Input function

Point out the errors, if any, in the following programs

```
int main( )
{
int a ; float b ; int c ;
a = 25 ; b = 3.24 ; c = a + b * b – 35 ;
}
```

No error. Multiple C statements can be written in a single line.

Point out the errors, if any, in the following programs

```
#include <stdio.h>
int main( )
{
int a = 35 ; float b = 3.24 ;
printf ( "%d %f %d", a, b + 1.5, 235 ) ;
}
```

No error. The list being printed in printf( ) may contain variables, constants or expressions.

Point out the errors, if any, in the following programs

```
#include <stdio.h>
int main( )
{
int a, b, c ;
scanf ( "%d %d %d", a, b, c ) ;
}
```

Error. We should use & before each variable used in scanf( ).

Point out the errors, if any, in the following programs

```
#include <stdio.h>
int main( )
{
int m1, m2, m3
printf ( "Enter values of marks in 3 subjects: " )
scanf ( "%d %d %d", &m1, &m2, &m3 )
printf ( "You entered %d %d %d", m1, m2, m3 )
}
```

Error. Semicolon should be present at the end of type declaration, printf( ) and scanf( ) statements.

Tokens in C

- Keywords
  - These are reserved words of the C language. For example **`int, float, if, else, for, while`** etc.
- Identifiers
  - An Identifier is a sequence of letters and digits, but must start with a letter. Underscore (_) is treated as a letter. Identifiers are case sensitive. Identifiers are used to name variables, functions etc.
  - Valid: **`Root, _getchar, __sin, x1, x2, x3, x_1, If`**
  - Invalid: **`324, short, price$, My Name`**
- Constants
  - Constants like 13, ‘a’, 1.3e-5 etc.
- String Literals
  - A sequence of characters enclosed in double quotes as “…”. For example “13” is a string literal and not number 13. ‘a’ and “a” are different.
- Operators
  - Arithmetic operators like `+, -, *, / ,%` etc.
  - Logical operators like `||, &&, !` etc. and so on.
- White Spaces
  - Spaces, new lines, tabs, comments (A sequence of characters enclosed in /* and */) etc. These are used to separate the adjacent identifiers, keywords and constants.

Character and string constants

- **`'c'`** , a single character in single quotes are stored as char. Some special character are represented as two characters in single quotes.
  `'\n'` = newline, **`'\t'`** = tab, **`'\\'`** = backlash, **`'\"'`** = double quotes.
  Char constants also can be written in terms of their ASCII code.
  `'\060' = '0'` (Decimal code is 48).
- A sequence of characters enclosed in double quotes is called a string constant or string literal. For example
  `"Charu"`
  `"A"`
  `"3/9"`
  `"x = 5"`

Escape sequence

Escape sequence	Purpose	Escape sequence	Purpose
\a	Audible signal	\?	Question mark
\b	Backspace	\\	Back slash
\t	Tab	\’	Single quote
\n	Newline	\\”	Double quote
\v	Vertical tab	\0	Octal constant
\f	New page/clear screen	\x	Hex constant
\r	Carriage return		

Declarations

- Any variable used in the program must be declared before using it in any statement. The type declaration statement is written at the beginning of **main( )** function.

Ex.:

```
int bas ;
float rs, grosssal ;
char name, code ;
```

Declarations

(a) While declaring the type of variable we can also initialize it as shown below.

```
int i = 10, j = 25 ;
float a = 1.5, b = 1.99 + 2.4 * 1.44 ;
```

Declarations

(b)The order in which we define the variables is sometimes important, sometimes not.

For example, **int i = 10, j = 25 ;**

is same as **int j = 25, i = 10 ;**

However, **float a = 1.5, b = a + 3.1 ;**

is alright, but **float b = a + 3.1, a = 1.5 ;**

is not.

This is because here we are trying to use **a** before defining it.

Declarations

(c) The following statements would work

```
int a, b, c, d ;
a = b = c = 10 ;
```

However, the following statement would not work

```
int a = b = c = d = 10 ;
```

Once again we are trying to use **b** (to assign to **a**) before defining it.

Arithmetic Instruction

- A C arithmetic instruction consists of a variable name on the left hand side of = and variable names and constants on the right hand side of =.
- The variables and constants appearing on the right hand side of = are connected by arithmetic operators like **+, -, *, and /**.

```
int ad ;
float kot, deta, alpha, beta, gamma ;
ad = 3200 ;
kot = 0.0056 ;
deta = alpha * beta / gamma + 3.2 * 2 / 5 ;
```

Here, ***, /, -, +** are the arithmetic operators.

= is the assignment operator.

2, 5 and 3200 are integer constants.

3.2 and 0.0056 are real constants.

ad is an integer variable.

kot, deta, alpha, beta, gamma are real variables.

Arithmetic Instruction

Though Arithmetic instructions look simple to use, one often commits mistakes in writing them. Let us take a closer look at these statements.

Note the following points carefully:

(a) C allows only one variable on left-hand side of **=**. **That is, z = k * l is legal, whereas k * l = z is illegal.**

(b) In addition to the division operator C also provides a modular division operator. This operator returns the remainder on dividing one integer with another.

Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0.

Note that the modulus operator (%) cannot be applied on a float. Also note that on using % the sign of the remainder is always same as the sign of the numerator.

Thus -5 % 2 yields –1, whereas, 5 % -2 yields 1.

Arithmetic Instruction

(c) An arithmetic instruction is at times used for storing character constants in character variables.

```
char a, b, d ;                    a = 'F' ;
b = 'G' ;                         d = '+' ;
```

When we do this, the ASCII (American Standard `1Code for Information Interchange)values of the characters are stored in the variables. ASCII codes are used to represent any character in memory. For example, ASCII codes of ‘F’ and ‘G’ are 01000110 and 01000111. ASCII values are nothing but the decimal equivalent of ASCII codes. Thus ASCII values of ‘F’ and ‘G’ are 70 and 71.

