Basics of "C" Lab-4

Constant, variable, Operators, Expression

Objectives

- Understand the basic structure of a program in C.
- Learn the commands used in UNIX/LINUX and MS-DOS for compiling and running a program in C.
- Obtain a preliminary idea of the keywords in C.
- Learn the data types, variables, constants, operators, and expressions in C.
- Understand and grasp the precedence and associativity rules of operators in C.
- Get acquainted with the rules of type conversions in C.

Key Words

- **ASCII :** It is a standard code for representing characters as numbers that is used on most microcomputers, computer terminals, and printers. In addition to printable characters, the ASCII code includes control characters to indicate carriage return, backspace, etc.

- **Assembler :**The assembler creates the object code.

- **Associativity :**The associativity of operators determines the order in which operators of equal precedence are evaluated when they occur in the same expression. Most operators have a left-to-right associativity, but some have right-to-left associativity.

Key Words

- **Compiler:** A system software that translates the source code to assembly code.
- **Constant :**A constant is an entity that doesn't change.
- **Data type:** The type, or data type, of a variable determines a set of values that the variable might take and a set of operations that can be applied to those values.
- **Debugger:** A debugger is a program that enables you to run another program step-by-step and examine the value of that program's variables.
- **Identifier:** An identifier is a symbolic name used in a program and defined by the programmer. Ex : name of variables, arrays, functions etc like emp_name, total_amount

Key Words

- **IDE :**An Integrated Development Environment or IDE is an editor which offers a complete environment for writing, developing, modifying, deploying, testing, and debugging the programs.
- **Identifier:** An identifier or name is a sequence of characters invented by the programmer to identify or name a specific object.
- **Keyword:** Keywords are explicitly reserved words that have a strict meaning as individual tokens to the compiler. They cannot be redefined or used in other contexts.

Key Words

- **Linker:** If a source file references library functions or functions defined in other source files, the linker combines these functions to create an executable file.
- **Precedence :**The precedence of operators determines the order in which different operators are evaluated when they occur in the same expression. Operators of higher precedence are applied before operators of lower precedence.
- **Pre processor :**The C pre processor is used to modify the source program before compilation according to the pre processor directives specified.

Key Words

- **L-value:** An l-value is an expression to which a value can be assigned.
- **R-value :**An r-value can be defined as an expression that can be assigned to an l-value.
- **Token:** A token is one or more symbols understood by the compiler that help it interpret your code.
- **Word :**A word is the natural unit of memory for a given computer design. The word size is the computer's preferred size for moving units of information around; technically it's the width of the processor's registers.

Key Words

- **Whitespace Space, newline, tab character and comment are** collectively known as whitespace.

- **Variable:** A variable is a named memory location. Every variable has a type, which defines the possible values that the variable can take, and an identifier, which is the name by which the variable is referred.

- **Bug:**Any type of error in a program is known as bug. There are three types of errors that may occur:
  - ✓Compile errors,
  - ✓Linking errors,
  - ✓Runtime errors

Why Learn "C" ?

- There are a large number of programming languages in the world today even so, there are several reasons to learn C, some of which are stated as follows:

a . *C is quick*.
b . *C is a core language* : In computing, C is a general purpose, cross-platform, block structured procedural, imperative computer programming language.
c *. C is a small language*: C has only thirty-two keywords. This makes it relatively easy to learn compared to bulkier languages.
d . *C is portable*.

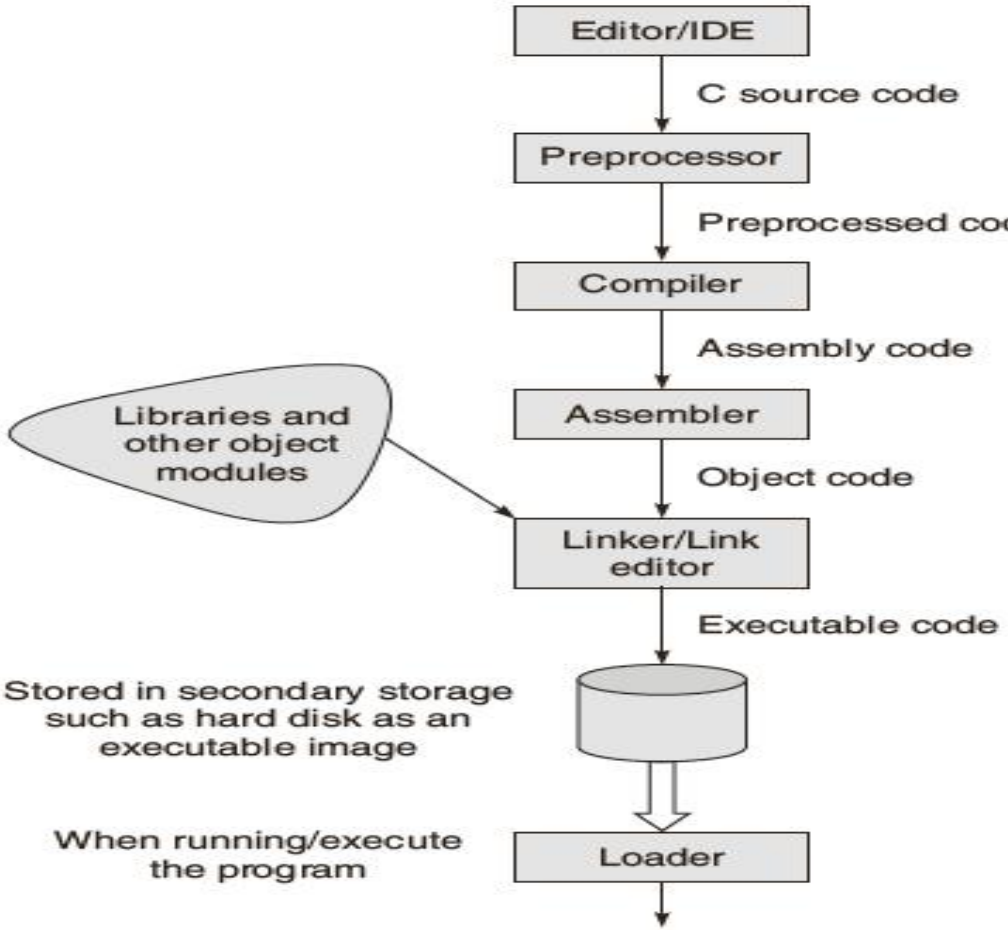
Developing Programs in "C"

There are mainly three steps:

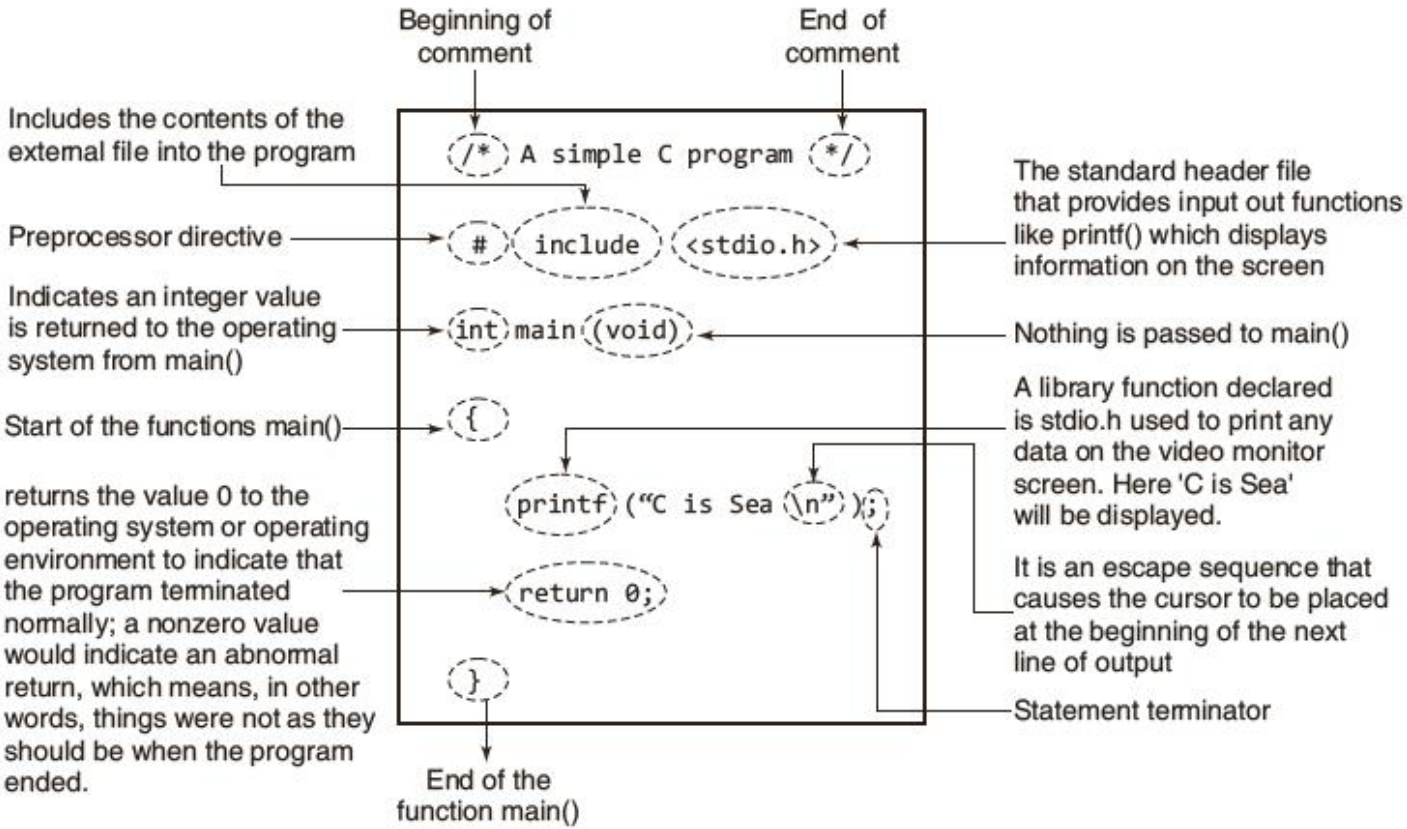
1. **Writing the C program**
2. **Compiling the program and**
3. **Executing it.**

- For these steps, some software components are required, namely an operating system, a text editor(*integrated development environment*), the C compiler, assembler, and linker.
- C uses a semicolon as a statement terminator; the semicolon is required as a signal to the compiler to indicate that a statement is complete.
- All program instructions, which are also called statements, have to be written in lower case characters.

Developing Programs in "C"



Illustrated Version of a Program



Backlash Code

Code	Meaning
\a	Ring terminal bell (a is for alert) [ANSI] extension]
\?	Question mark [ANSI extension]
\b	Backspace
\r	Carriage return
\f	Form feed
\t	Horizontal tab
\v	Vertical tab
\0	ASCII null character
\\	Backslash
\”	Double quote
\’	Single quote
\n	New line
\o	Octal constant
\x	Hexadecimal constant

Parts of C Program

Header File

- The header files, usually incorporate data types, function declarations and macros, resolves this issue. The file with .h extension is called header file, because it's usually included at the head of a program.
- Every C compiler that conforms to the international standard (ISO/IEC 9899) for the language will have a set of standard header files supplied with it.
- The header files primarily contain declarations relating to standard library functions and macros that are available with C.

