PROGRAMMING IN C

Dr P.V. Praveen Sundar,
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

23-01-2021

- I CS

Introduction

- C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.
- In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.
- The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C.
- C has now become a widely used professional language for various reasons –
    - Easy to learn
    - Structured language
    - It produces efficient programs
    - It can handle low-level activities
    - It can be compiled on a variety of computer platforms.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and MySQL have been written in C.

Origin of C

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

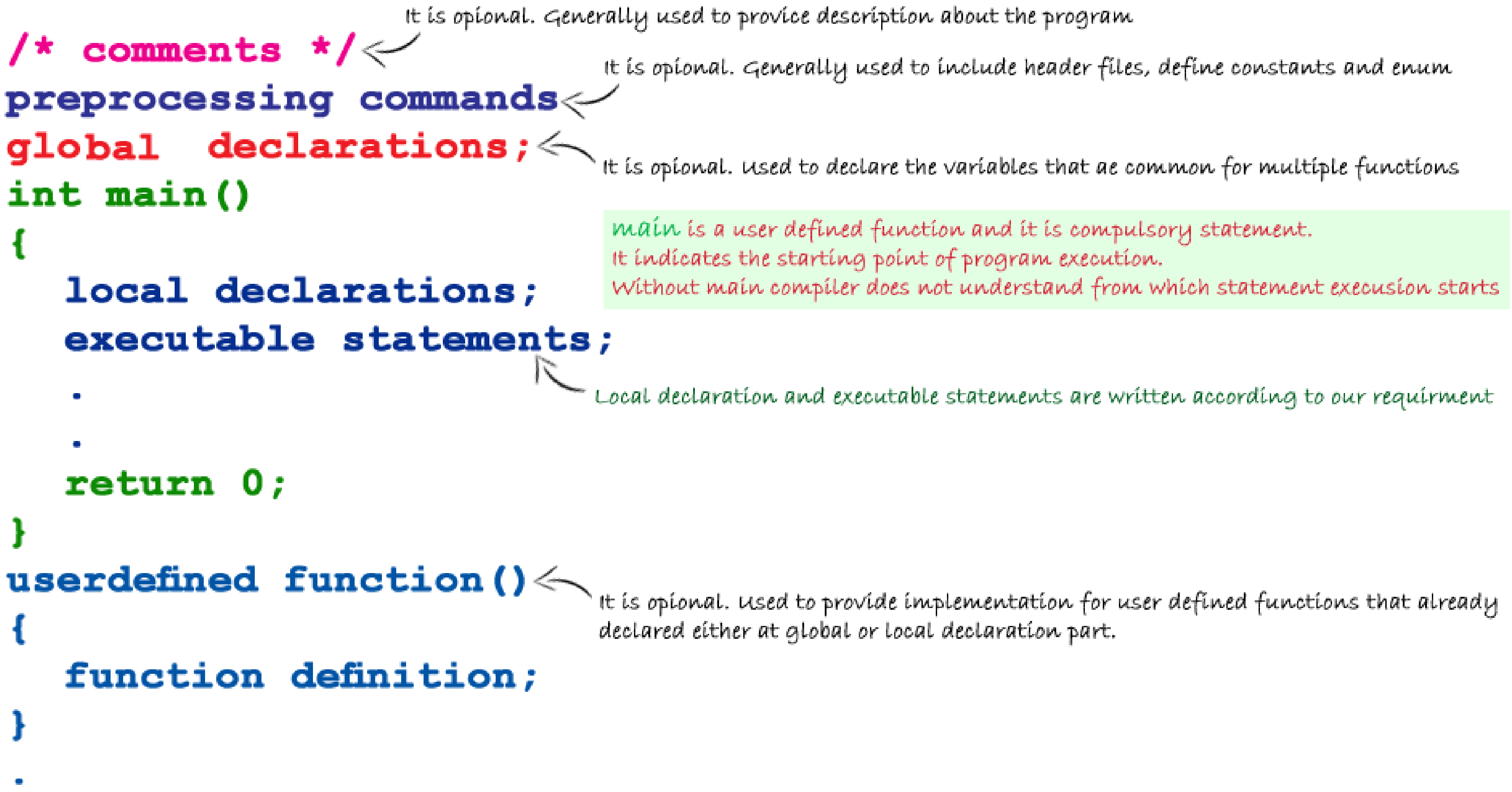
Features

C is the widely used language. It provides many features that are given below.

- Simple
- Machine Independent or Portable
- Mid-level programming language
- Structured programming language
- Rich Library
- Memory Management
- Fast Speed
- Pointers
- Recursion
- Extensible

C Program Basics

C is a structured programming language. Every c program and its statements must be in a particular structure. Every c program has the following general structure…



```
/* comments */
preprocessing commands
global  declarations;
int main()
{
   local declarations;
   executable statements;
   .
   .
   return 0;
}
userdefined function()
{
   function definition;
}
.
```

It is opional. Generally used to provice description about the program

It is opional. Generally used to include header files, define constants and enum

It is opional. Used to declare the variables that ae common for multiple functions

main is a user defined function and it is compulsory statement.
It indicates the starting point of program execution.
Without main compiler does not understand from which statement execusion starts

Local declaration and executable statements are written according to our requirment

It is opional. Used to provide implementation for user defined functions that already declared either at global or local declaration part.

- **Line 1: Comments -** They are ignored by the compiler
- This section is used to provide a small description of the program. The comment lines are simply ignored by the compiler, that means they are not executed. In C, there are two types of comments.
  - Single Line Comments: Single line comment begins with // symbol. We can write any number of single line comments.
  - Multiple Lines Comments: Multiple lines comment begins with /* symbol and ends with */. We can write any number of multiple lines comments in a program.

- In a C program, the comment lines are optional. Based on the requirement, we write comments. All the comment lines in a C program just provide the guidelines to understand the program and its code.
- **Line 2: Preprocessing Commands**
- Preprocessing commands are used to include header files and to define constants. We use the #include statement to include the header file into our program. We use a #define statement to define a constant. The preprocessing statements are used according to the requirements. If we don't need any header file, then no need to write #include statement. If we don't need any constant, then no need to write a #define statement.

Line 3: Global Declaration

- The global declaration is used to define the global variables, which are common for all the functions after its declaration. We also use the global declaration to declare functions. This global declaration is used based on the requirement.

Line 4: int main()

- Every C program must write this statement. This statement (main) specifies the starting point of the C program execution. Here, main is a user-defined method which tells the compiler that this is the starting point of the program execution. Here, int is a data type of a value that is going to return to the Operating System after completing the main method execution. If we don't want to return any value, we can use it as void.

Line 5: Open Brace ({)

- □ The open brace indicates the beginning of the block which belongs to the main method. In C program, every block begins with a '{' symbol.

Line 6: Local Declaration

- □ In this section, we declare the variables and functions that are local to the function or block in which they are declared. The variables which are declared in this section are valid only within the function or block in which they are declared.

Line 7: Executable statements

- □ In this section, we write the statements which perform tasks like reading data, displaying the result, calculations, etc., All the statements in this section are written according to the requirements.

Line 8: Return Statement

- □ Return Statement will returns the value to the operating system.