Arithmetic Instruction

(d) Arithmetic operations can be performed on **int**s, **float**s and **char**s. Thus the statements,

```
char x, y ;
int z ;
x = 'a' ;
y = 'b' ;
z = x + y ;
```

are perfectly valid, since the addition is performed on the ASCII values of the characters and not on characters themselves. The ASCII values of ‘a’ and ‘b’ are 97 and 98, and hence can definitely be added.

Arithmetic Instruction

(e) No operator is assumed to be present. It must be written explicitly.

In the following example, the multiplication operator after b must be explicitly written.

a = c.d.b(xy)	usual arithmetic statement
a = c*d*b*(x*y)	C statement

Arithmetic Instruction

(f) There is no operator in C to perform exponentiation operation. Exponentiation has to be carried out as shown below:

```
# include <math.h>
# include <stdio.h>
int main( )
{
float a ;
a = pow ( 3.0, 2.0 ) ;
printf ( "%f", a ) ;
}
```

Here **pow( )** function is a standard library function. It is being used to raise 3.0 to the power of 2.0. The **pow( )** function works only with real numbers, hence we have used 3.0 and 2.0 instead of 3 and 2.

#include <math.h> is a preprocessor directive. It is being used here to ensure that the **pow( )** function works correctly.

Explore other mathematical functions like **abs( )**, **sqrt( )**,

sin( ), **cos( )**, **tan( )**, etc., declared in **math.h** on your own.

Global and Local Variables

- **Global Variables**
  - These variables are declared outside all functions.
  - Life time of a global variable is the entire execution period of the program.
  - Can be accessed by any function defined below the declaration, in a file

```
/* Compute Area and Perimeter of  a circle */
#include <stdio.h>
float pi = 3.14159;  /* Global */

main() {
  float      rad;   /* Local */
  printf( “Enter the radius “ );
  scanf(“%f” , &rad);
  if ( rad > 0.0 ) {
    float area = pi * rad * rad;
    float peri = 2 * pi * rad;
    printf( “Area = %f\n” , area );
    printf( “Peri = %f\n” , peri );
  }
  else
    printf( “Negative radius\n”);
  printf( “Area = %f\n” , area );
}
```

Global and Local Variables

Local Variables

- These variables are declared inside some functions.
- Life time of a local variable is the entire execution period of the function in which it is defined.
- Cannot be accessed by any other function.
- In general variables declared inside a block are accessible only in that block.

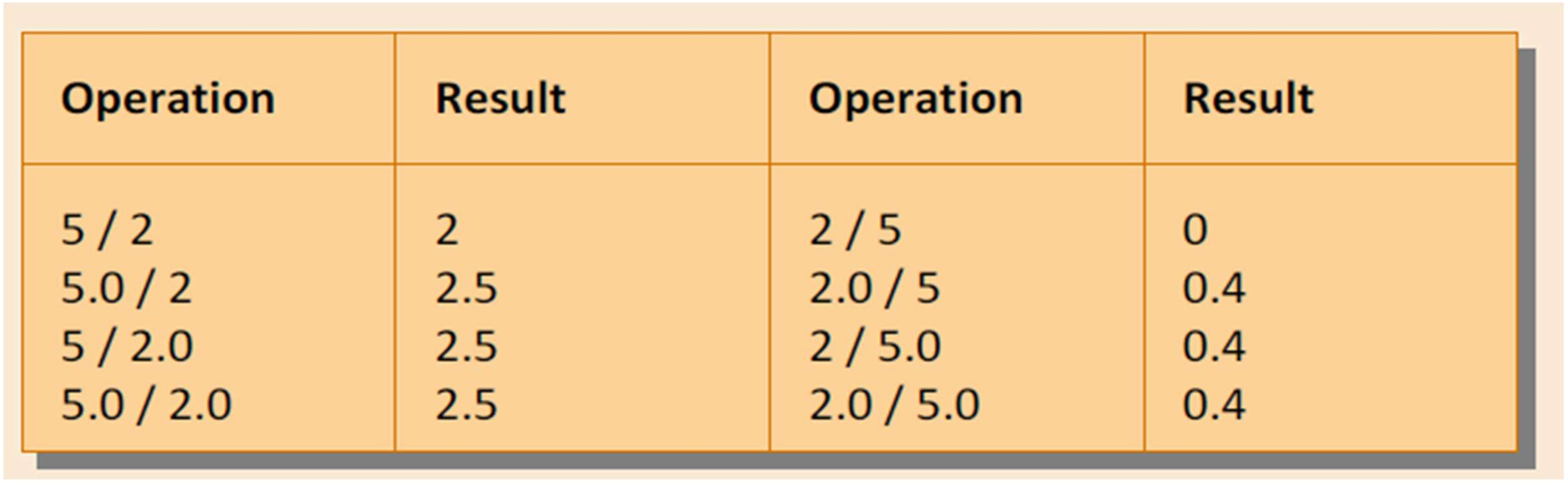
```
/* Compute Area and Perimeter of  a circle */
#include <stdio.h>
float pi = 3.14159;  /* Global */

main() {
  float  rad;      /* Local */
  printf( "Enter the radius " );
  scanf("%f" , &rad);
  if ( rad > 0.0 ) {
    float area = pi * rad * rad;
    float peri = 2 * pi * rad;
    printf( "Area = %f\n" , area );
    printf( "Peri = %f\n" , peri );
  }
  else
    printf( "Negative radius\n");
  printf( "Area = %f\n" , area );
}
```

Integer and Float Conversions

(a) An arithmetic operation between an integer and integer always yields an integer result.