Standard Header Files

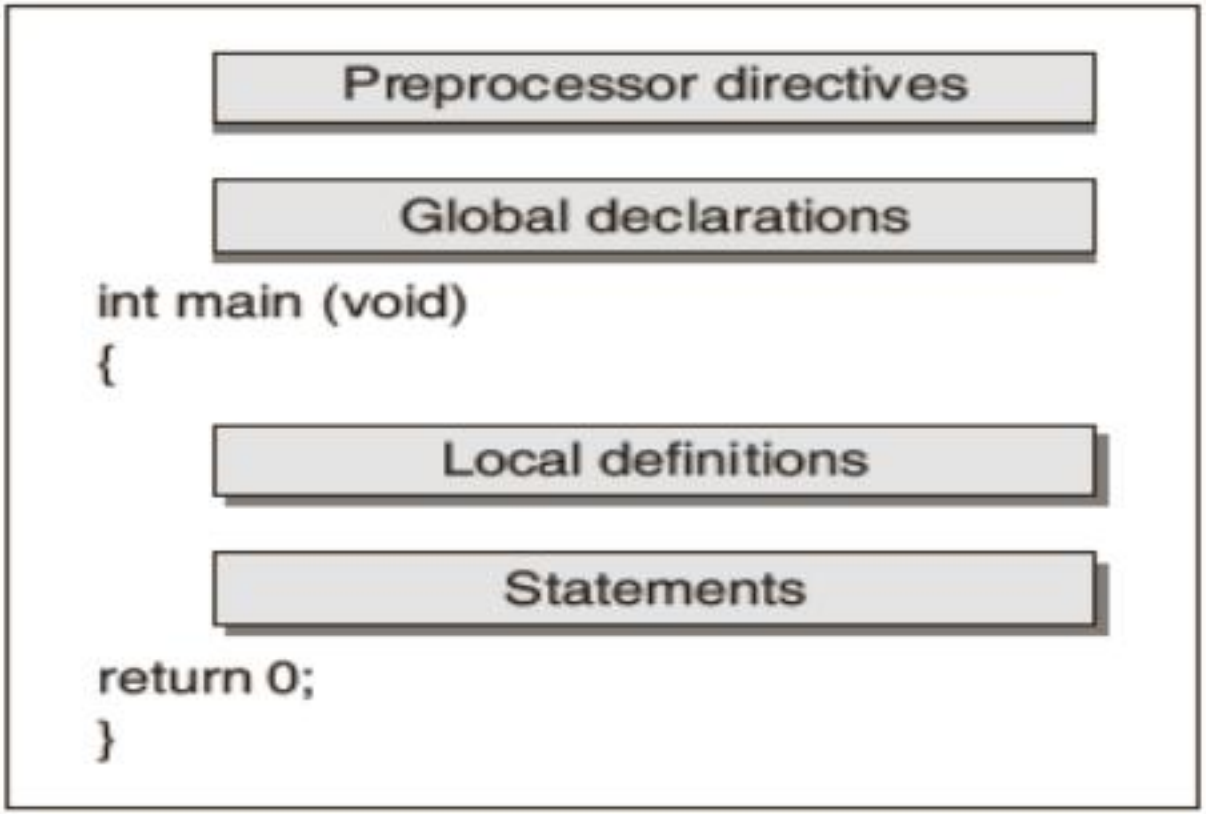
- During compilation, the compilers perform type checking to ensure that the calls to the library and other user-defined functions are correct. This form of checking helps to ensure the semantic correctness of the program.

assert.h	inttypes.h	signal.h	stdlib.h
complex.h	iso646.h	stdarg.h	string.h
ctype.h	limits.h	stdbool.h	tgmath.h
errno.h	locale.h	stddef.h	time.h
fenv.h	math.h	stdint.h	wchar.h
float.h	setjmp.h	stdio.h	wctype.h

Philosophy : main()

- main() is a user defined function. main() is the first function in the program which gets called when the program executes. The start up code c calls main() function. We can't change the name of the main() function.
- main() is must.
- According to ANSI/ISO/IEC 9899:1990 International Standard for C, the function called at program start up is named main. The implementation declares no prototype for this function. It can be defined with no parameters:
- int main(void) { /* ... */ }
- or with two parameters (referred to here as argc and argv) :
- int main(int argc, char *argv[ ]) { /* ... */ }

Structure : C Program



Declaration & Definition

- Declaration means describing the type of a data object to the compiler but not allocating any space for it.
  - ✓ A *declaration announces the properties of a data* object or a function. If a variable or function is declared and then later make reference to it with data objects that do not match the types in the declaration, the compiler will complain.
  - ✓ data_type variable_name_1,
- Definition means declaration of a data object and also allocating space to hold the data object.
  - ✓ A *definition, on the other hand, actually sets aside* storage space (in the case of a data object) or indicates the sequence of statements to be carried out (in the case of a function).

ASCII – Binary Character Table

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Variables : Attributes

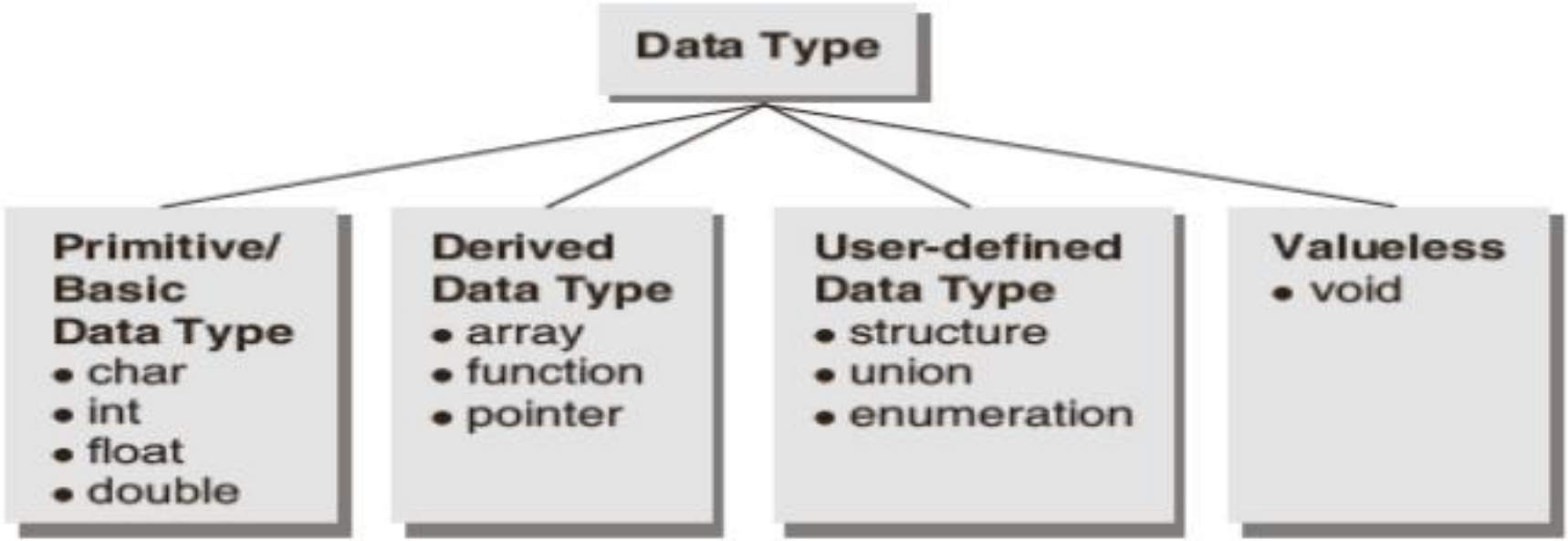
- **All variables have three important attributes**:

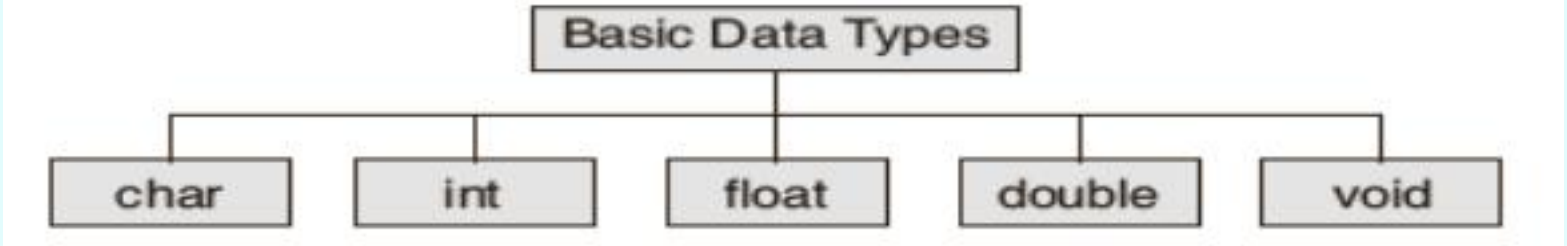
  ✓A **data type** that is established when the variable is defined, e.g., integer, real, character. Once defined , the type of a C variable cannot be changed.

  ✓ A **name** of the variable.

  ✓ A **value** that can be changed by assigning a new value to the variable. The kind of values a variable can assume depends on its type. For example, an integer variable can only take integer values, e.g., 2, 100, –12.

Classification : Data Types





Basic Data Types : Size & Range

- **16 bit computer:**

Data type	Size (in bits)	Range
char	8	–128 to 127
int	16	–32768 to 32767
float	32	1.17549×10^{-38} to 3.40282×10^{38}
double	64	2.22507×10^{-308} to 1.79769×10^{308}
void	8	valueless

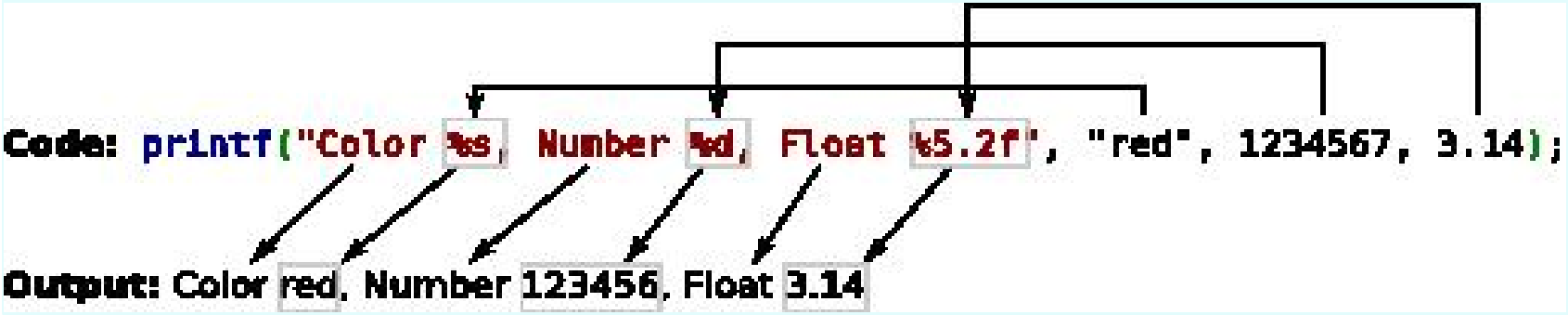
- **32 bit computer:**

Data type	Size (in bits)	Range
char	8	–128 to 127
int	32	–2147483648 to 2147483647
float	32	1.17549×10^{-38} to 3.40282×10^{38}
double	64	2.22507×10^{-308} to 1.79769×10^{308}
void	8	valueless

printf function

The printf function (the name comes from “print formatted”) prints a string on the screen using a “format string” that includes the instructions to mix several strings and produce the final string to be printed on the screen.

printf("The value is %d\n", counter);



printing number

```c
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

printing float and double

```c
#include <stdio.h>
int main()
{
    float number1 = 13.5;
    double number2 = 12.4;

    printf("number1 = %f\n", number1);
    printf("number2 = %lf", number2);
    return 0;
}
```

Output

```
number1 = 13.500000
number2 = 12.400000
```

print characters

```c
#include <stdio.h>
int main()
{
    char chr = 'a';
    printf("character = %c", chr);
    return 0;
}
```

Output

```
character = a
```

scanf function

- In the C programming language, scanf is a function that reads formatted data from stdin (i.e, the standard input stream, which is usually the keyboard, unless redirected) and then writes the results into the arguments given.
- It uses the **& (address operator)** to store the user entered value into the declared variable.

scanf

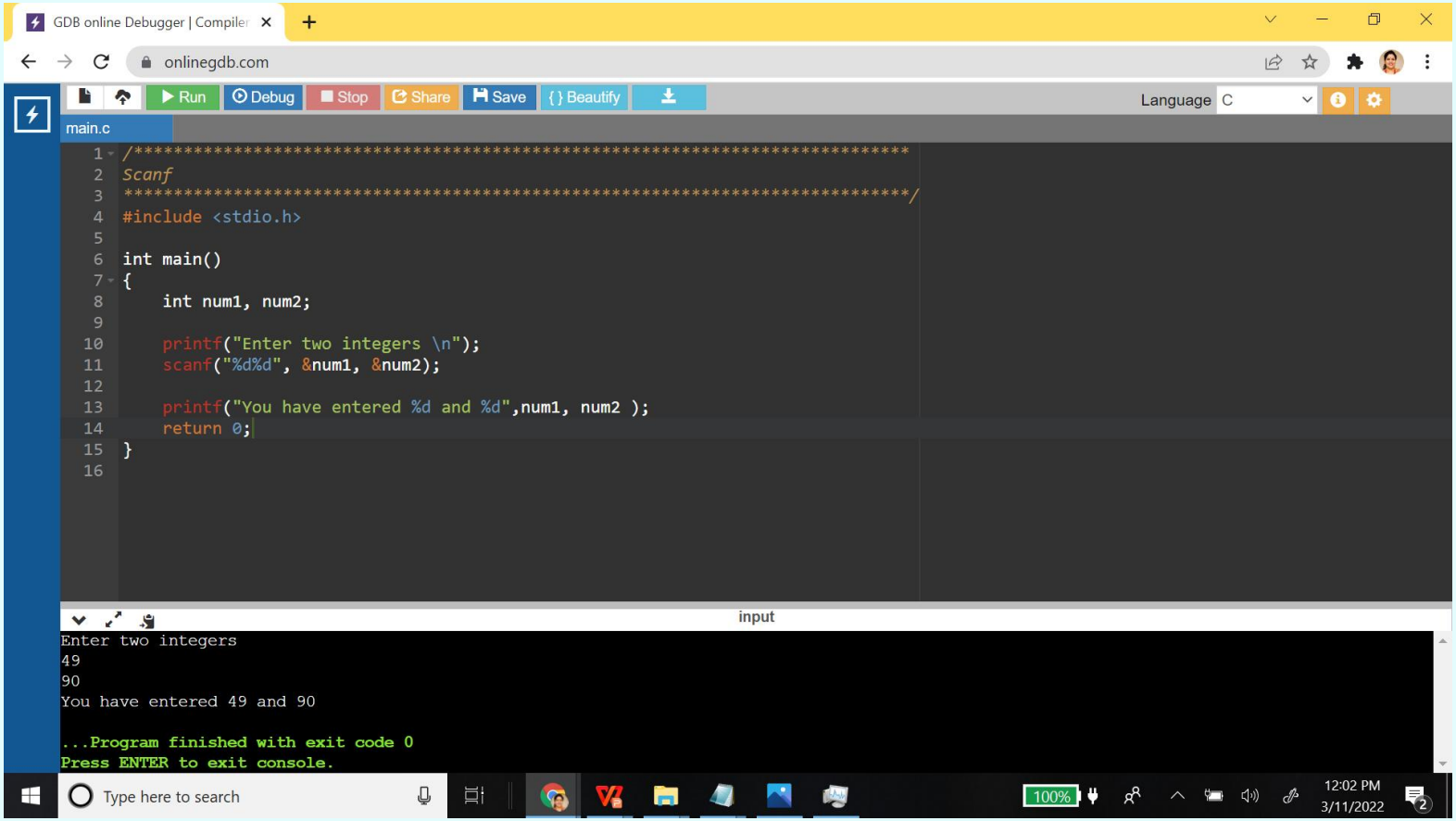
```c
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Take two numbers