Line 9: Closing Brace (})

- The close brace indicates the end of the block which belongs to the main method. In C program every block ends with a '}' symbol.

Line 10, 11, 12, ...: User-defined function()

- This is the place where we implement the user-defined functions. The user-defined function implementation can also be performed before the main method. In this case, the user-defined function need not be declared. Directly it can be implemented, but it must be before the main method. In a program, we can define as many user-defined functions as we want. Every user-defined function needs a function call to execute its statements.

General rules for any C program

- Every executable statement must end with a semicolon symbol (;).
- Every C program must contain exactly one main method (Starting point of the program execution).
- All the system-defined words (keywords) must be used in lowercase letters.
- Keywords can not be used as user-defined names(identifiers).
- For every open brace ({), there must be respective closing brace (}).
- Every variable must be declared before it is used.

C Character Set

- As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters that include alphabets, digits, and special symbols. C language supports a total of 256 characters.
- Every C program contains statements. These statements are constructed using words and these words are constructed using characters from the C character set. C language character set contains the following set of characters.
  - Alphabets
  - Digits
  - Special Symbols

Alphabets

- C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.
  - lower case letters - a to z
  - UPPER CASE LETTERS - A to Z

Digits

- C language supports 10 digits which are used to construct numerical values in C language.
  - Digits - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Symbols

- C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.
  - Special Symbols - ~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ | tab newline space NULL bell backspace vertical tab etc.,

These are Control Characters

ASCII Value	Character	Meaning
0	NULL	null
1	SOH	Start of header
2	STX	start of text
3	ETX	end of text
4	EOT	end of transaction
5	ENQ	enquiry
6	ACK	acknowledgement
7	BEL	bell
8	BS	back Space
9	HT	Horizontal Tab
10	LF	Line Feed
11	VT	Vertical Tab
12	FF	Form Feed
13	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledgement
22	SYN	Synchronous Idle
23	ETB	End of Trans Block
24	CAN	Cancel
25	EM	End of Mediium
26	SUB	Sunstitute
27	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

These are Printable Characters

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
32	Space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39		71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

DEL is also Control Characters

Creating and Running C Program

- Generally, the programs created using programming languages like C, C++, Java, etc., are written using a high-level language like English. But, the computer cannot understand the high-level language. It can understand only low-level language. So, the program written in the high-level language needs to be converted into the low-level language to make it understandable for the computer. This conversion is performed using either Interpreter or Compiler.
- Popular programming languages like C, C++, Java, etc., use the compiler to convert high-level language instructions into low-level language instructions.
- A compiler is a program that converts high-level language instructions into low-level language instructions. Generally, the compiler performs two things, first it verifies the program errors, if errors are found, it returns a list of errors otherwise it converts the complete code into the low-level language.

Step 1 Create Source Code

Write program in the Editor & save it with .c extension

Step 2 Compile Source Code

Press Alt + F9 to compile

Step 3 Run Executable Code

Press Ctrl + F9 to run

Step 4 Check Result

Press
Alt + F5
to open UserScreen

Step 1: Creating a Source Code

- Source code is a file with C programming instructions in a high-level language. To create source code, we use any text editor to write the program instructions. The instructions written in the source code must follow the C programming language rules. The following steps are used to create a source code file in Windows OS…
  - Click on the Start button
  - Select Run
  - Type cmd and press Enter
  - Type cd c:\TC\bin in the command prompt and press Enter
  - Type TC press Enter
  - Click on File -> New in C Editor window
  - Type the program
  - Save it as FileName.c (Use shortcut key F2 to save)

Step 2: Compile Source Code (Alt + F9)

- □ The compilation is the process of converting high-level language instructions into low-level language instructions. We use the shortcut key **Alt + F9** to compile a C program in Turbo C.
- □ The compilation is the process of converting high-level language instructions into low-level language instructions.
- □ Whenever we press **Alt + F9**, the source file is going to be submitted to the Compiler. On receiving a source file, the compiler first checks for the Errors. If there are any Errors then compiler returns List of Errors, if there are no errors then the source code is converted into object code and stores it as a file with .obj extension. Then the object code is given to the Linker. The Linker combines both the object code and specified header file code and generates an Executable file with a .exe extension.

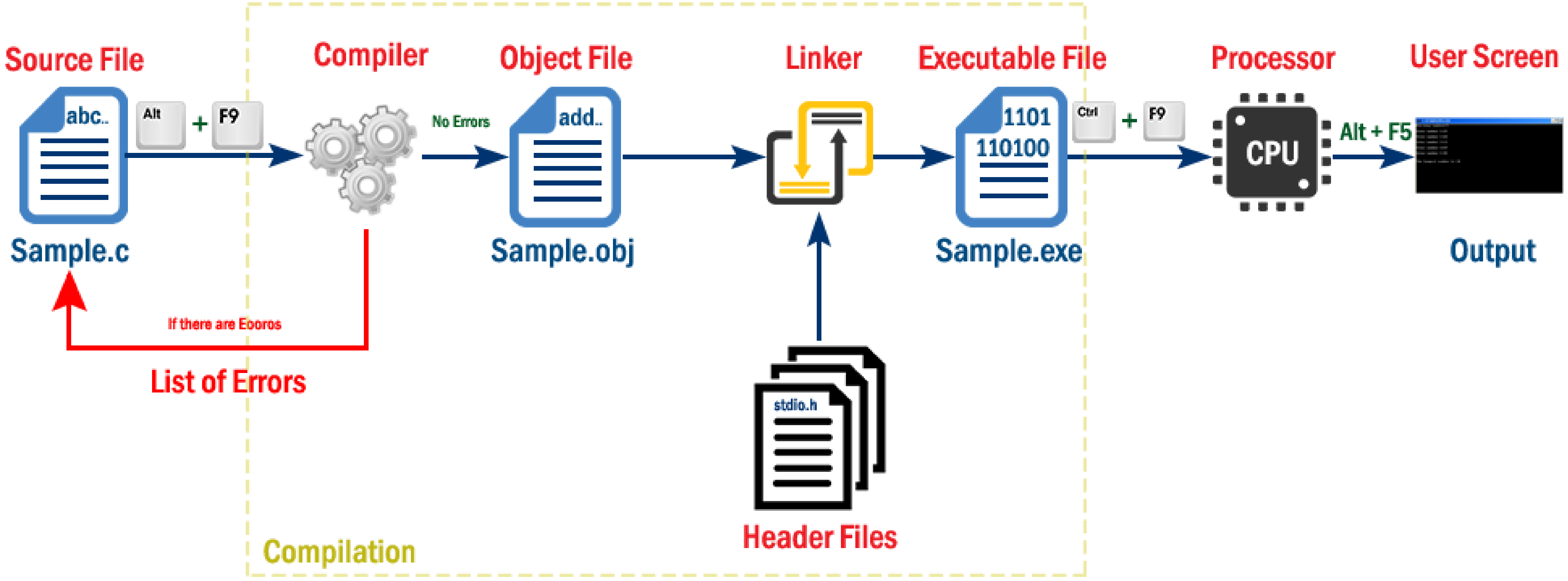
Step 3: Executing / Running Executable File (Ctrl + F9)

- After completing compilation successfully, an executable file is created with a **.exe** extension. The processor can understand this .exe file content so that it can perform the task specified in the source file.
- We use a shortcut key **Ctrl + F9** to run a C program. Whenever we press **Ctrl + F9**, the **.exe** file is submitted to the CPU. On receiving .exe file, CPU performs the task according to the instruction written in the file. The result generated from the execution is placed in a window called User Screen.