(b) An operation between a real and real always yields a real result.

(c) An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed.



Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Integer and Float Conversions

It may so happen that the type of the expression on right hand side and the type of the variable on the left-hand side of an assignment operator may not be same. In such a case, the value of the expression is promoted or demoted depending on the type of the variable on lefthand side of =.

For example, consider the following assignment statements.

```
int i ;
float b ;
i = 3.5 ;
b = 30 ;
```

```
main()
{
int i ;
float b ;
i = 3.5 ;
b = 30 ;
printf("%d %f",i,b);
}
```

Integer and Float Conversions

- Here in the first assignment statement, though the expression's value is a **float** (3.5), it cannot be stored in **i** since it is an **int**. In such a case, the **float** is demoted to an **int** and then its value is stored. Hence what gets stored in **i** is 3.

Exactly opposite happens in the next statement.

- Here, 30 is promoted to 30.0 and then stored in **b**, since **b** being a **float** variable cannot hold anything except a **float** value.

```
main()
{
int i ;
float b ;
i = 3.5 ;
b = 30 ;
printf("%d %f",i,b);
}
```

Integer and Float Conversions

```
float a, b, c ; int s ;
s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;
```

At the end of calculation, what will be stored in s?

int? / float?

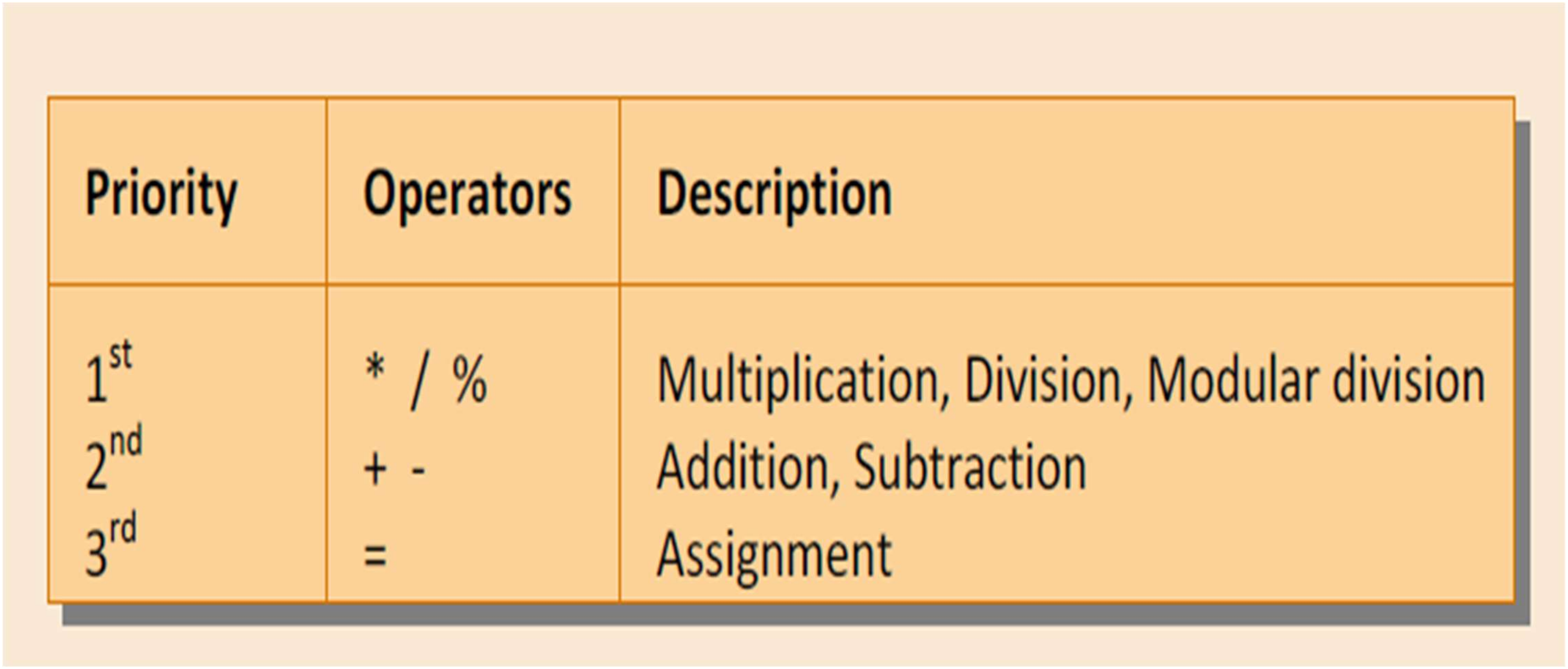
Some operands are **int**s whereas others are **float**s. As we know, during evaluation of the expression, the **int**s would be promoted to **float**s and the result of the expression would be a **float**.

But when this **float** value is assigned to **s** it is again demoted to an **int** **and then stored in s.**

Integer and Float Conversions K is int a is real

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.222222
k = 2 / 9.0	0	a = 2 / 9.0	0.222222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.222222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