format specifier

Data Type	Format Specifier
`int`	`%d`
`char`	`%c`
`float`	`%f`
`double`	`%lf`
`short int`	`%hd`
`unsigned int`	`%u`
`long int`	`%li`
`long long int`	`%lli`

WAP to perform the addition of two integers & display the result.

PROGRAM CODE

```
#include<stdio.h>
int main()
{
  int a, b, c;
  printf("\nEnter two numbers to add :");
  scanf("%d%d",&a,&b);
  c = a + b;
  printf("\nThe addition of %d and %d is %d", a,b,c);
  return 0;
}
```

INPUT/OUTPUT

RUN-1

```
Enter two numbers to add: 4 6
The addition of 4 and 6 is 10
```

RUN-2

```
Enter two numbers to add: 5 7
The addition of 5 and 7 is 12
```

Specifiers or Modifiers

- In addition, C has four type specifiers or modifiers and three type qualifiers.
  - ✓Each of these type modifiers can be applied to the base type int.
  - ✓The modifiers signed and unsigned can also be applied to the base type char.
  - ✓In addition, long can be applied to double.
  - ✓When the base type is omitted from a declaration, int is assumed.
  - ✓The type void does not have these modifiers.

Specifiers : Data Types

- The specifiers and qualifiers for the data types can be broadly classified into three types:

  ✓Size specifiers— short and long
  ✓Sign specifiers— signed and unsigned
  ✓ Type qualifiers— const, volatile and restrict

	16-bit Machine	32-bit Machine	64-bit Machine
short int	2	2	2
int	2	4	4
long int	4	4	8

	Size (in bytes)	Range
long long int	8	9, 223, 372, 036, 854, 775, 808 to +9, 223, 372, 036, 854, 775, 807
unsigned long int or unsigned long	4	0 to 4, 294, 967, 295
unsigned long long int or unsigned long long	8	0 to +18, 446, 744, 073, 709, 551, 615

Specifiers : Data Types

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

Data Type	Size (bits)	Range	Default Type
char	8	–128 to 127	signed char
unsigned char	8	0 to 255	None
signed char	8	–128 to 127	char
int	16	–32768 to 32767	signed int
unsigned int	16	0 to 65535	unsigned
signed int	16	–32768 to 32767	int
short int	16	–32768 to 32767	short, signed short, signed short int
unsigned short int	16	0 to 65535	unsigned short
signed short int	16	–32768 to 32767	short, signed short, short int
long int	32	–2147483648 to 2147483647	long, signed long, signed long int
unsigned long int	32	0 to 4294967295	unsigned long
signed long int	32	–2147483648 to 2147483647	long int, signed long, long
float	32	3.4E–38 to 3.4E+38	None
double	64	1.7E–308 to 1.7E+308	None
long double	80	3.4E–4932 to 1.1E+4932	None

Qualifiers

- ✓ Modifies the properties of a variable
- ✓ Controls the way variables may be accessed or modified.
- **Const** – read only variable, will never change by the program

  Const float pi = 3.14
- **Volatile** –unexpectedly change by events outside the program (directly linked with component)

  if global variable's address is passed to clock routine of the operating system to store the system time, the value in this address keep on changing without any assignment by the program. These variables are named as volatile variable.

2. WAP to perform the addition of two integers & display the result.

Program Code

```
#include<stdio.h>
int main()
{
    int a, b, c;
    printf("\nEnter two numbers to
    add :");
    scanf("%d%d",&a,&b);
    c = a + b;
    printf("\nThe addition of %d and
    %d is %d", a,b,c);
    return 0;
}
```

Q1. Declare some int variable and store different values and display them
a = 20, b = 'A', c = 2.09

Q2. Input 2 Integer values and display the addition

Q3.Write programs Input 2 int values and display the addition(+), subtraction(-), multiplication(*) division(/) and modular division(%)

1. Program 1 - WAP to calculate area and the perimiter of a circle.
2. Program 2 - Calculate the area of a rectangle (take two values as input = lenth, Width)

WAP to calculate area of a circle.

PROGRAM CODE

```
#include <stdio.h>
int main()
{
    float radius,area;
    printf("\nEnter the radius of  a circle : ");
    scanf("%f",&radius);
    area = 3.14 * radius * radius;
    printf("\nArea of Circle : %f",area);
    return 0;
}
```

Program Statements

- A statement is a syntactic constructions that performs an action when a program is executed. All C program statements are terminated with a semi-colon (;).

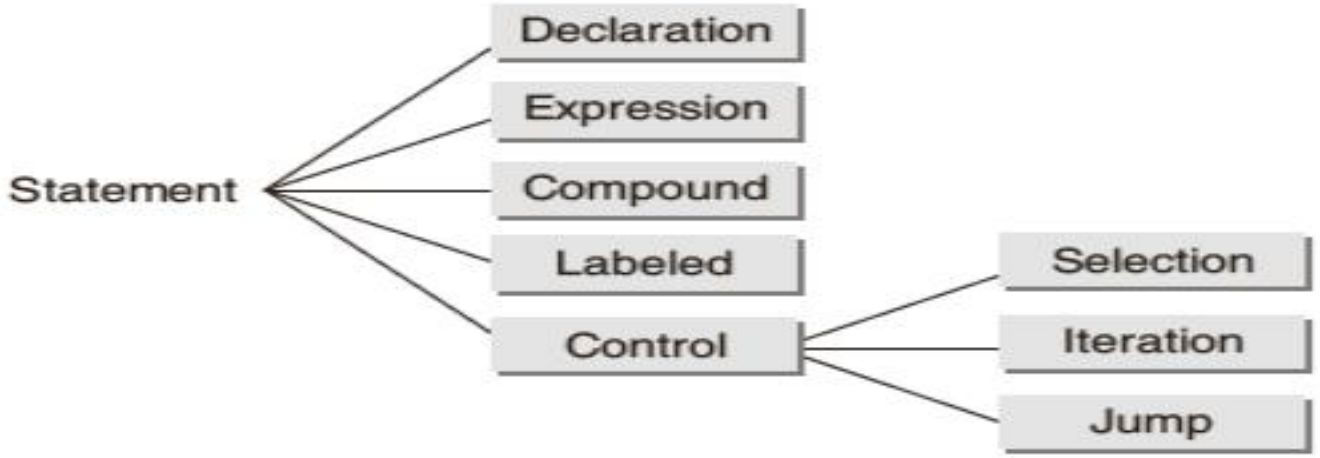


Figure Different types of program statements available in C

Program Statements

- ***Declaration :***It 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.
  - int a;
  - int b;
  - int c;

  Or int a,b,c;

  This line informs the C compiler that it needs to allocate space for 5 integers
- ***Expression statement:*** It is the simplest kind of statement which is no more than an expression followed by a semicolon. An *expression is a sequence of operators and* operands that specifies computation of a value . Example :x = 4

Program Statements

- ***Compound statement is a sequence of statements that*** may be treated as a single statement in the construction of larger statements.
- A compound statement (also called a "block") typically appears as the body of another statement, such as the **if** statement
- ```
  if ( i > 0 )
   {
     line[i] = x;
     x++;
     i--;
   }
  ```
  ```

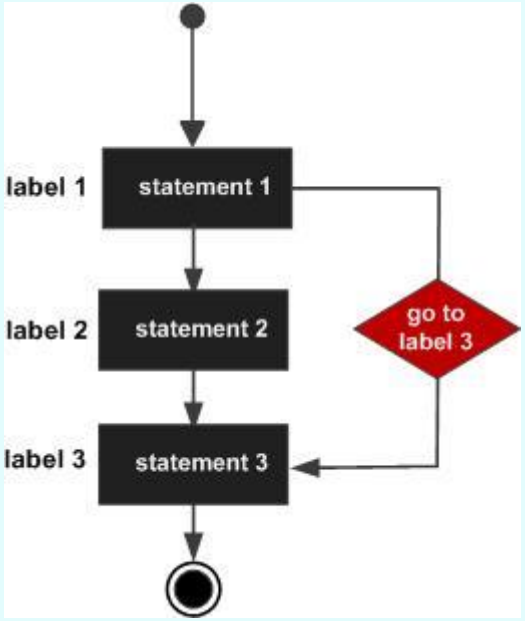
Program Statements

- ***Labelled statements can be used to mark any statement so*** that control may be transferred to the statement by *switch* statement

goto label;

.. .

label: statement;



Program Statements

- ***Control statement is a statement whose execution results*** in a choice being made as to which of two or more paths should be followed. In other words, the control statements determine the 'flow of control' in a program.

  - ✓ **Selection statements allow a program to select a particular** execution path from a set of one or more alternatives. Various forms of the if..else statement belong to this category.
  - ✓ **Iteration statements are used to execute a group of one** or more statements repeatedly. "while, for, and do..while" statements falls under this group.
  - ✓ **Jump statements cause an unconditional jump to some** other place in the program. Goto statement falls in this group

Program Statements(control)

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

```
while ( expression )
{
      Single statement
      or
      Block of statements;
 }

for( expr1; expr2; expr3)
{
      Single statement
      or Block of statements;
}
```

Program Statements(control)