Step 4: Check Result (Alt + F5)

- After running the program, the result is placed into User Screen. Just we need to open the User Screen to check the result of the program execution. We use the shortcut key **Alt + F5** to open the User Screen and check the result.

Execution Process of a C Program



Overall Process

- Type the program in C editor and save with **.c extension** (Press **F2** to save).
- Press **Alt + F9** to compile the program.
- If there are errors, correct the errors and recompile the program.
- If there are no errors, then press **Ctrl + F9** to execute/run the program.
- Press **Alt + F5** to open User Screen and check the result.

C Tokens

- Every C program is a collection of instructions and every instruction is a collection of some individual units. Every smallest individual unit of a c program is called token. Every instruction in a c program is a collection of tokens. Tokens are used to construct c programs and they are said to the basic building blocks of a c program.
- In a c program tokens may contain the following...
  - Keywords
  - Identifiers
  - Operators
  - Special Symbols
  - Constants
  - Strings
  - Data values
- **In a C program, a collection of all the keywords, identifiers, operators, special symbols, constants, strings, and data values are called tokens.**

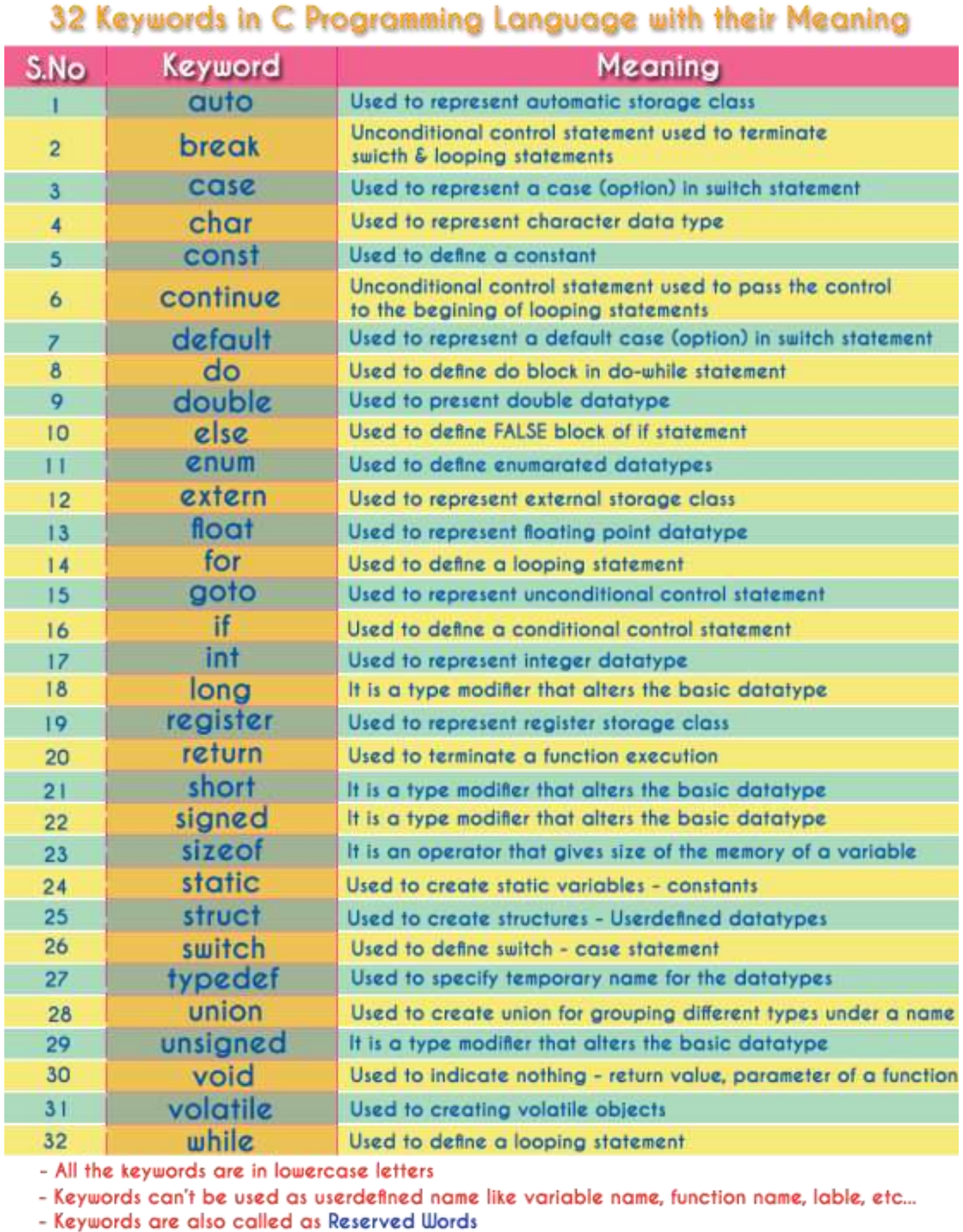
Consider the following C program…

```
#include<stdio.h>
#include<conio.h>
int main() {
        int i;
      clrscr();
        printf("ASCII  ==>  Character\n");
        for(i = -128; i <= 127; i++)
          printf("%d    ==>     %c\n", i, i);
      getch();
        return 0;
}
```

C Keywords

- As every language has words to construct statements, C programming also has words with a specific meaning which are used to construct c program instructions. In the C programming language, keywords are special words with predefined meaning. Keywords are also known as reserved words in C programming language.
- In the C programming language, there are **32 keywords**. All the 32 keywords have their meaning which is already known to the compiler.
- Keywords are the reserved words with predefined meaning which already known to the compiler
- Whenever C compiler come across a keyword, automatically it understands its meaning.

- Properties of Keywords
  - All the keywords in C programming language are defined as lowercase letters so they must be used only in lowercase letters
  - Every keyword has a specific meaning, users can not change that meaning.
  - Keywords can not be used as user-defined names like variable, functions, arrays, pointers, etc...
  - Every keyword in C programming language represents something or specifies some kind of action to be performed by the compiler.
  - The following table specifies all the 32 keywords with their meaning