Hierarchy of Operations

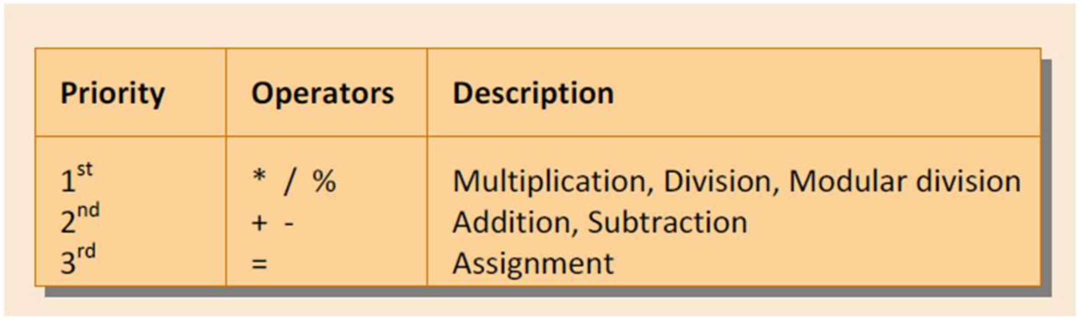


Priority	Operators	Description
1st	* / %	Multiplication, Division, Modular division
2nd	+ -	Addition, Subtraction
3rd	=	Assignment

Hierarchy of Operations

Determine the hierarchy of operations and evaluate the following expression, assuming that **i** is an integer variable:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8



Priority	Operators	Description
1st	* / %	Multiplication, Division, Modular division
2nd	+ -	Addition, Subtraction
3rd	=	Assignment

Hierarchy of Operations

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8 operation: *

i = 1 + 4 / 4 + 8 - 2 + 5 / 8 operation: /

i = 1 + 1+ 8 - 2 + 5 / 8 operation: /

i = 1 + 1 + 8 - 2 + 0 operation: /

i = 2 + 8 - 2 + 0 operation: +

i = 10 - 2 + 0 operation: +

i = 8 + 0 operation : -

i = 8 operation: +

Hierarchy of Operations

Determine the hierarchy of operations and evaluate the following expression, assuming that **kk** is a float variable:

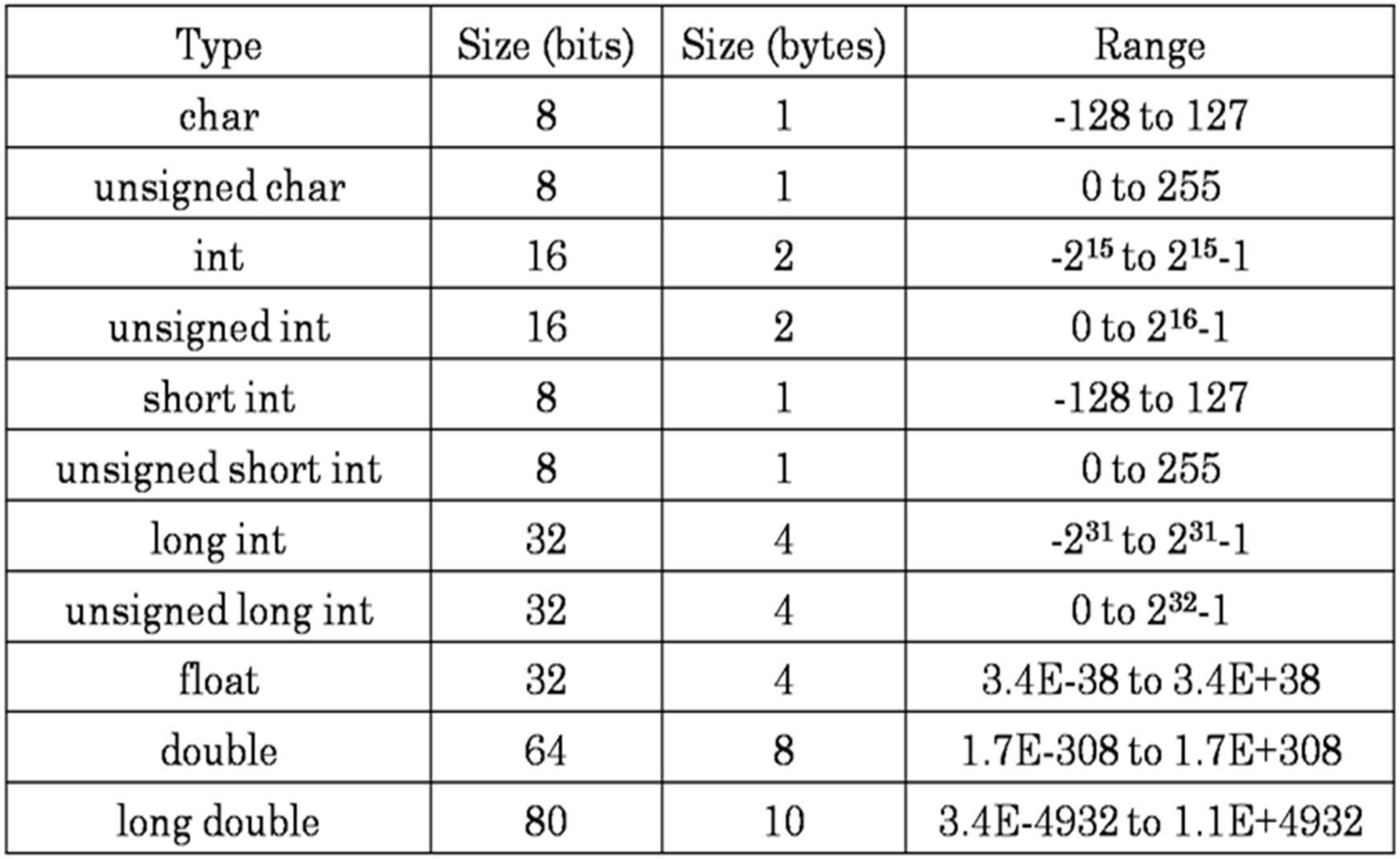
kk = 3 / 2 * 4 + 3 / 8

Re_LOOK

Basic data types

Integer	Int, unsigned int,
Integer	short int, unsigned short int
Integer	long int, unsigned long int
Character	signed char, unsigned char
floating point	Float, double, long double