```
do
   {
      Single statement
      or Block of statements;
    }while(expression);
```

Key Words

- Compiler vendors (like Microsoft, Borland ,etc.) provide their own keywords apart from the ones mentioned below. These include extended keywords like **near, far, asm, etc.**

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

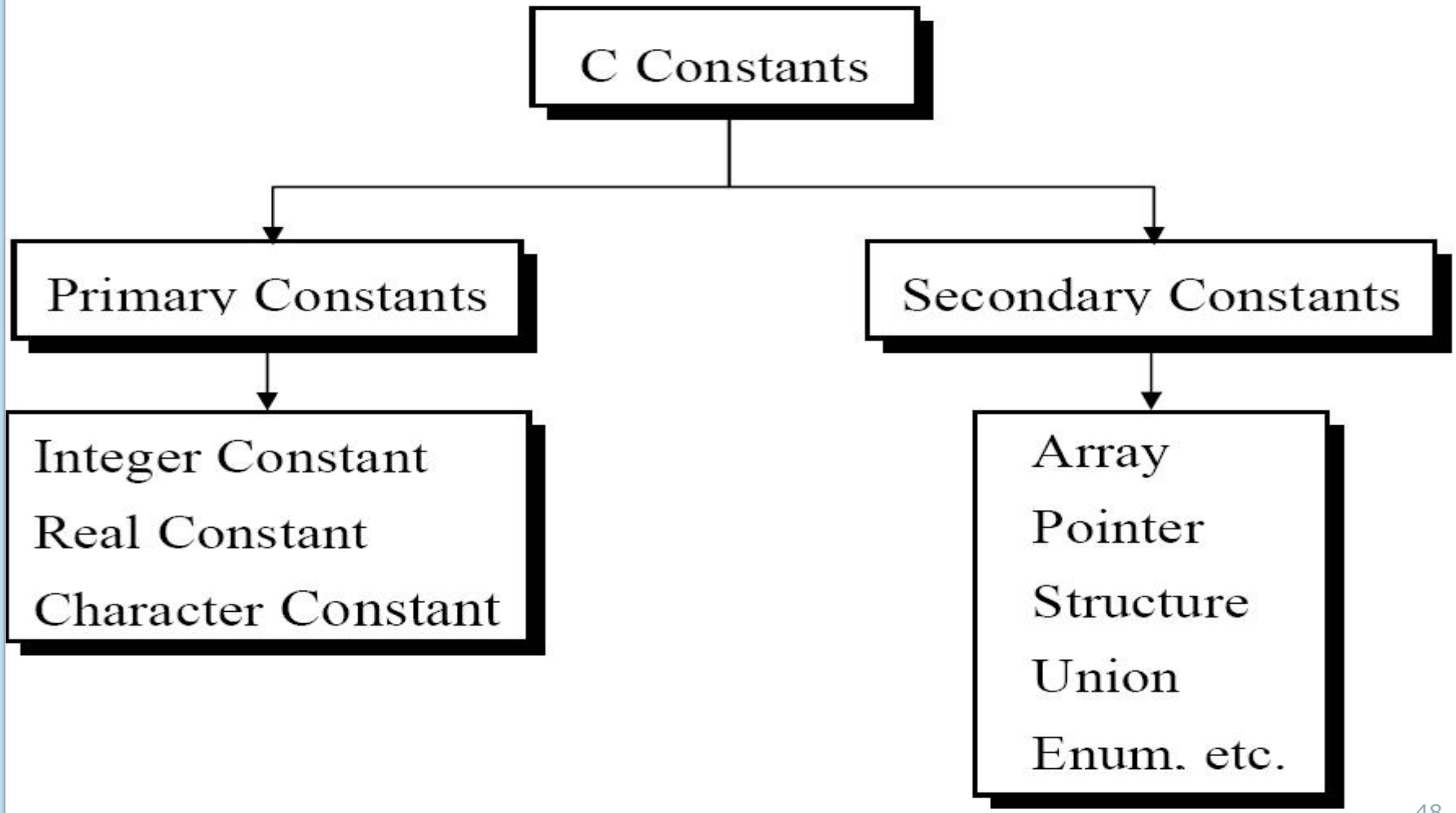
Constants

- A constant is an explicit data value written by the programmer. Thus, it is a value known to the compiler at compiling time.
- In ANSI C, a decimal integer constant is treated as an unsigned long if its magnitude exceeds that of the signed long. An octal or hexadecimal integer that exceeds the limit of int is taken to be unsigned; if it exceeds this limit, it is taken to be long; and if it exceeds this limit, it is treated as an unsigned long.
- An integer constant is regarded as unsigned if its value is followed by the letter ‘u’ or ‘U’, e.g.,0x9999u;

Specifications of Different Constants

Type	Specification	Example
Decimal	nil	50
Hexadecimal	Preceded by 0x or 0X	0x10
Octal	Begins with 0	010
Floating constant	Ends with f/F	123.0f
Character	Enclosed within single quote	'A' 'o'
String	Enclosed within double quote	"welcome"
Unsigned integer	Ends with U/u	37 u
Long	Ends with L/l	37 L
Unsigned long	Ends with UL/w	37 UL

Types of C Constant



Rules for Constructing Const

Constructing Integer Const

- Must have at least one digit.
- Must not have a decimal point.
- Can be either positive or negative
- If no sign precedes an integer constant it is assumed to be positive.
- No commas or blanks are allowed

Ex.: 426
+782
-8000
-7605

Constructing Real Const in fractional form

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form. Following rules must be observed while constructing real constants expressed in fractional form:

- Must have at least one digit.
- Must have a decimal point.
- Could be either positive or negative.
- Default sign is positive.
- No commas or blanks are allowed.

Ex.: +325.34
426.0
-32.76
-48.5792

Rules for Constructing Const

Constructing Real Const in exponential form

The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large. It however doesn't restrict us in any way from using exponential form of representation for other real constants.

In exponential form of representation, the real constant is represented in two parts. The part appearing before 'e' is called **mantissa**, whereas the part following 'e' is called **exponent**.

Following rules must be observed while constructing real constants expressed in exponential form:

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa part may have a positive or negative sign.
- Default sign of mantissa part is positive.
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

Ex.: +3.2e-5
4.1e8
-0.2e+3
-3.2e-5

Rules for Constructing Const

Constructing Character Const

- Is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example, 'A' is a valid character constant whereas 'A' is not.

- The maximum length can be 1 character.

Ex.: 'A'
'I'
'5'
'='

```
Note : 3.2e^-5 = 3.2(10)^-5
3.14159 /* Legal */
314159E-5L /* Legal */
510E /* Illegal: incomplete exponent */
210f /* Illegal: no decimal or exponent
*/ .e55 /* Illegal: missing integer or
fraction */
```

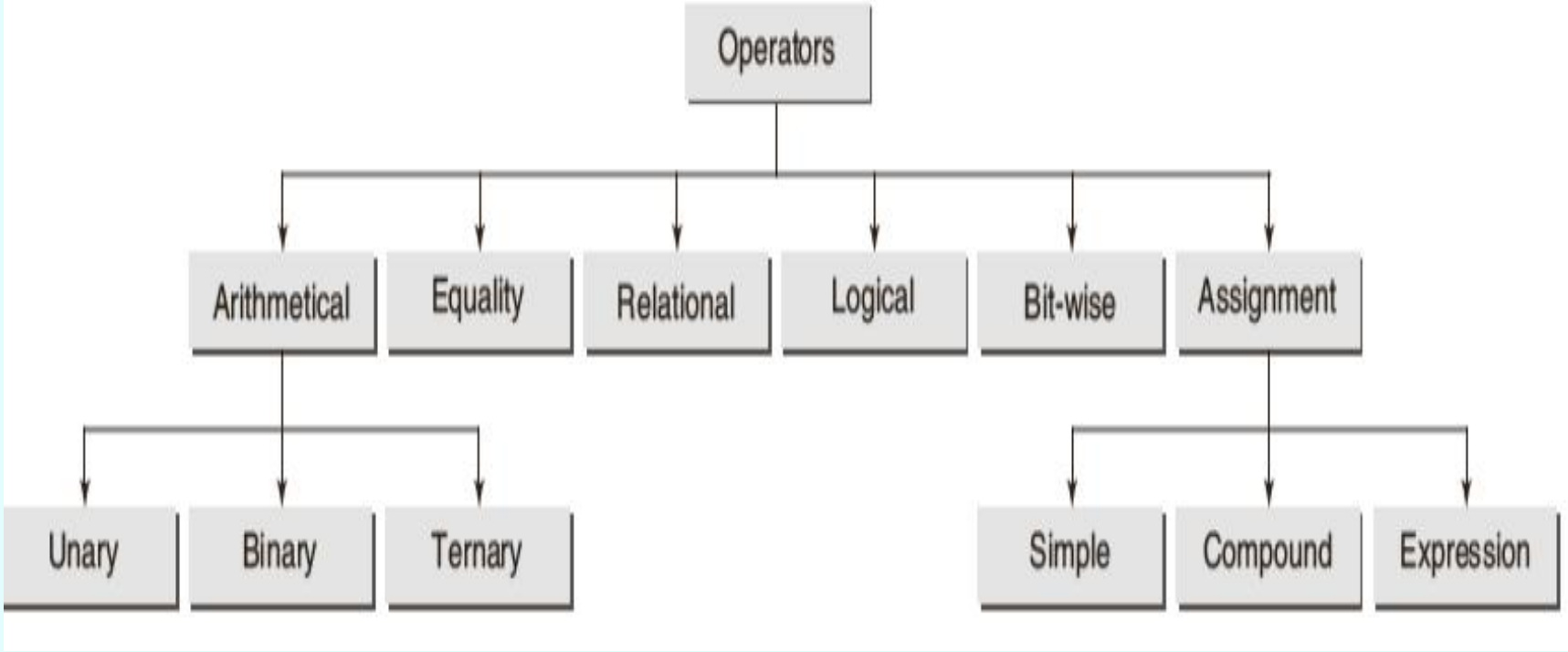
Constant example

```
#include <stdio.h>
void main()
{
const int height = 100; /*int constant*/
const float number = 3.14; /*Real constant*/
const char letter = 'A'; /*char constant*/
const char letter_sequence[10] = "ABC"; /*string constant*/
const char backslash_char = '\?'; /*special char cnst*/