32 Keywords in C Programming Language with their Meaning

S.No	Keyword	Meaning
1	auto	Used to represent automatic storage class
2	break	Unconditional control statement used to terminate swicth & looping statements
3	case	Used to represent a case (option) in switch statement
4	char	Used to represent character data type
5	const	Used to define a constant
6	continue	Unconditional control statement used to pass the control to the begining of looping statements
7	default	Used to represent a default case (option) in switch statement
8	do	Used to define do block in do-while statement
9	double	Used to present double datatype
10	else	Used to define FALSE block of if statement
11	enum	Used to define enumarated datatypes
12	extern	Used to represent external storage class
13	float	Used to represent floating point datatype
14	for	Used to define a looping statement
15	goto	Used to represent unconditional control statement
16	if	Used to define a conditional control statement
17	int	Used to represent integer datatype
18	long	It is a type modifier that alters the basic datatype
19	register	Used to represent register storage class
20	return	Used to terminate a function execution
21	short	It is a type modifier that alters the basic datatype
22	signed	It is a type modifier that alters the basic datatype
23	sizeof	It is an operator that gives size of the memory of a variable
24	static	Used to create static variables - constants
25	struct	Used to create structures - Userdefined datatypes
26	switch	Used to define switch - case statement
27	typedef	Used to specify temporary name for the datatypes
28	union	Used to create union for grouping different types under a name
29	unsigned	It is a type modifier that alters the basic datatype
30	void	Used to indicate nothing - return value, parameter of a function
31	volatile	Used to creating volatile objects
32	while	Used to define a looping statement

- All the keywords are in lowercase letters
- Keywords can't be used as userdefined name like variable name, function name, lable, etc...
- Keywords are also called as Reserved Words

C Identifiers

- In C programming language, programmers can specify their name to a variable, array, pointer, function, etc... An identifier is a collection of characters which acts as the name of variable, function, array, pointer, structure, etc... In other words, an identifier can be defined as the user-defined name to identify an entity uniquely in the c programming language that name may be of the variable name, function name, array name, pointer name, structure name or a label.
- The identifier is a user-defined name of an entity to identify it uniquely during the program execution.
- Example

```
int marks;
char studentName[30];
```

  - Here, marks and studentName are identifiers.

Rules for Creating Identifiers

- An identifier can contain letters (UPPERCASE and lowercase), numerics & underscore symbol only.
- An identifier should not start with a numerical value. It can start with a letter or an underscore.
- We should not use any special symbols in between the identifier even whitespace. However, the only underscore symbol is allowed.
- Keywords should not be used as identifiers.
- There is no limit for the length of an identifier. However, the compiler considers the first 31 characters only.
- An identifier must be unique in its scope.

Rules for Creating Identifiers for better programming

- The following are the commonly used rules for creating identifiers for better programming...
  - The identifier must be meaningful to describe the entity.
  - Since starting with an underscore may create conflict with system names, so we avoid starting an identifier with an underscore.
  - We start every identifier with a lowercase letter. If an identifier contains more than one word then the first word starts with a lowercase letter and second word onwards first letter is used as an UPPERCASE letter. We can also use an underscore to separate multiple words in an identifier.

Valid Identifiers

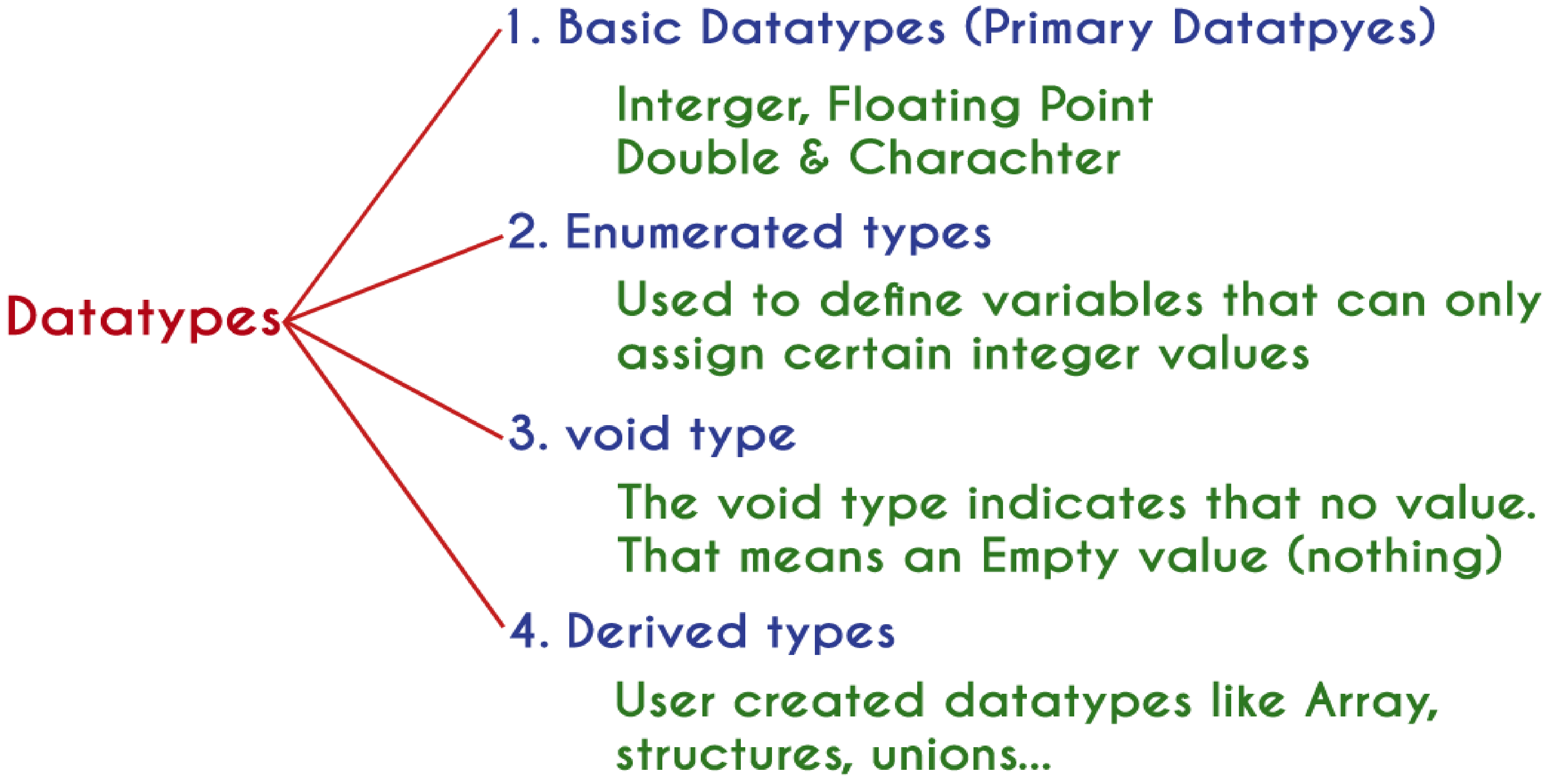
- int a,b;
- float _a;
- char _123;
- double pi;
- int value,Value,vAlue;
- int Auto;

Invalid Identifiers

- int a b;
- float 123a;
- char str-;
- double pi, a;
- int break;

Datatypes

- Data used in c program is classified into different types based on its properties. In the C programming language, a data type can be defined as a set of values with similar characteristics. All the values in a data type have the same properties.
- Data types in the c programming language are used to specify what kind of value can be stored in a variable. The memory size and type of the value of a variable are determined by the variable data type. In a c program, each variable or constant or array must have a data type and this data type specifies how much memory is to be allocated and what type of values are to be stored in that variable or constant or array. The formal definition of a data type is as follows...
- **The Data type is a set of value with predefined characteristics. data types are used to declare variable, constants, arrays, pointers, and functions.**