Basic data types in C



Type	Size (bits)	Size (bytes)	Range
char	8	1	-128 to 127
unsigned char	8	1	0 to 255
int	16	2	-2^{15} to $2^{15}-1$
unsigned int	16	2	0 to $2^{16}-1$
short int	8	1	-128 to 127
unsigned short int	8	1	0 to 255
long int	32	4	-2^{31} to $2^{31}-1$
unsigned long int	32	4	0 to $2^{32}-1$
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+308
long double	80	10	3.4E-4932 to 1.1E+4932

Source: https://www.startertutorials.com/blog/data-types-c.html

- **Ranges of Data Types:**
  The range of a data type gives us the minimum and maximum value that can be stored inside the variable of the given data type.
- **Examples:**
  Range of character = -128 to 127

✓ signed data types, use formula $\mathbf{-2^{(n-1)}}$ to $\mathbf{(2^{(n-1)}) - 1}$

✓ unsigned data types, use formula **0** to $\mathbf{(2^{n}) - 1}$

✓ Where n is the number of bits in both the cases

User defined data types

- Allows to define identifiers as their own data types

typedef (Keyword)

Example:

typedef int num;

typedef temp;

- Enum (Keyword)

- Example:
- enum identifier{value1,value2}
- enum days{1,2,3,4}

Derived data types

- The data types which are created using the already existing primitive or fundamental types are known as derived data types.
- Like user-defined data types we cannot declare new variables using the derived data types.
- **Examples**
- Arrays, functions, structures, unions and pointers

C operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Conditional Operator

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operator	Description	Example
=	Assign Used to assign the values from right side of the operands to left side of the operand.	C = A + B will assign the value of A + B to C.
+=	Add then assign Adds the value of the right operand to the value of the left operand and assigns the result to the left operand.	C += A is same as C = C + A
-=	Subtract then assign Subtracts the value of the right operand from the value of the left operand and assigns the result to the left operand.	C -= A is same as C = C – A
*=	Multiply then assign Multiplies the value of the right operand with the value of the left operand and assigns the result to the left operand.	C *= A is same as C = C * A

%=	Modulus then assign Takes modulus using the values of the two operands and assigns the result to the left operand.	C %= A is same as C = C % A
<<=	Left shift and assign Used for left shift AND assignment operator.	C <<= 4 is same as C = C << 4
>>=	Right shift and assign Used for right shift AND assignment operator.	C >>= 5 is same as C = C >> 5
&=	Bitwise AND assign Used for bitwise AND assignment operator.	C &= 7 is same as C = C & 7
^=	Used for bitwise exclusive OR and assignment operator.	C ^= 6 is same as C = C ^ 6
|=	Used for bitwise inclusive OR and assignment operator.	C |= 9 is same as C = C | 9

Relational operators

- A relational operator also known as a comparison operator, is an that compares two values
- Relational operators return true or false values
-

Operator	Meaning	Example
<	Less than	3<5 gives 1
>	Greater than	7>9 gives 0
<=	Less than or equal to	100<=100 gives 1
>=	Grater than or equal to	50>=100 gives 0

Logical operator

Operator	Description	Example
&&	This is the **AND** operator in C programming language. It performs logical conjunction of two expressions. (If both expressions evaluate to True, then the result is True. If either of the expression evaluates to False, then the result is False)	((A==7) && (B>7)) equals to 0
||	It is the **OR** operator in C programming language. It performs a logical disjunction on two expressions. (If either or both of the expressions evaluate to True, then the result is True)	((A==7) || (B>7)) equals to 1
!	It is the **Logical NOT** Operator in C programming language. It is used to reverse the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false and vice versa.	!(A && B) is true

Re-Cap

Point out the errors, if any, in C statements

A. x = (y + 3) ;
B. cir = 2 * 3.141593 * r ;
C. char = ‘3’ ;
D. 4 / 3 * 3.14 * r * r * r = vol_of_sphere;
E. volume = a3 ;
F. area = 1 / 2 * base * height ;
G. si = p * r * n / 100 ;
H. area of circle = 3.14 * r * r ;
I. peri_of_tri = a + b + c ;
J. slope = (y2 – y1) ÷ (x2 – x1) ;
K. 3 = b = 4 = a ;
L. count = count + 1 ;
M. char ch = '25 Apr 12' ;

A) No error
B) No error
C) **Error.** Keyword cannot be used as a variable name.
D) **Error.** On the left-hand side of equal to (=) there can only be a variable.
E) **Error.** a3 is not a valid statement. Instead, we can use a * a * a.
F) No error
G) No error
H) **Error.** **area of circle** is an invalid variable name.
I) No error
J) **Error.** ‘÷’ is an invalid operator
K) **Error.** A variable name must be present on the left hand side of ‘=’ operator.
L) No error
M) **Error.** Multiple characters cannot be stored in a char variable.

Point out the errors, if any, in C statements

A. x = (y + 3) ;
B. cir = 2 * 3.141593 * r ;
C. char = ‘3’ ;
D. 4 / 3 * 3.14 * r * r * r = vol_of_sphere;
E. volume = a3 ;
F. area = 1 / 2 * base * height ;
G. si = p * r * n / 100 ;
H. area of circle = 3.14 * r * r ;
I. peri_of_tri = a + b + c ;
J. slope = (y2 – y1) ÷ (x2 – x1) ;
K. 3 = b = 4 = a ;
L. count = count + 1 ;
M. char ch = '25 Apr 12' ;

A) **No error**
B) **No error**
C) **Error.** Keyword cannot be used as a variable name.
D) **Error.** On the left-hand side of equal to (=) there can only be a variable.
E) **Error.** a3 is not a valid statement. Instead, we can use a * a * a.
F) **No error**
G) **No error**
H) **Error.** **area of circle** is an invalid variable name.
I) **No error**
J) **Error.** ‘÷’ is an invalid operator
K) **Error.** A variable name must be present on the left hand side of ‘=’ operator.
L) **No error**
M) **Error.** Multiple characters cannot be stored in a char variable.