printf("value of height :%d \n", height );
printf("value of number : %f \n", number );
printf("value of letter : %c \n", letter );
printf("value of letter_sequence : %s \n", letter_sequence);
printf("value of backslash_char : %c \n", backslash_char);
}
```

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

Classification : Operations in C



Different Operators

Type of operator	Operator symbols with meanings
Arithmetical	Unary + (Unary) — (Unary) ++ Increment —— Decrement
	Binary + Addition — Subtraction * Multiplication / Division % Modulas
	Ternary ?: Discussed later on
Assignment	Simple Assignment =
	Compound Assignment +=, -=, *=, /=, %=, &=, ^=, |=
	Expression Assignment A= 5+(b=8 + (c=2)) -4
Relational	>, <, >=, <=
Equality	= = (Equal to) != (Not equal to)
Logical	&& (Logical AND) || (Logical OR) ! (Logical NOT)
Bitwise	& (Bitwise AND) | (Bitwise OR) ~ (Complement) ^ (Exclusive OR) >> (Right Shift) << (Left Shift)
Others	, (Comma) * (indirection), . (membership operator) -> (membership operator)

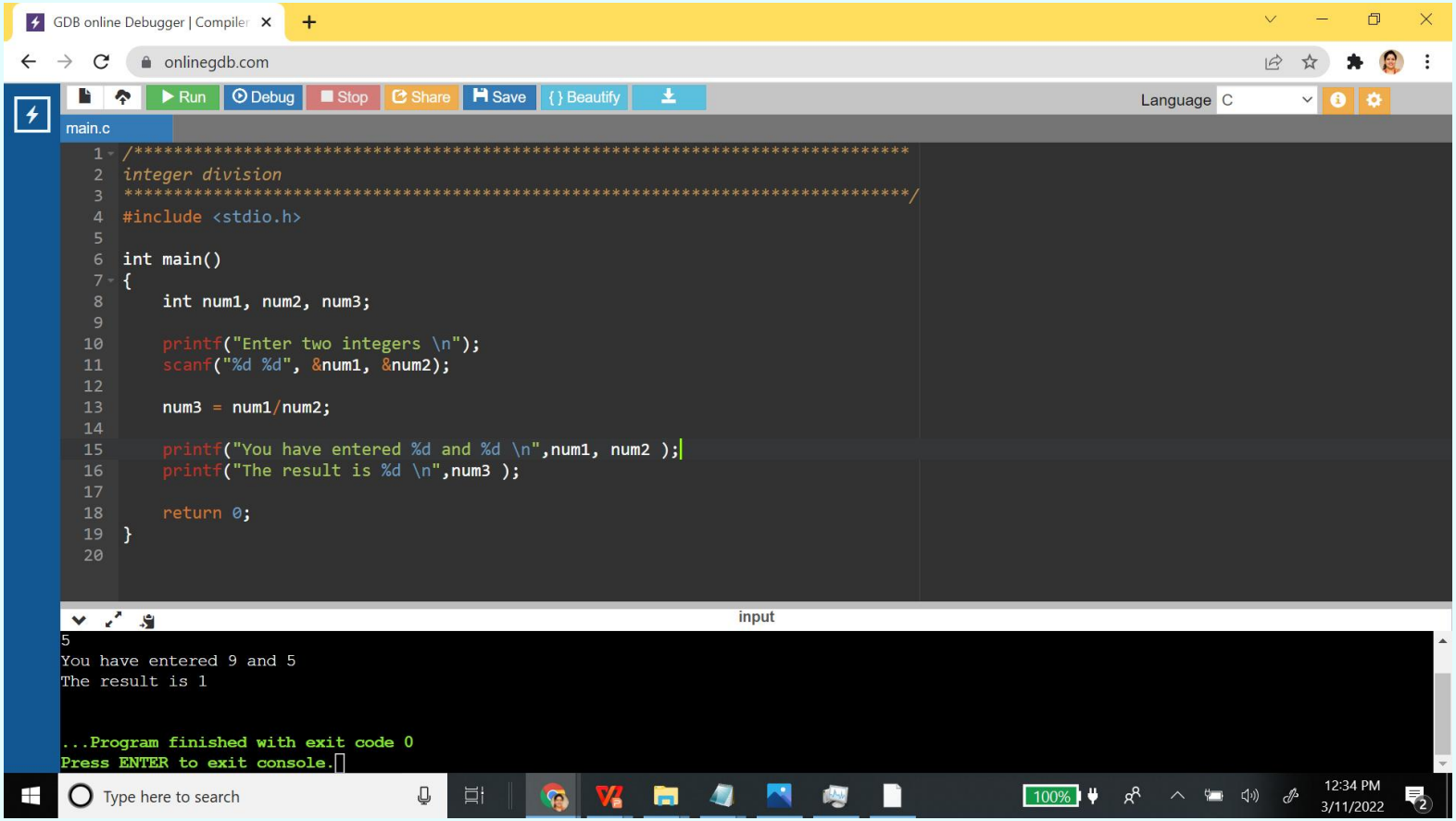
A) Arithmetic Operators

- There are three types of arithmetic operators in C:binary,unary, and ternary.
- **A-1) Binary operators:** C provides five basic arithmetic binary operators.

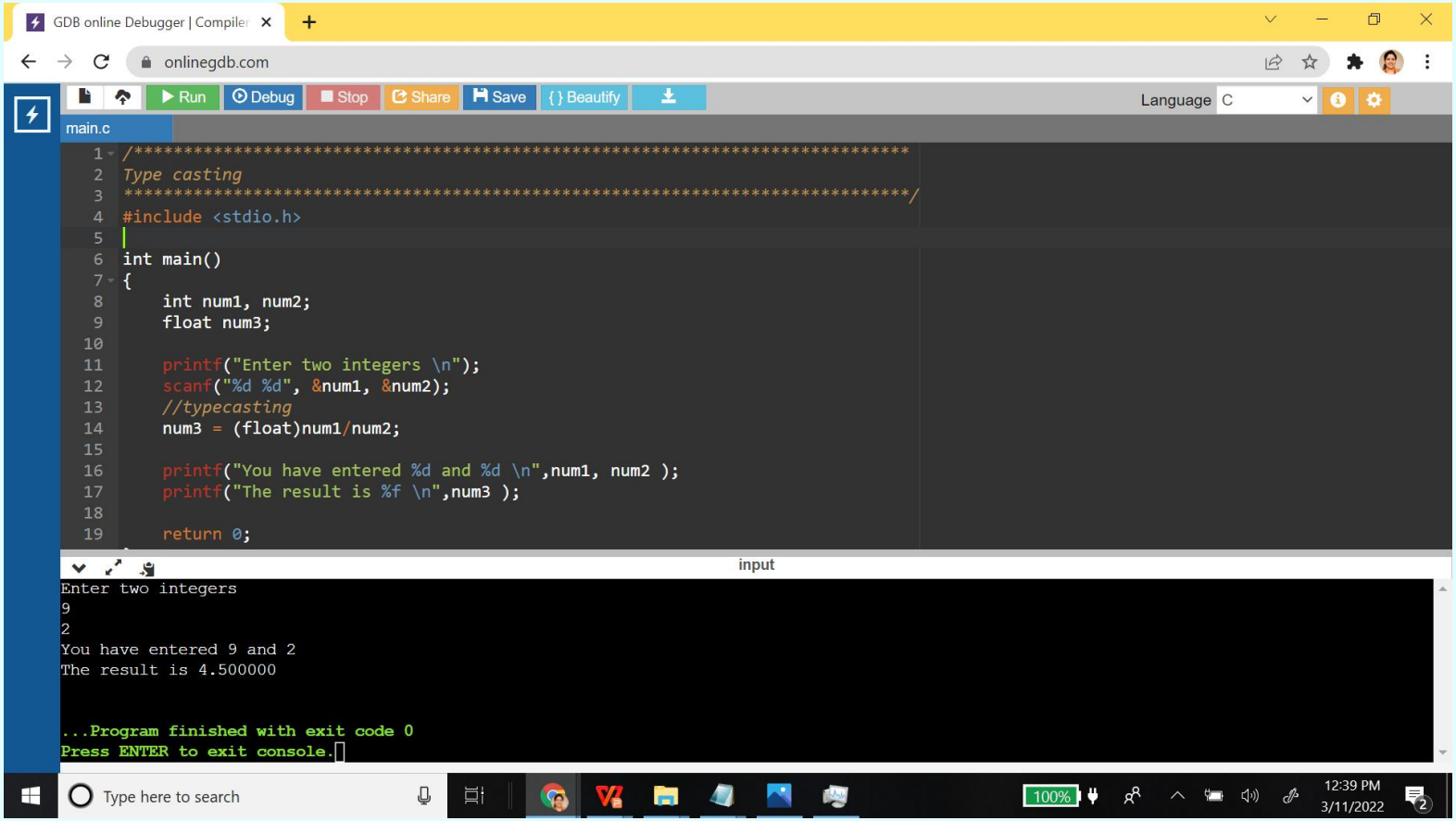
Operator	Name	Example
+	Addition	`12 + 4.9 /* gives 16.9*/`
-	Subtraction	`3.98 – 4 /* gives –0.02*/`
*	Multiplication	`2 * 3.4 /* gives 6.8  */`
/	Division	`9 / 2.0 /* gives 4.5 */`
%	Remainder	`13 % 3 /* gives 1  */`

  - The mod operator (%) will carry the sign of the numerator

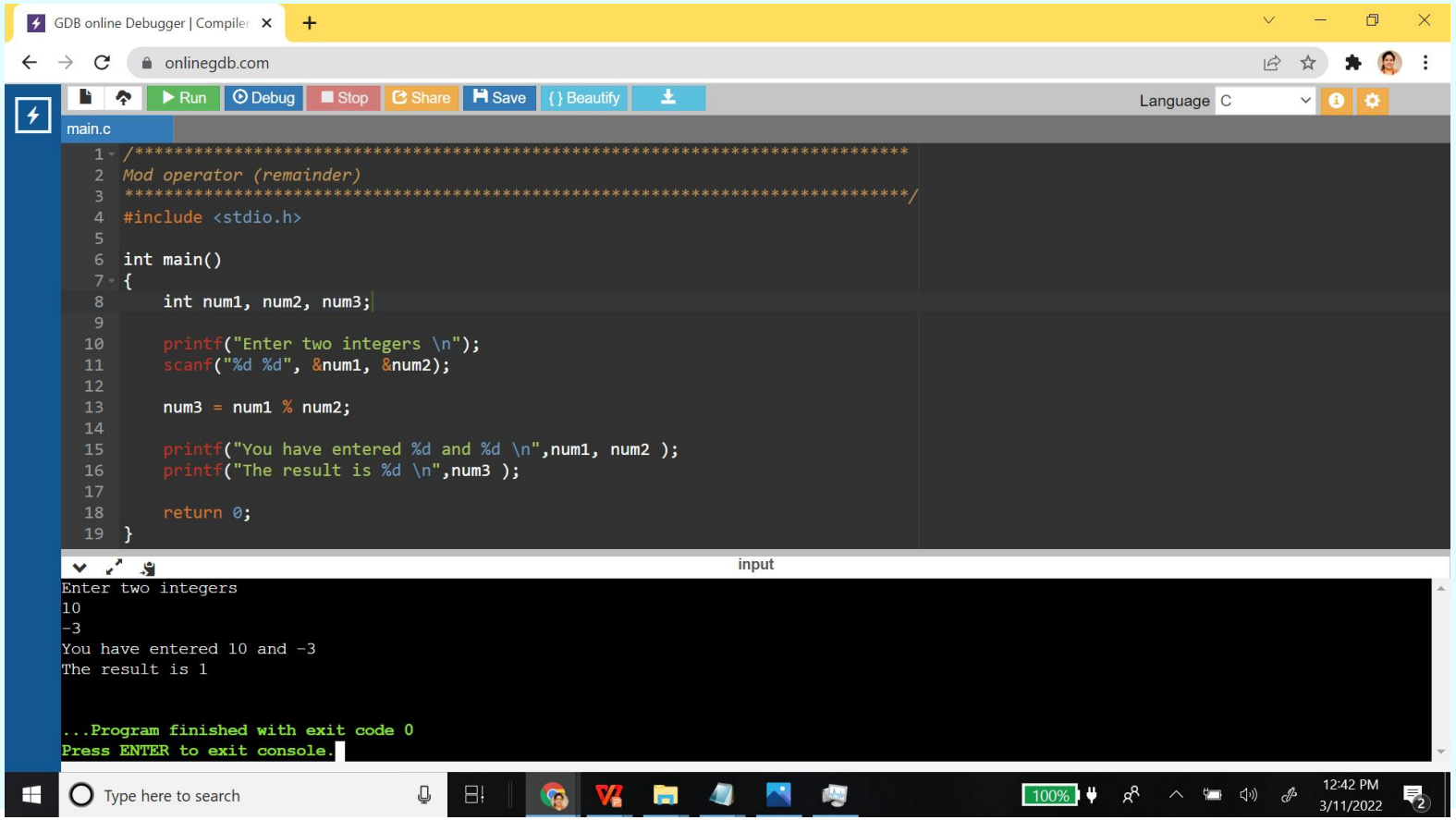
integer division



type casting



mod operator - check sign



- Program 1 : Addition of two float numbers, taking from user input.
- Program 2 : Division of two integers
- Program 3 : Division of two floats
- Program 4 : Modulus function

A-2) Unary Operators

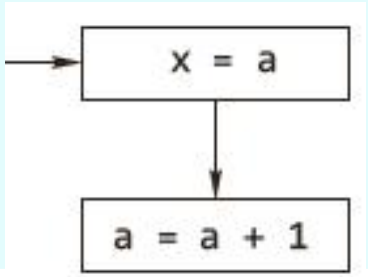
- ***Unary operators:*** The **unary operators** operate on a single operand and following are the examples of **Unary operators**:.
- ***Unary increment and decrement operators*** '++' and '--' operators increment or decrement the value in a variable by 1.
- ***Basic rules for using ++ and – – operators:***

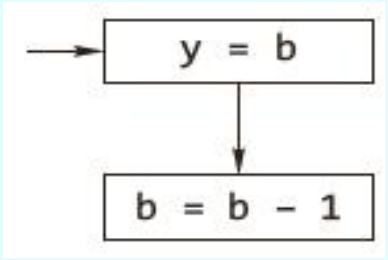
  ✓The operand must be a variable but not a constant or an expression.
  ✓ The operator ++ and -- may precede or succeed the operand.

Postfix

Postfix:

- **(a) x = a++;**
  - ✓ First action: store value of a in memory location for variable x.
  - ✓ Second action: increment value of a by 1 and store result in memory location for variable a.
- **(b) y = b−−;**
  - ✓ First action: put value of b in memory location for variable y.
  - ✓ Second action: decrement value of b by 1 and put result in memory location for variable b.

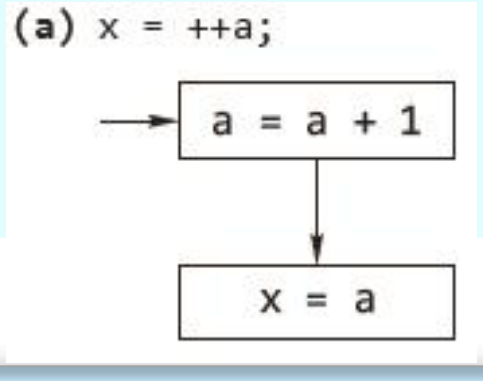


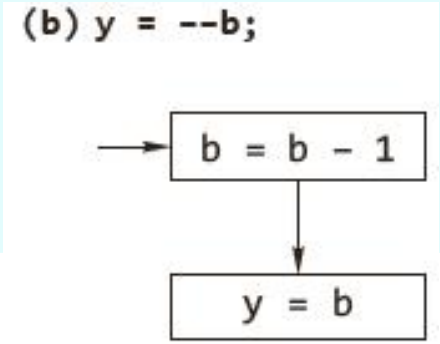


Prefix

Prefix :

- **(a) x = ++a;**
  - ✓*First action: increment value of a* by 1 and store result in memory location for variable a.
  - ✓*Second action: store value of a* in memory location for variable x.
- **(b) y = --b;**
  - ✓*First action: decrement value of b* by 1 and put result in memory location for variable b.
  - ✓*Second action: put value of b in memory location for variable y.*





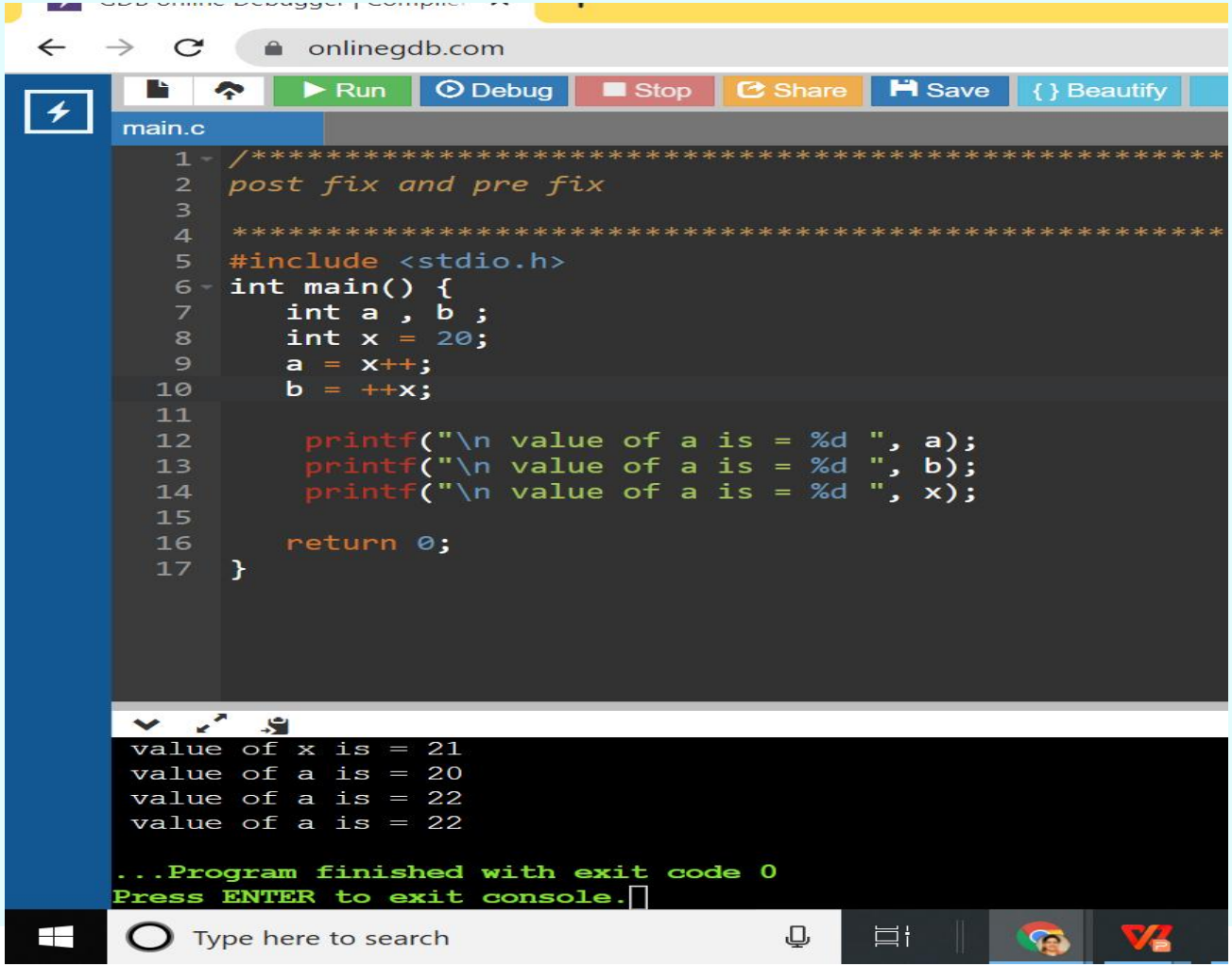
prefix and post fix

main.c