Datatypes

1. Basic Datatypes (Primary Datatpyes)

   Interger, Floating Point
   Double & Charachter

2. Enumerated types

   Used to define variables that can only
   assign certain integer values

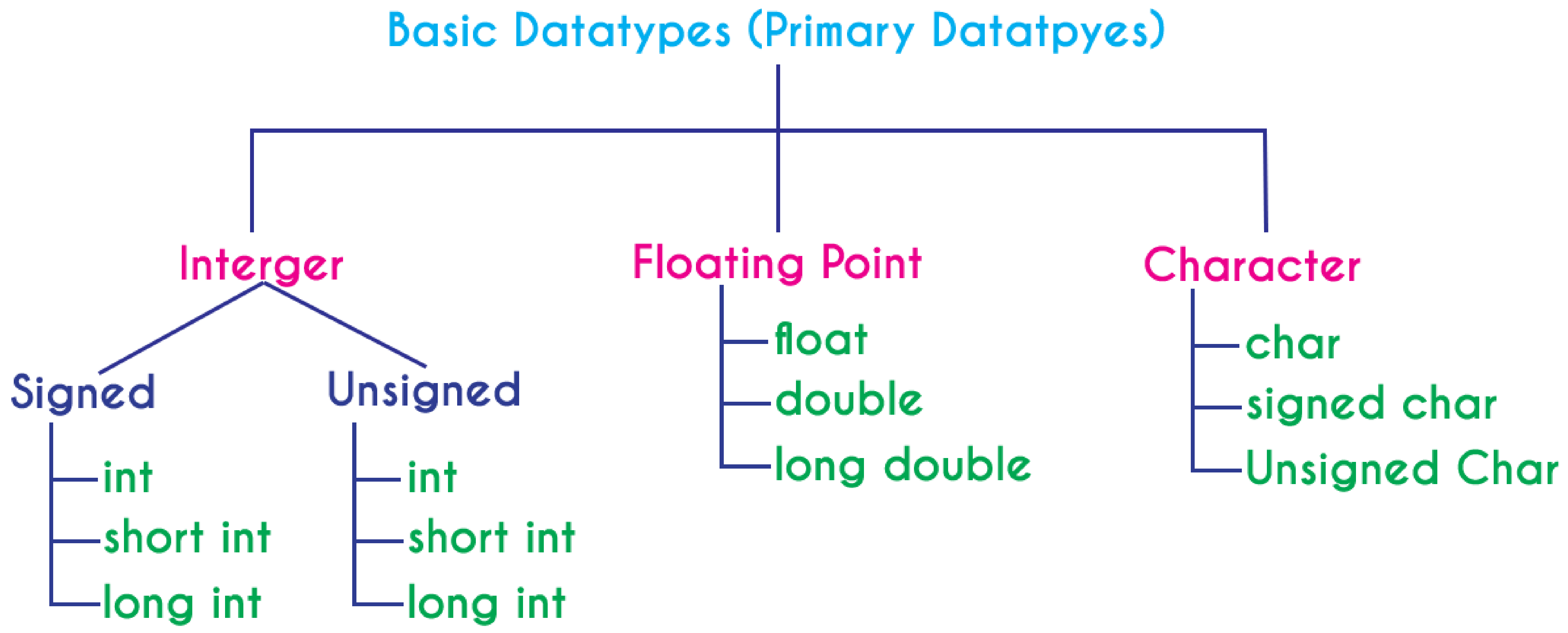
3. void type

   The void type indicates that no value.
   That means an Empty value (nothing)

4. Derived types

   User created datatypes like Array,
   structures, unions...

- In the c programming language, data types are classified as follows...
  - Primary data types (Basic data types or Predefined data types)
  - Derived data types (Secondary data types OR User-defined data types)
  - Enumeration data types
  - Void data type
- **Primary data types**
- The primary data types in the C programming language are the basic data types. All the primary data types are already defined in the system. Primary data types are also called as Built-In data types. The following are the primary data types in c programming language...
  - Integer data type
  - Floating Point data type
  - Double data type
  - Character data type

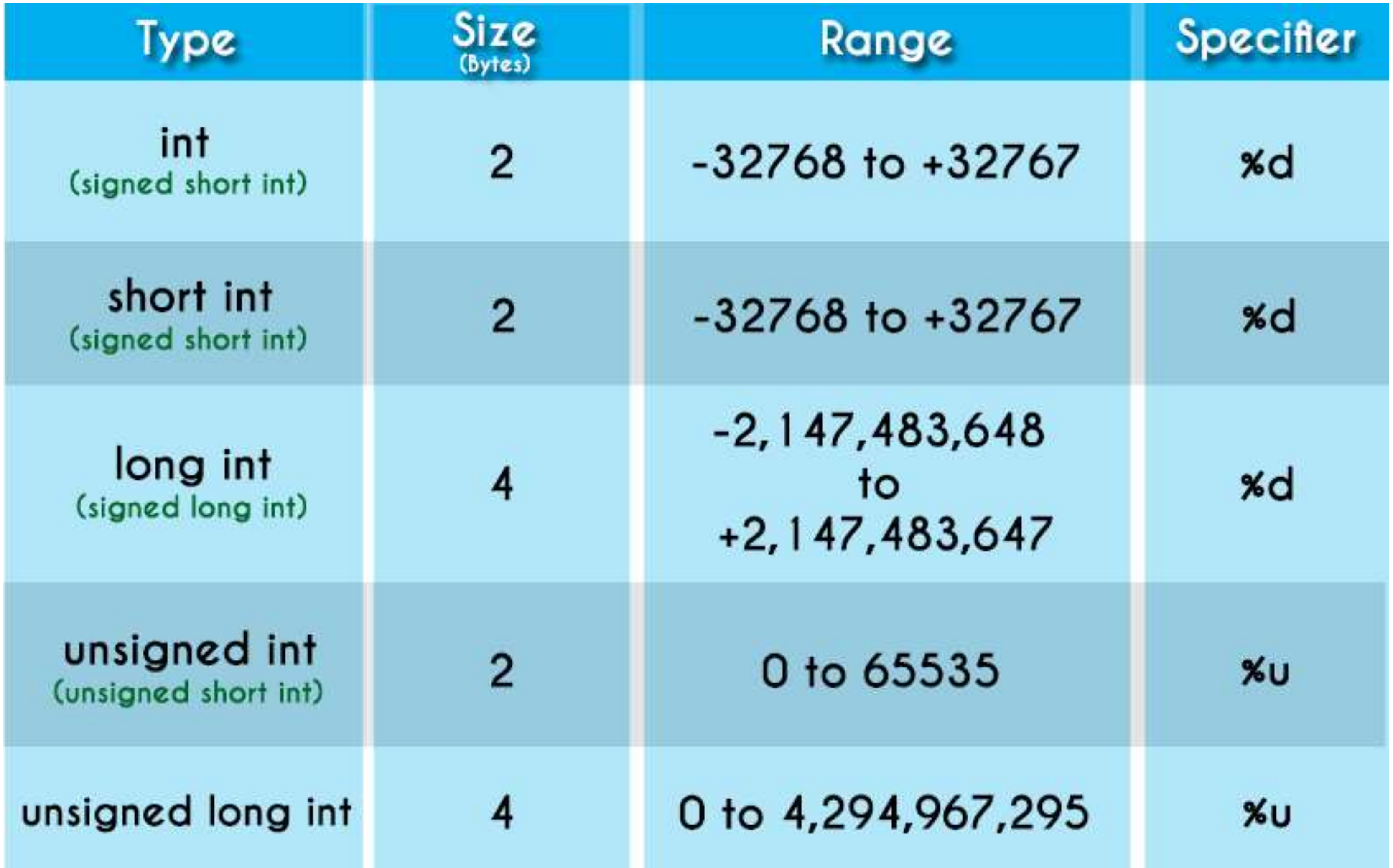


Basic Datatypes (Primary Datatpyes)