Evaluate the following expressions and show their hierarchy

ans = 5 * b * b * x - 3 * a * y * y -8 * b * b *x + 10 * a * y ;

(a = 3, b = 2, x = 5, y = 4 assume **ans** to be an int)

ans = **5 * 2** * 2 * 5 - 3 * 3 * 4 * 4 - 8 * 2 * 2 * 5 + 10 * 3 *4

ans = **10 * 2** * 5 - 3 * 3 * 4 * 4 - 8 * 2 * 2 * 5 + 10 * 3 * 4 operation: *

ans = **20 * 5** - 3 * 3 * 4 * 4 - 8 * 2 * 2 * 5 + 10 * 3 * 4 operation: *

ans = 100 - **3 * 3** * 4 * 4 - 8 * 2 * 2 * 5 + 10 * 3 * 4 operation: *

ans = 100 - **9 * 4** * 4 - 8 * 2 * 2 * 5 + 10 * 3 * 4 operation: *

ans = 100 - **36 * 4** - 8 * 2 * 2 * 5 + 10 * 3 * 4 operation: *

ans = 100 - 144 - **8 * 2** * 2 * 5 + 10 * 3 * 4 operation: *

ans = 100 - 144 - **16 * 2** * 5 + 10 * 3 * 4 operation: *

ans = 100 - 144 - **32 * 5** + 10 * 3 * 4 operation: *

ans = 100 - 144 - 160 + **10 * 3** * 4 operation: *

ans = 100 - 144 - 160 + **30 * 4** operation: *

ans = **100 - 144** - 160 + 120 operation: -

ans = **-44 - 160** + 120 operation: -

ans = **-204 + 120** operation: +

ans = -84

Evaluate the following expressions and show their hierarchy

res = 4 * a * y / c - a * y / c ;

(a = 4, y = 1, c = 3, assume res to be an int)

res = **4 * 4** * 1 / 3 - 4 * 1 / 3 operation: *

res = **4 * 4** * 1 / 3 - 4 * 1 / 3 operation: *

res = **16 * 1** / 3 - 4 * 1 / 3 operation: *

res = **16 / 3** - 4 * 1 / 3 operation: /

res = 5 - **4 * 1** / 3 operation: *

res = 5 - **4 / 3** operation: /

res = 5 - 1 operation: -

res = 4

Evaluate the following expressions and show their hierarchy

R = x * x + 2 * x + 1 / 2 * x * x + x + 1 ;

(x = 3.5, assume R to be a float)

R = 23.79081

What will be the output of the program?

```
# include <stdio.h>
int main( )
{
int i = 2, j = 3, k, l ;
float a, b ;
k = i / j * j ;
l = j / i * i ;
a = i / j * j ;
b = j / i * i ;
printf ( "%d %d %f %f\n", k, l, a, b ) ;
return 0 ;
}
```

Output:

```
0 2 0.000000 2.000000
```

What will be the output of the program?

```
# include <stdio.h>
int main( )
{
int a, b, c, d ;
a = 2 % 5 ;
b = -2 % 5 ;
c = 2 % -5 ;
d = -2 % -5 ;
printf ( "a = %d b = %d c = %d d = %d\n", a, b, c, d ) ;
return 0 ;
}
```

Output:

```
a = 2 b = -2 c = 2 d = -2
```

Relational Operators

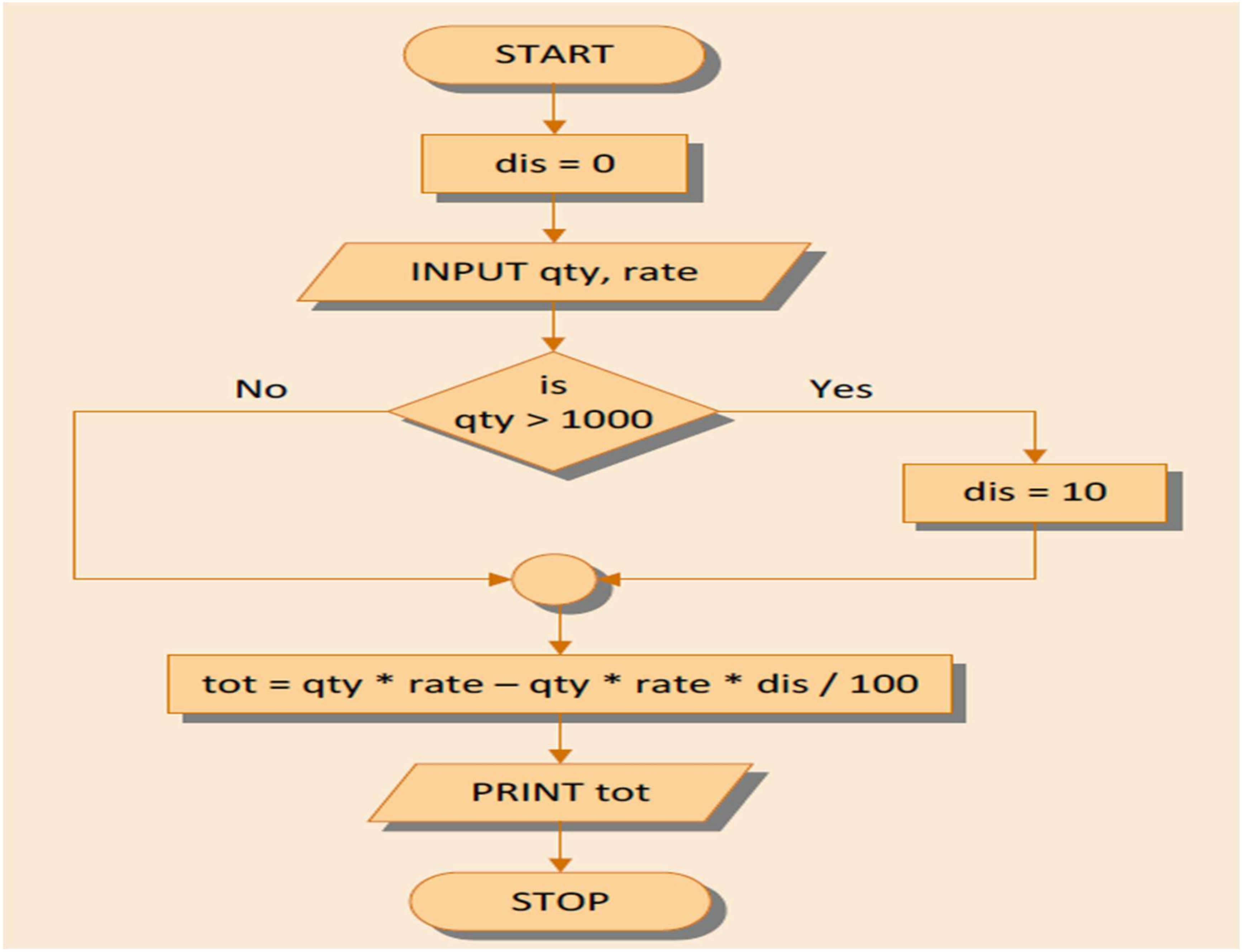
OPERATOR SYMBOL	OPERATOR NAME
==	Equal to
!=	Not Equal to
<	Less Than
>	Greater Than
<=	Less Than Equal to
>=	Greater Than Equal to