```c
/*****************************************************************************
post fix and pre fix

*****************************************************************************/
#include <stdio.h>
int main() {
   int var1 = 5, var2 = 5;

   // 5 is displayed
   // Then, var1 is increased to 6.
   printf("%d\n", var1++);

   // var2 is increased to 6
   // Then, it is displayed.
   printf("%d\n", ++var2);

   return 0;
}
```

input

```
5
6
```


onlinegdb.com
Run
Debug
Stop
Share
Save
{ } Beautify
main.c
/***
post fix and pre fix
****
#include <stdio.h>
int main() {
int a , b ;
int x = 20;
a = x++;
b = ++x;
printf("\n value of a is = %d ", a);
printf("\n value of a is = %d ", b);
printf("\n value of a is = %d ", x);
return 0;
}
value of x is = 21
value of a is = 20
value of a is = 22
value of a is = 22
...Program finished with exit code 0
Press ENTER to exit console.
Type here to search

B) Relational Operators

- C provides six relational operators for comparing numeric quantities. Relational operators evaluate to 1, representing the *true outcome, or* 0, representing the *false outcome.*

Operator	Action	Example
==	Equal	5 == 5 /* gives 1 */
!=	Not equal	5 != 5 /* gives 0 */
<	Less than	5 < 5.5 /* gives 1 */
<=	Less than or equal	5 <= 5 /* gives 1 */
>	Greater than	5 > 5.5 /* gives 0 */
>=	Greater than or equal	6.3 >= 5 /* gives 1 */

WAP to convert given paisa into its equivalent rupee and paisa as per the following format.
Ex. 550 paisa = 5 Rupee and 50 paisa

PROGRAM CODE

```
#include<stdio.h>
int main()

{
        int p,p1,r;
        printf("\nEnter paisa = ");
        scanf("%d",&p);
        r=p/100;
        p1=p%100;
        printf("\n%d paisa = %d rupees and %d paisa",p,r,p1);
        return 0;
}
```

INPUT/OUTPUT

RUN-1

Enter paisa = 2550

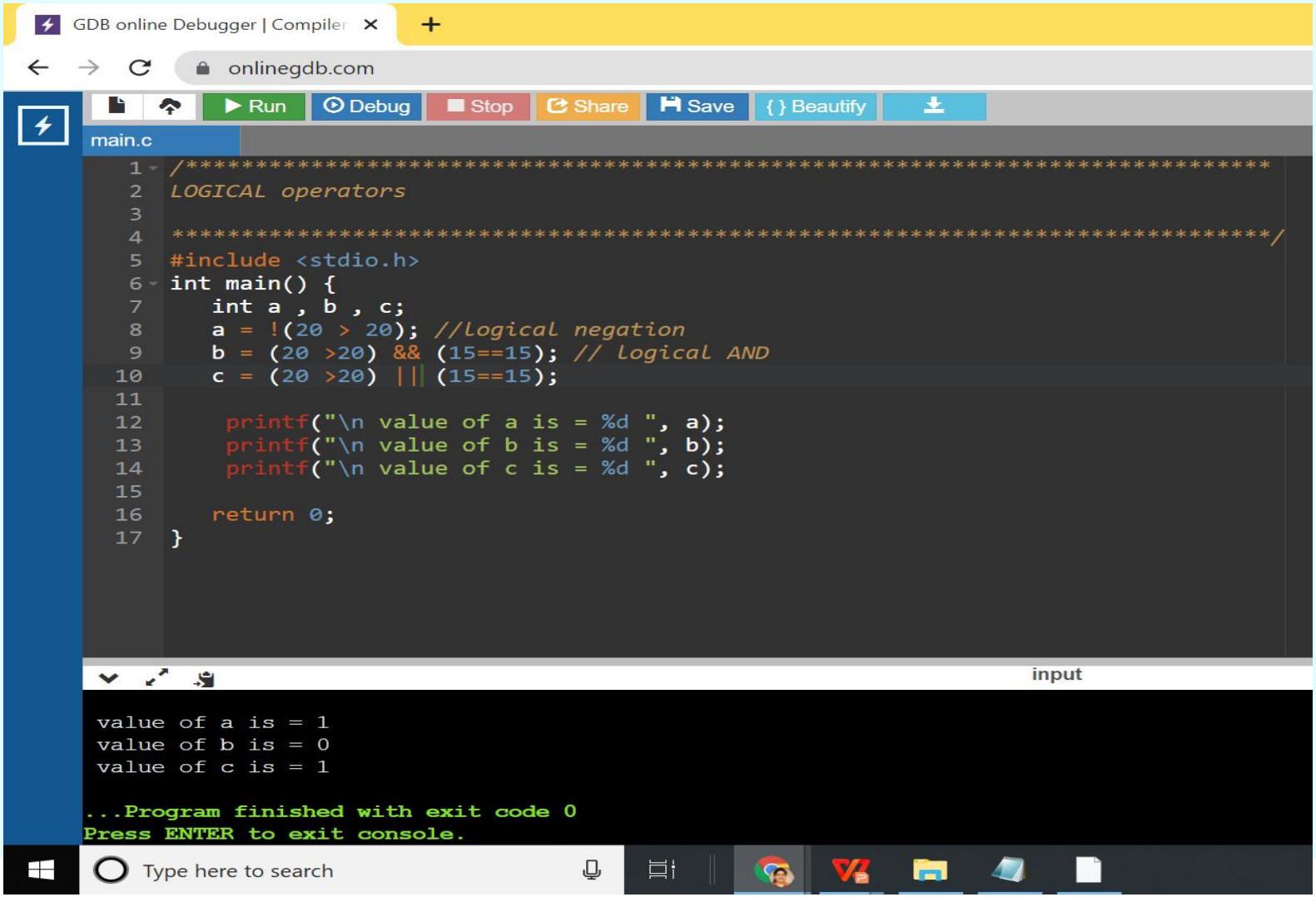
PROGRAM CODE