- Interger
  - Signed
    - int
    - short int
    - long int
  - Unsigned
    - int
    - short int
    - long int
- Floating Point
  - float
  - double
  - long double
- Character
  - char
  - signed char
  - Unsigned Char

Integer Data type

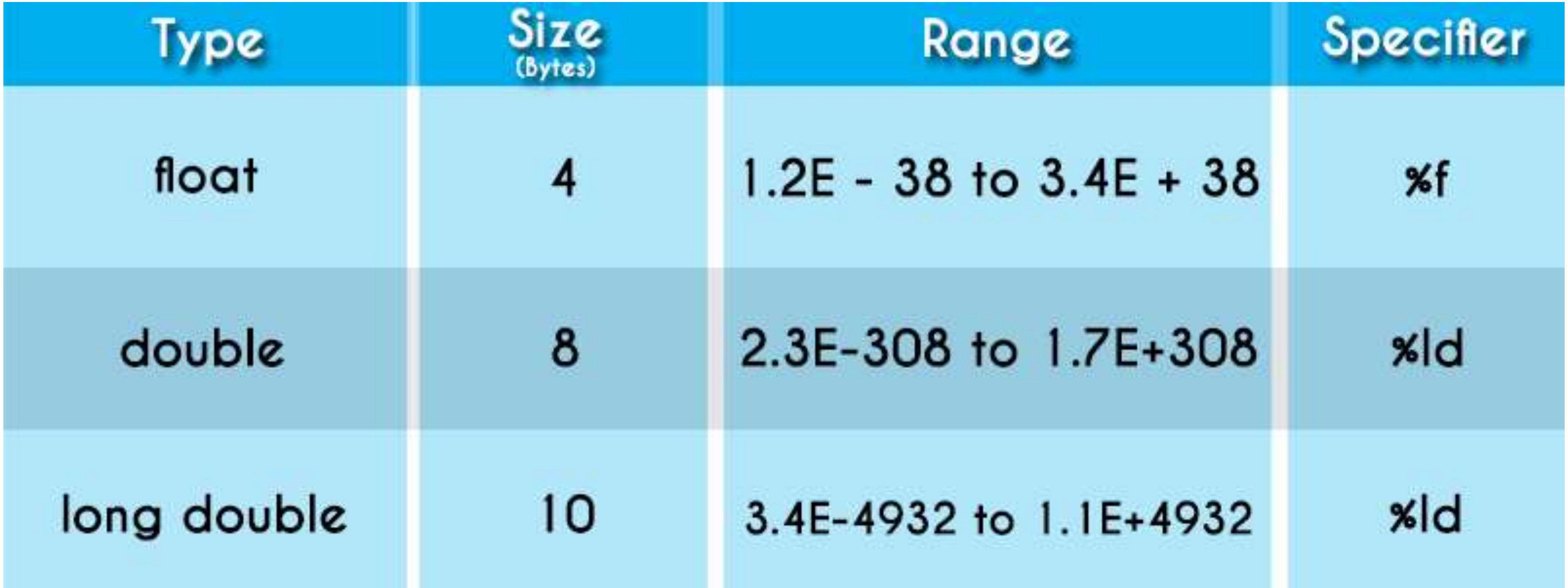
- The integer data type is a set of whole numbers. Every integer value does not have the decimal value. We use the keyword "int" to represent integer data type in c. We use the keyword int to declare the variables and to specify the return type of a function. The integer data type is used with different type modifiers like short, long, signed and unsigned. The following table provides complete details about the integer data type.



Type	Size (Bytes)	Range	Specifier
int (signed short int)	2	-32768 to +32767	%d
short int (signed short int)	2	-32768 to +32767	%d
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647	%d
unsigned int (unsigned short int)	2	0 to 65535	%u
unsigned long int	4	0 to 4,294,967,295	%u

Floating Point Data Types

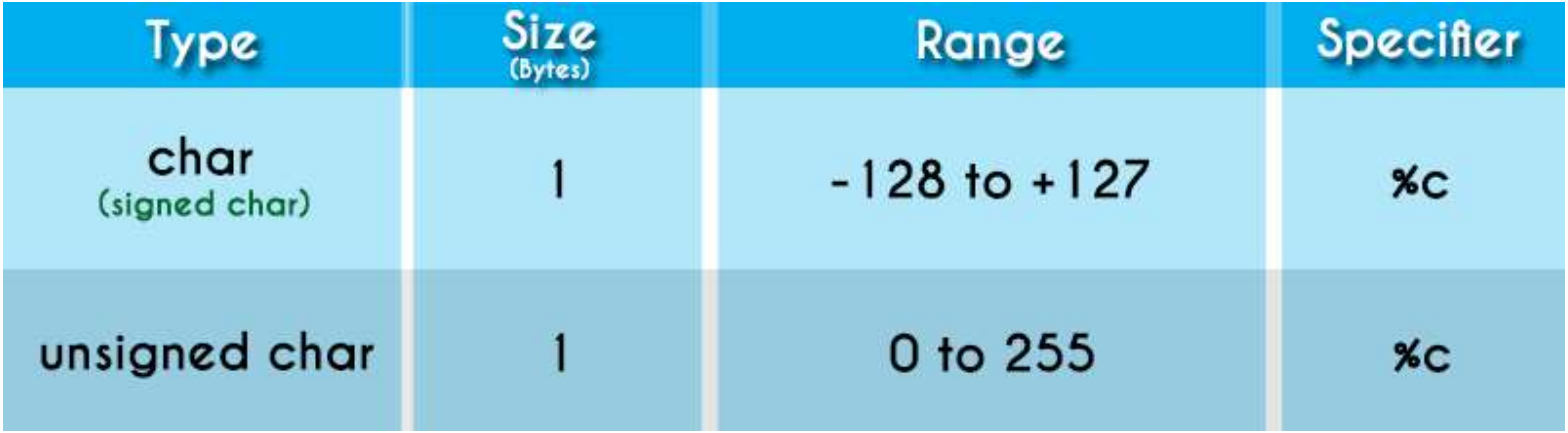
- Floating-point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating-point data type has two variants...
  - float
  - double
- We use the keyword "float" to represent floating-point data type and "double" to represent double data type in c. Both float and double are similar but they differ in the number of decimal places. The float value contains 6 decimal places whereas double value contains 15 or 19 decimal places. The following table provides complete details about floating-point data types.