Relational Operators

Equal to (==) operator is a binary operator hence it requires two operands to perform the comparison.
If the two values are equal, it returns true. Otherwise, it returns false.
It does not work for strings or arrays.
5==5 will return true.

While purchasing certain items, a discount of 10% is offered if the quantity purchased is more than 1000. If quantity and price per item are input through the keyboard, write a program to calculate the total expenses.

Relational Operators



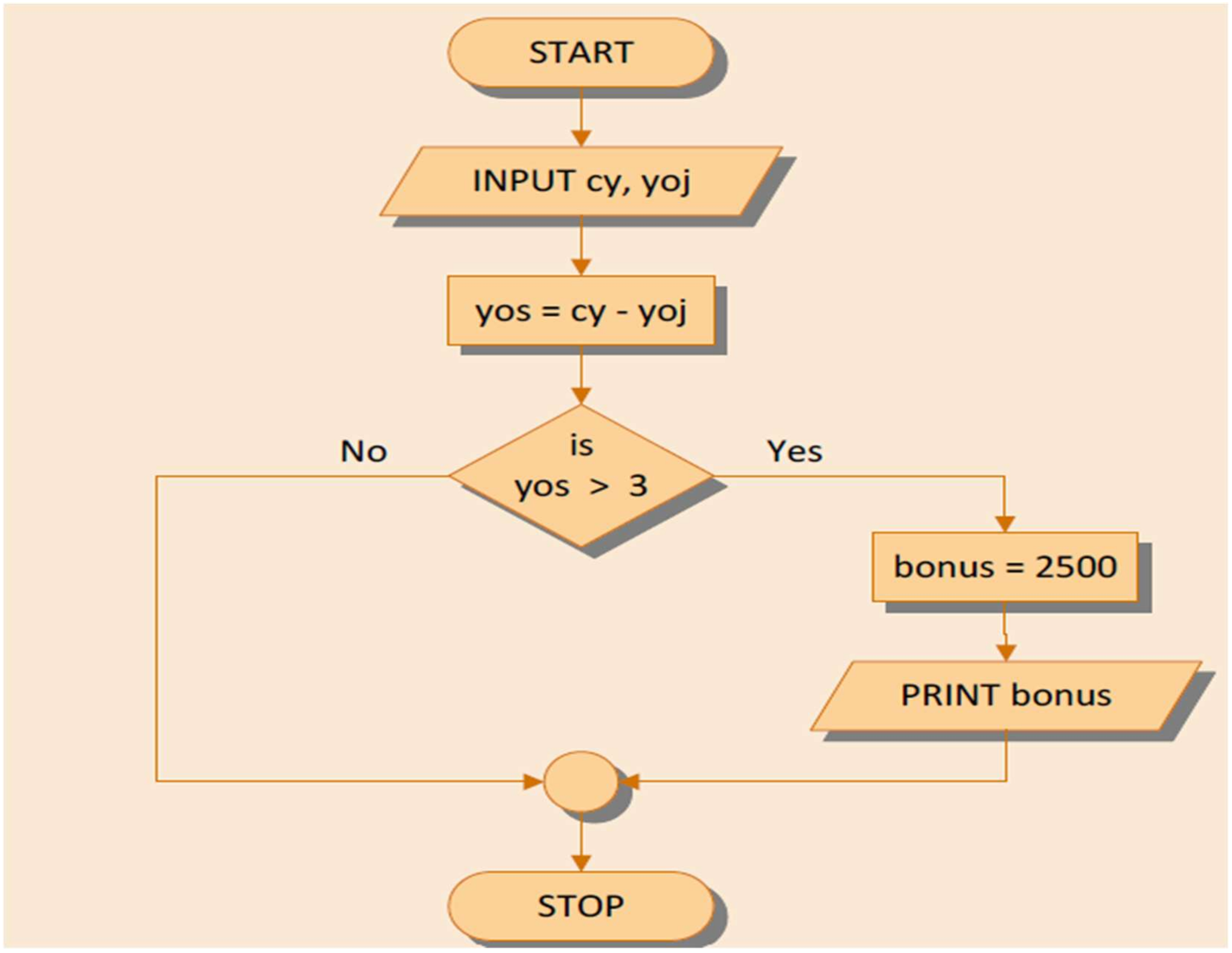
Relational Operators

```
/* Calculation of total expenses */
# include <stdio.h>
int main( )
{
int qty, dis = 0 ;
float rate, tot ;
printf ( "Enter quantity and rate " ) ;
scanf ( "%d %f", &qty, &rate) ;
if ( qty > 1000 )
dis = 10 ;
tot = ( qty * rate ) - ( qty * rate * dis / 100 ) ;
printf ( "Total expenses = Rs. %f\n", tot ) ;
return 0 ;
}
```