```
#include<stdio.h>
int main()
{
    long sec1, sec2, hr, min, t;
    printf("\nEnter time in seconds: ");
    scanf("%ld", &sec1);
    hr = sec1/3600;
    t = sec1%3600;
    min = t/60;
    sec2 = t%60;
    printf("\n\n %ld second= %ld Hour %ld Minute and %ld  Second",sec1, hr, min,sec2);
    return 0;
}
```

C) Logical Operators

- C provides three logical operators for forming logical expressions. Like the relational operators, logical operators evaluate to 1 or 0.
  - ✓Logical negation is a unary operator that negates the logical value of its single operand. If its operand is non-zero, it produces 0, and if it is 0, it produces 1.
  - ✓Logical AND produces 0 if one or both its operands evaluate to 0. Otherwise, it produces 1 (both true in this case of 1).
  - ✓Logical OR produces 0 if both its operands evaluate to 0. Otherwise , it produces 1 (if any one is true).

Operator	Action	Example	Result
!	Logical Negation	!(5 == 5)	0
&&	Logical AND	5 < 6 && 6 < 6	0
||	Logical OR	5 < 6 || 6 < 5	1


GDB online Debugger | Compiler
onlinegdb.com
Run
Debug
Stop
Share
Save
{ } Beautify
main.c
/***
LOGICAL operators
***/
#include <stdio.h>
int main() {
int a , b , c;
a = !(20 > 20); //logical negation
b = (20 >20) && (15==15); // logical AND
c = (20 >20) || (15==15);
printf("\n value of a is = %d ", a);
printf("\n value of b is = %d ", b);
printf("\n value of c is = %d ", c);
return 0;
}
input
value of a is = 1
value of b is = 0
value of c is = 1
...Program finished with exit code 0
Press ENTER to exit console.
Type here to search

Bitwise Operators

- C provides six bitwise operators for manipulating the individual bits in an integer quantity . Bitwise operators expect their operands to be integer quantities and treat them as bit sequences.

  - ✓Bitwise *negation is a unary operator that complements the bits in* its operands.
  - ✓Bitwise AND compares the corresponding bits of its operands and produces a 1 when both bits are 1, and 0 otherwise.
  - ✓ Bitwise OR compares the corresponding bits of its operands and produces a 0 when both bits are 0, and 1 otherwise.
  - ✓Bitwise *exclusive or compares the corresponding bits of its operands and* produces a 0 when both bits are 1 or both bits are 0, and 1 otherwise.

Bitwise Operators

A = 0011 1100 (60)
B = 0000 1101 (13)

OP	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 0000 1111

Bitwise Operators

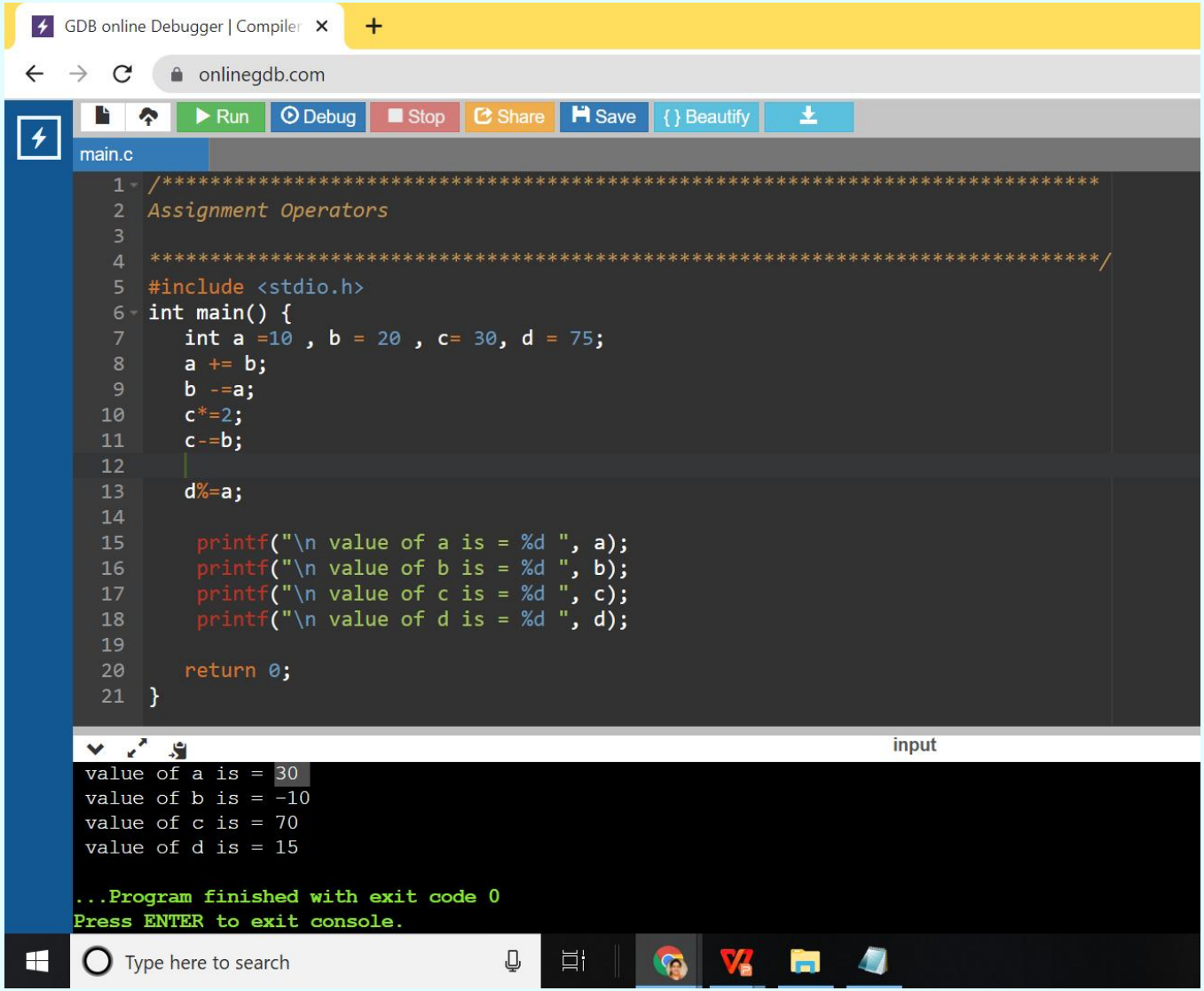
Operator	Action	Example
~	Bitwise Negation	~'\011' /* gives '\066' */
&	Bitwise AND	'\011' & '\027' /* gives '\001' */
|	Bitwise OR	'\011' | '\027' /* gives '\037' */
^	Bitwise Exclusive OR	'\011' ^ '\027' /* gives '\036' */
<<	Bitwise Left Shift	'\011' << 2 /* gives '\044' */
>>	Bitwise Right Shift	'\011' >> 2 /* gives '\002' */

How the bits are calculated

Example	Octal value	Bit sequence							
x	011	0	0	0	0	1	0	0	1
y	027	0	0	0	1	0	1	1	1
~x	366	1	1	1	1	0	1	1	0
x & y	001	0	0	0	0	0	0	0	1
x | y	037	0	0	0	1	1	1	1	1
x ^ y	036	0	0	0	1	1	1	1	0
x << 2	044	0	0	1	0	0	1	0	0
x >> 2	002	0	0	0	0	0	0	1	0

D) Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
−=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C −= A is equivalent to C = C − A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A


GDB online Debugger | Compiler
onlinegdb.com
Run
Debug
Stop
Share
Save
{ } Beautify
main.c
/***
Assignment Operators
***/
#include <stdio.h>
int main() {
int a =10 , b = 20 , c= 30, d = 75;
a += b;
b -=a;
c*=2;
c-=b;
d%=a;
printf("\n value of a is = %d ", a);
printf("\n value of b is = %d ", b);
printf("\n value of c is = %d ", c);
printf("\n value of d is = %d ", d);
return 0;
}
input
value of a is = 30
value of b is = -10
value of c is = 70
value of d is = 15
...Program finished with exit code 0
Press ENTER to exit console.
Type here to search

Assignment Operators

Operator	Description	Example
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

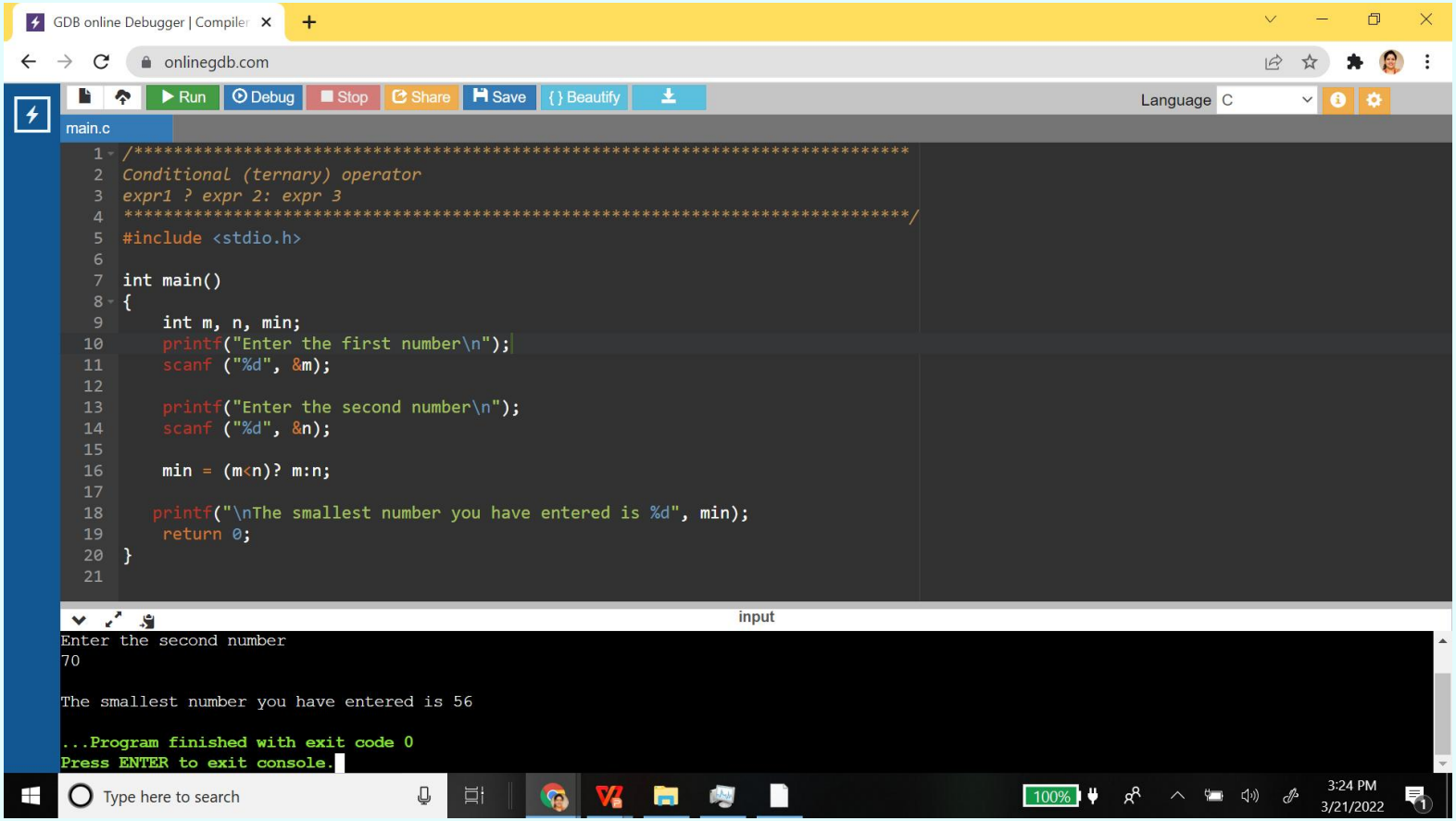
Example

```
#include <stdio.h>
main()
{
int a = 21;
int c ;
c = a; printf("Line 1 - = Operator Example, Value of c = %d\n", c );
c += a; printf("Line 2 - += Operator Example, Value of c = %d\n", c );
c -= a; printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
c *= a; printf("Line 4 - *= Operator Example, Value of c = %d\n", c );
 c /= a; printf("Line 5 - /= Operator Example, Value of c = %d\n", c );
c = 200;
c %= a; printf("Line 6 - %= Operator Example, Value of c = %d\n", c );
}
Line 1 - = Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - %= Operator Example, Value of c = 11
```

Conditional Operators

- The conditional operator has three expressions (ternary).
  - ✓ It has the general form
  - ✓**expression1 ? expression2 : expression3**
  - ✓First, expression1 is evaluated; it is treated as a logical condition.
  - ✓If the result is non-zero, then expression2 is evaluated and its value is the final result. Otherwise, expression3 is evaluated and its value is the final result.
- For example,int m = 1, n = 2, min;

  min = (m < n ? m : n); /* min is assigned a value 1 */
- In the above example, because m is less than n, m<n expression evaluates to be true, therefore, min is assigned the value m, i.e., 1.


GDB online Debugger | Compiler
onlinegdb.com
Run
Debug
Stop
Share
Save
{} Beautify
Language C
main.c
/***
Conditional (ternary) operator
expr1 ? expr 2: expr 3
***/
#include <stdio.h>
int main()
{
int m, n, min;
printf("Enter the first number\n");
scanf ("%d", &m);
printf("Enter the second number\n");
scanf ("%d", &n);
min = (m<n)? m:n;
printf("\nThe smallest number you have entered is %d", min);
return 0;
}
input
Enter the second number
70
The smallest number you have entered is 56
...Program finished with exit code 0
Press ENTER to exit console.
Type here to search
100%
3:24 PM
3/21/2022

Conditional Operators

```
#include <stdio.h>

 main()
{
int a , b;
a = 10;
printf( "Value of b is %d\n", (a == 1) ? 20: 30 );
printf( "Value of b is %d\n", (a == 10) ? 20: 30 );
}
```

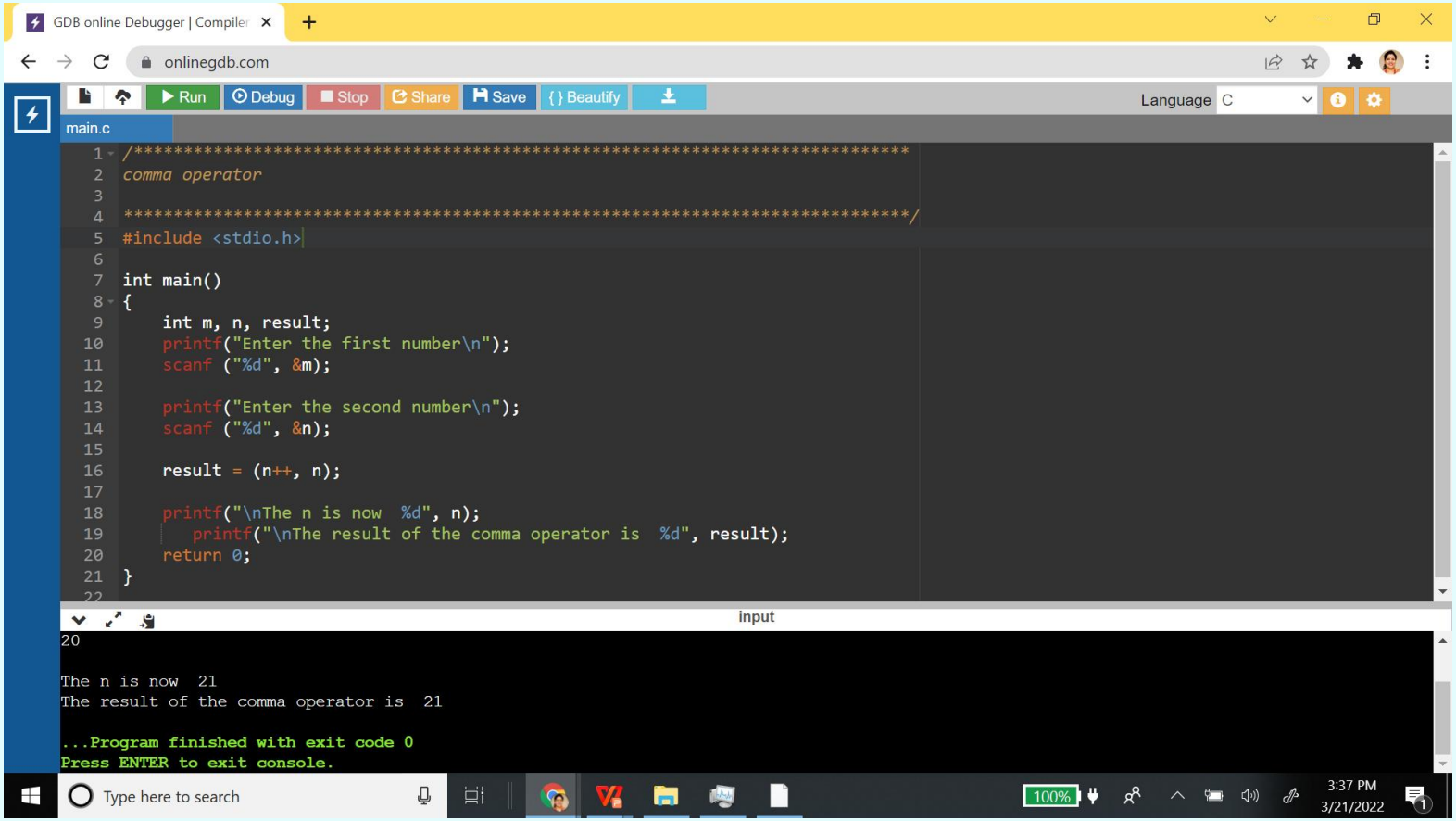
This will produce following result:

```
Value of b is 30
 Value of b is 20
```

Comma Operators

- This operator allows the evaluation of multiple expressions, separated by the comma, from left to right in order and the evaluated value of the rightmost expression is accepted as the final result. The general form of an expression using a comma operator is
- Expression M = (expression1, expression2, …,expression N);
- where the expressions are evaluated strictly from left to right and their values discarded, except for the last one, whose type and value determine the result of the overall expression.