Type	Size (Bytes)	Range	Specifier
float	4	1.2E - 38 to 3.4E + 38	%f
double	8	2.3E-308 to 1.7E+308	%ld
long double	10	3.4E-4932 to 1.1E+4932	%ld

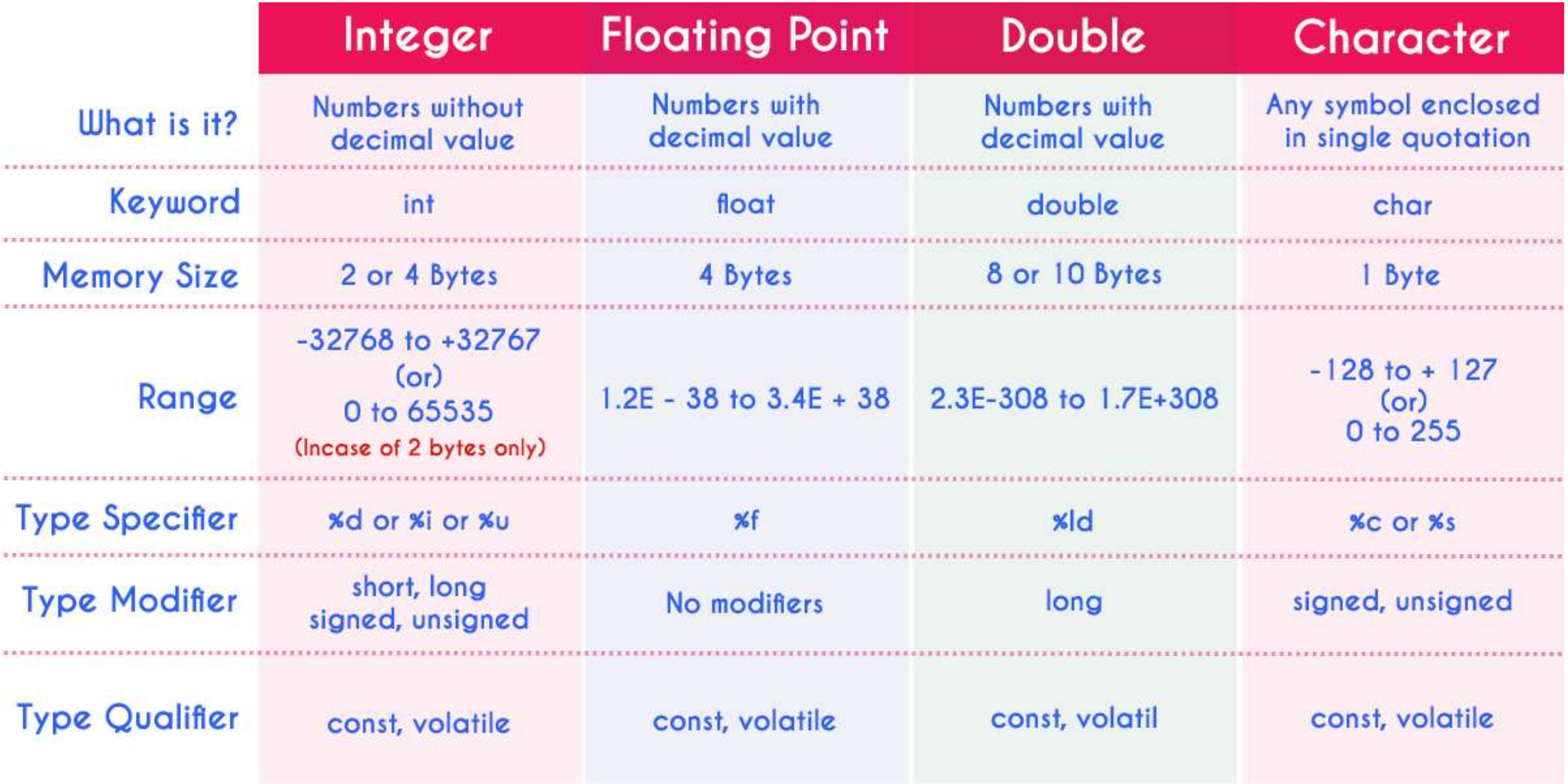
Character Data Type

- The character data type is a set of characters enclosed in single quotations. The following table provides complete details about the character data type.



Type	Size (Bytes)	Range	Specifier
char (signed char)	1	-128 to +127	%c
unsigned char	1	0 to 255	%c

The following table provides complete information about all the data types in c programming language..



	Integer	Floating Point	Double	Character
What is it?	Numbers without decimal value	Numbers with decimal value	Numbers with decimal value	Any symbol enclosed in single quotation
Keyword	int	float	double	char
Memory Size	2 or 4 Bytes	4 Bytes	8 or 10 Bytes	1 Byte
Range	-32768 to +32767 (or) 0 to 65535 (Incase of 2 bytes only)	1.2E - 38 to 3.4E + 38	2.3E-308 to 1.7E+308	-128 to + 127 (or) 0 to 255
Type Specifier	%d or %i or %u	%f	%ld	%c or %s
Type Modifier	short, long signed, unsigned	No modifiers	long	signed, unsigned
Type Qualifier	const, volatile	const, volatile	const, volatil	const, volatile

void data type

- The void data type means nothing or no value. Generally, the void is used to specify a function which does not return any value. We also use the void data type to specify empty parameters of a function.

Enumerated data type

- An enumerated data type is a user-defined data type that consists of integer constants and each integer constant is given a name. The keyword "enum" is used to define the enumerated data type.

Derived data types

- Derived data types are user-defined data types. The derived data types are also called as user-defined data types or secondary data types. In the c programming language, the derived data types are created using the following concepts...
  - Arrays
  - Structures
  - Unions
  - Enumeration

Variables

- Variables in a c programming language are the named memory locations where the user can store different values of the same datatype during the program execution. In other words, a variable can be defined as a storage container to hold values of the same datatype during the program execution.
- The formal definition of a variable is as follows...
- **Variable is a name given to a memory location where we can store different values of the same datatype during the program execution.**
- Every variable in c programming language must be declared in the declaration section before it is used. Every variable must have a datatype that determines the range and type of values be stored and the size of the memory to be allocated.
- A variable name may contain letters, digits and underscore symbol. The following are the rules to specify a variable name...
  - Variable name should not start with a digit.
  - Keywords should not be used as variable names.
  - A variable name should not contain any special symbols except underscore(_).
  - A variable name can be of any length but compiler considers only the first 31 characters of the variable name.

Declaration of Variable

- Declaration of a variable tells the compiler to allocate the required amount of memory with the specified variable name and allows only specified datatype values into that memory location. In C programming language, the declaration can be performed either before the function as global variables or inside any block or function. But it must be at the beginning of block or function.

Declaration Syntax:

```
datatype variableName;
```

Example

```
int number;
```

- The above declaration tells to the compiler that allocates **2 bytes** of memory with the name **number** and allows only integer values into that memory location.