Multiple Statements within *if*

The current year and the year in which the employee joined the organization are entered through the keyboard. If the number of years for which the employee has served the organization is greater than 3, then a bonus of Rs. 2500/- is given to the employee. If the years of service are not greater than 3, then the program should do nothing.

Multiple Statements within *if*



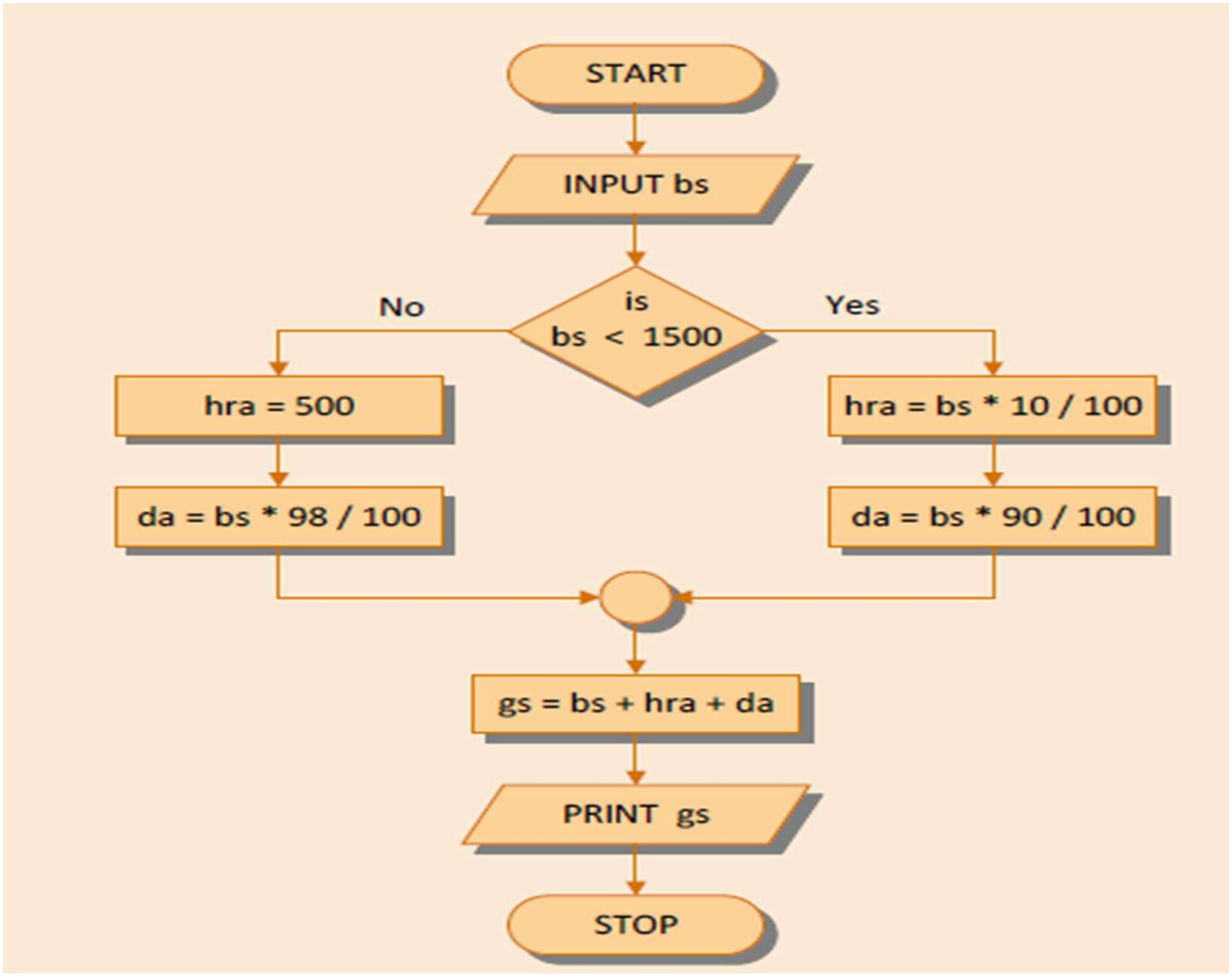
Multiple Statements within *if*

```
/* Calculation of bonus */
# include <stdio.h>
int main( )
{
int bonus, cy, yoj, yos ;
printf ( "Enter current year and year of joining " ) ;
scanf ( "%d %d", &cy, &yoj ) ;
yos = cy - yoj ;
    if ( yos > 3 )
    {
    bonus = 2500 ;
    printf ( "Bonus = Rs. %d\n", bonus ) ;
    }
return 0 ;
}
```

The if-else Statement

If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.

The if-else Statement



- https://www.w3resource.com/c-programming/c-variable.php