```
void main()
 { int num1 = 1, num2 = 2;                    /*separator*/
int res; res = (num1, num2);                  /* operator*/
printf("%d", res); }  .... Gives output as num2
```


GDB online Debugger | Compiler
onlinegdb.com
Run
Debug
Stop
Share
Save
{} Beautify
Language C
main.c
/***
comma operator
***/
#include <stdio.h>
int main()
{
int m, n, result;
printf("Enter the first number\n");
scanf ("%d", &m);
printf("Enter the second number\n");
scanf ("%d", &n);
result = (n++, n);
printf("\nThe n is now %d", n);
printf("\nThe result of the comma operator is %d", result);
return 0;
}
input
20
The n is now 21
The result of the comma operator is 21
...Program finished with exit code 0
Press ENTER to exit console.
Type here to search
100%
3:37 PM
3/21/2022

Sizeof Operators

- C provides a useful operator, sizeof, for calculating the size of any data item or type. It takes a single operand that may be a type name (e.g., int) or an expression (e.g.,100) and returns the size of the specified entity in bytes .The outcome is totally machine-dependent.
  - **For example:**

```
#include <stdio.h>
int main()
{
   printf("char size = %d bytes\n", sizeof(char));
   printf("short size = %d bytes\n", sizeof(short));
   printf("int size = %d bytes\n", sizeof(int));
   printf("long size = %d bytes\n", sizeof(long));
   printf("float size = %d bytes\n", sizeof(float));
   printf("double size = %d bytes\n", sizeof(double));
   printf("1.55 size = %d bytes\n", sizeof(1.55));
   printf("1.55L size = %d bytes\n", sizeof(1.55L));
   printf("HELLO size = %d bytes\n", sizeof("HELLO"));
return 0;
}
```

When run, the program will produce the following output (on the programmer's PC):

```
char size = 1 bytes
short size = 2 bytes
int size = 2 bytes
long size = 4 bytes
float size = 4 bytes
double size = 8 bytes
1.55 size = 8 bytes
1.55L size = 10 bytes
HELLO size = 6 bytes
```

Expression Evolution : Precedence & Associativity

- **Precedence of operators**
- If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.
- In C, precedence of

Arithmetic operators(*, %, /, +, -)
Relational operators(==, !=, >, <, >=, <=)
Logical operators(&&, || and !).

- **Example of precedence**
- (1 > 2 + 3 && 4)
- This expression is equivalent to: ((1 > (2 + 3)) && 4)
- i.e, (2 + 3) executes first resulting into 5 then, first part of the expression (1 > 5) executes resulting into 0 (false) then, (0 && 4) executes resulting into 0 (false)
- **Output - 0**

Expression Evolution : Precedence & Associativity

Associativity of operators

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute.

Example of associativity

1 == 2 != 3

Here, operators == and != have same precedence.

The associativity of both == and != is left to right, i.e, the expression on the left is executed first and moves towards the right.

Thus, the expression above is equivalent to :

((1 == 2) != 3)

i.e, (1 == 2) executes first resulting into 0 (false)

then, (0 != 3)

executes resulting into 1 (true)

Expression Evolution : Precedence & Associativity

- Evaluation of an expression in C is very important to understand. Unfortunately there is no 'BODMAS' rule in C language as found in algebra.

The precedence of operators determines the order in which different operators are evaluated when they occur in the same expression. Operators of higher precedence are applied before operators of lower precedence.

Operaors	Associativity
( ) [ ] . ++ (postfix) -- (postfix)	L to R
++ (prefix) -- (prefix) !~ sizeof(type) + (unary) - (unary) & (address) * (indirection)	R to L
* / %	L to R
+ -	L to R
<< >>	L to R
< <= > >=	L to R
== !=	L to R
&	L to R
^	L to R
|	L to R
&&	L to R
||	L to R
?:	R to L
= += -= *= /= %= >>= <<= &= ^= |=	R to L
, (comma operator)	L to R

Example : Associativity of Operator

```
#include <stdio.h>
int main()
  {
    int a;
    int b = 4;
    int c = 8;
    int d = 2;
    int e = 4;
    int f = 2;
    a = b + c / d + e * f;
                  /* result without parentheses */
    printf("The value of a is = %d \n", a);
    a = (b + c) / d + e * f;
                    /* result with parentheses */
    printf("The value of a is = %d \n", a);
    a = b + c / ((d + e) * f);
           /* another result with parentheses */
    printf("The value of a is = %d \n", a);
    return 0;
}
```

Output:

```
The value of a is = 16
The value of a is = 14
The value of a is = 6
```

L-Values & R-Values

- An l- value is an expression to which a value can be assigned.
- An r- value can be defined as an expression that can be assigned to an l- value.
- The l- value expression is located on the left side of an assignment statement, whereas an r- value is located on the right side of an assignment statement.
- The address associated with a program variable in C is called its l-value; the contents of that location are its r- value, the quantity that is supposed to be the value of the variable.
- The r- value of a variable may change as program execution proceeds; but never its l- value.

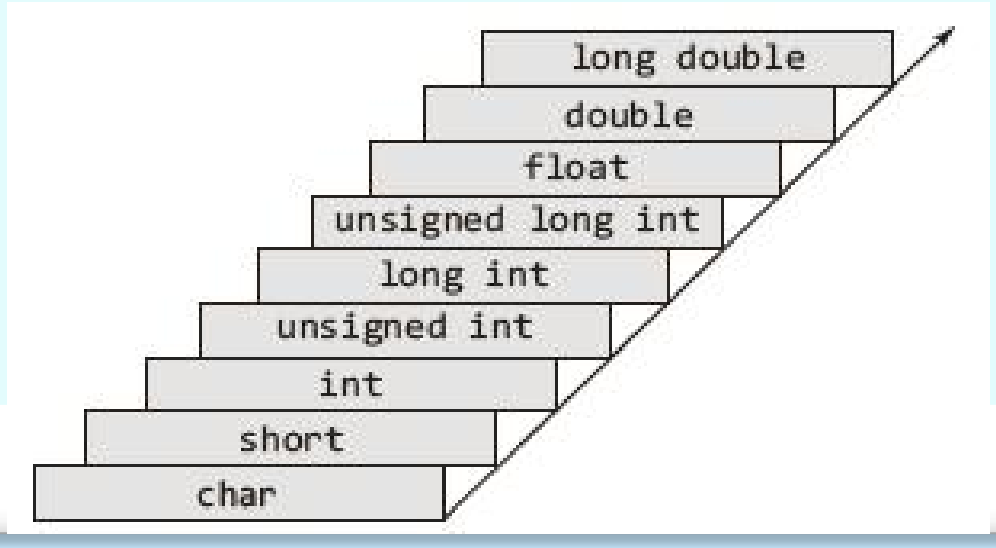
L-Values & R-Values

- For example :
  - ✓ a = b;
  - ✓ b, on the right-hand side of the assignment operator, is the quantity to be found at the address associated with b, i.e., an r- value. a is assigned the value stored in the address associated with b. a, on the left-hand side, is the address at which the contents are altered as a result of the assignment. a is an l- value. The assignment operation deposits b's r- value at a's l- value.

Lvalue	Rvalue
Consider the following assignment statement: a = b;	
Refers to the address that 'a' represents.	Means the content of the address that b represents.
is known at compile time.	is not known until runtime.
Says where to store the value.	Tells what is to be stored.
Cannot be an expression or a constant	Can be an expression or a constant

Type Conversion

- Though the C compiler performs *automatic type conversions, the programmer should be aware of what is* going on so as to understand how C evaluates expressions.
  - ✓ When a C expression is evaluated, the resulting value has a particular data type. If all the variables in the expression are of the same type, the resulting type is of the same type as well. For example, if x and y are both of int type , the expression x +y is of int type as well.
  - ✓ The smallest to the largest data types conversion with respect to size is along the arrow as shown below:



RULE : Type Conversion

- Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly using the **cast operator**

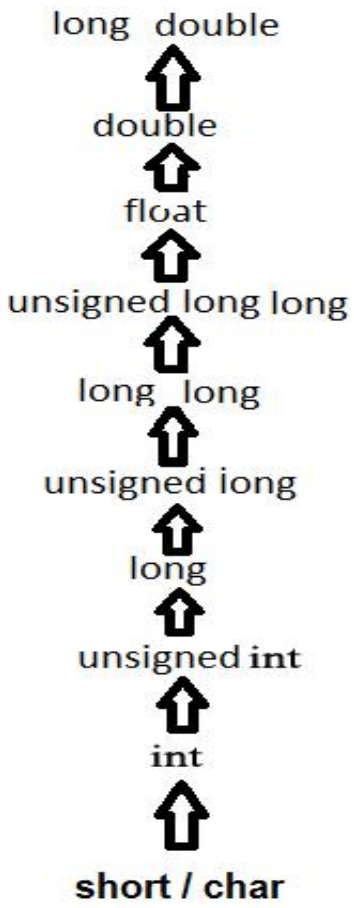
```
#include <stdio.h>
main()
{
 int sum = 17, count = 5;
double mean;
mean = (double) sum / count;
printf("Value of mean : %f\n", mean );
 }
```

Output

```
Value of mean : 3.400000
```

RULE : Type Conversion

- ✓ The **usual arithmetic conversions** are implicitly performed to cast their values to a common type. The compiler first performs *integer promotion (convert to int)*; if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy
- ✓ **char or short** (signed or unsigned) are converted to **int** (signed or unsigned).
- ✓ **float operands are converted to double.**
- ✓ **If any one operand is double, the other operand is also converted to double**, and that is the type of the result;
- ✓ **If any one operand is long, the other operand is treated as long**, and that is the type of the result;
- ✓ **If any one operand is of type unsigned, the other operand is converted to unsigned**, and that is also the type of the result.



RULE : Type Conversion

```
#include <stdio.h>
 main()
{
    int i = 17;
    char c = 'c'; /* ascii value is 99 */
    float sum;

    sum = i + c;
     printf("Value of sum : %f\n", sum );
}
```

When the above code is compiled and executed, it produces the following result −

```
Value of sum : 116.000000
```

- program : swap two numbers

- a=b
- b=a ??

SWAP two numbers

WAP to swap two integer numbers using third variable.

PROGRAM CODE

```
#include<stdio.h>
int main()
{
        int a,b,temp;
        printf("\nEnter two integers a and b : ");
        scanf("%d%d", &a,&b);
        temp=a;
        a=b;
        b=temp;
        printf("\nAfter swapping a=%d and b=%d",a,b);
        return 0;
}
```

RUN-1
Enter two integers a and b : 2 3
After swapping a=3 and b=2
RUN-2
Enter two integers a and b : 10 20
After swapping a=20 and b=10

- Program = add two times
- s, s1, s2
- m, m1, m2
- h, h1, h2
- day ?

```
#include<stdio.h>
main()
 {
 int h,m,s,h1,m1,s1,h2,m2,s2,day;
 printf("Enter first hours,minutes and
seconds\n");
 scanf("%d%d%d",&h1,&m1,&s1);
 printf("Enter second hours,minutes and
seconds\n");
 scanf("%d%d%d",&h2,&m2,&s2);
 s=h=m=day=0;
 s=s1+s2;
 if(s>60)
  {
  m=s/60;
  s=s%60;
  }
 m=m+m1+m2;
 if(m>60)
  {
  h=m/60;
  m=m%60;
  }
```

```
h=h+h1+h2;
 if(h>24)
  {
  day=1;
  h=h%24;
  }
 printf("First time =
%d:%d:%d",h1,m1,s1);
 printf("\nSecond time =
%d:%d:%d",h2,m2,s2);
 printf("\nAdded time =");
 if(day==0)
 printf("%d:%d:%d\n",h,m,s);
 else
  {
  printf("%d day",day);
  printf("%d:%d:%d\n",h,m,s);
  }
 }
```