Constants

- □ In C programming language, a constant is similar to the variable but the constant hold only one value during the program execution. That means, once a value is assigned to the constant, that value can't be changed during the program execution. Once the value is assigned to the constant, it is fixed throughout the program. A constant can be defined as follows...
- □ A constant is a named memory location which holds only one value throughout the program execution.
- □ In C programming language, a constant can be of any data type like integer, floating-point, character, string and double, etc.,

Integer constants

- An integer constant can be a decimal integer or octal integer or hexadecimal integer. A decimal integer value is specified as direct integer value whereas octal integer value is prefixed with 'o' and hexadecimal value is prefixed with 'OX'.
- An integer constant can also be unsigned type of integer constant or long type of integer constant. Unsigned integer constant value is suffixed with 'u' and long integer constant value is suffixed with 'l' whereas unsigned long integer constant value is suffixed with 'ul'.
- Example
  - 125 → Decimal Integer Constant
  - O76 → Octal Integer Constant
  - OX3A → Hexa Decimal Integer Constant
  - 50u → Unsigned Integer Constant
  - 30l → Long Integer Constant
  - 100ul → Unsigned Long Integer Constant

Floating Point constants

- □ A floating-point constant must contain both integer and decimal parts. Some times it may also contain the exponent part. When a floating-point constant is represented in exponent form, the value must be suffixed with 'e' or 'E'.

Example

The floating-point value 3.14 is represented as 3E-14 in exponent form.

Character Constants

- □ A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.

Example

'A'

'2'

'+'

String Constants

A string constant is a collection of characters, digits, special symbols and escape sequences that are enclosed in double quotations.

We define string constant in a single line as follows...

```
"This is C Programming class"
```

We can define string constant using multiple lines as follows...

```
" This\
is\
C Programming class "
```

We can also define string constant by separating it with white space as follows...

```
"This" "is" " C Programming "
```

All the above three defines the same string constant.

Creating constants in C

- In a c programming language, constants can be created using two concepts...
  - Using the 'const' keyword
  - Using '#define' preprocessor

Using the 'const' keyword

- We create a constant of any datatype using 'const' keyword. To create a constant, we prefix the variable declaration with 'const' keyword.
- The general syntax for creating constant using 'const' keyword is as follows...

  const datatype constantName ;

  OR

  const datatype constantName = value ;
- **Example**

  const int x = 10 ;

  Here, 'x' is a integer constant with fixed value 10.

Example Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i = 9 ;
   const int x = 10 ;

   i = 15 ;
   x = 100 ; // creates an error
   printf("i = %d\n x = %d", i, x ) ;
}
```

The above program gives an error because we are trying to change the constant variable value (x = 100).

Using '#define' preprocessor

We can also create constants using '#define' preprocessor directive. When we create constant using this preprocessor directive it must be defined at the beginning of the program (because all the preprocessor directives must be written before the global declaration).

We use the following syntax to create constant using '#define' preprocessor directive...

#define CONSTANTNAME value

Example

#define PI 3.14

Here, PI is a constant with value 3.14

Example Program

```
#define  PI  3.14
void main(){
   int r, area ;
   printf("Please enter the radius of circle : ") ;
   scanf("%d", &r) ;
   area = PI * (r * r) ;
   printf("Area of the circle = %d", area) ;
}
```

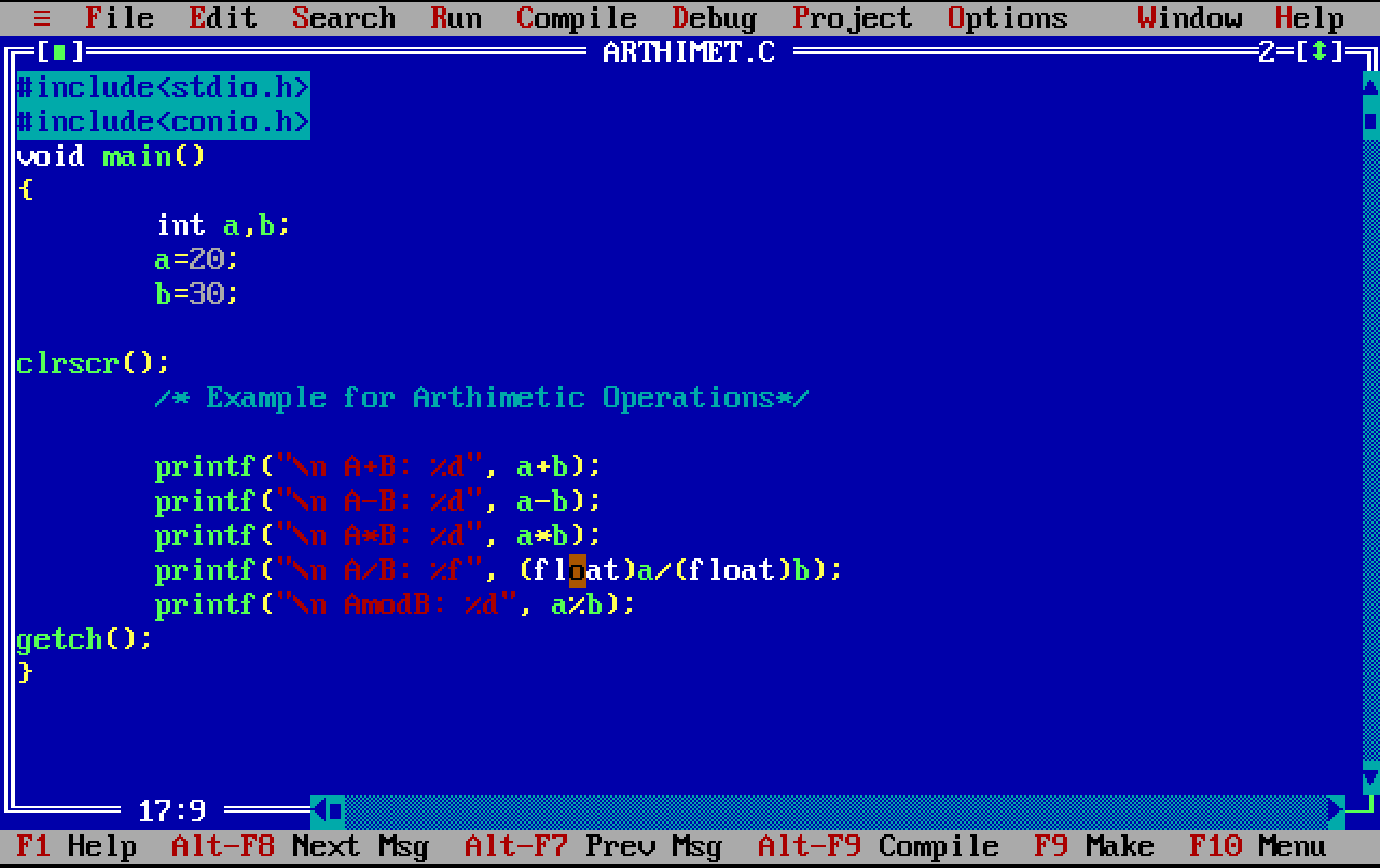
Operators

- An operator is a symbol used to perform arithmetic and logical operations in a program. That means an operator is a special symbol that tells the compiler to perform mathematical or logical operations. C programming language supports a rich set of operators that are classified as follows.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Increment & Decrement Operators
  - Assignment Operators
  - Bitwise Operators
  - Conditional Operator
  - Special Operators

Arithmetic Operators (+, -, *, /, %)

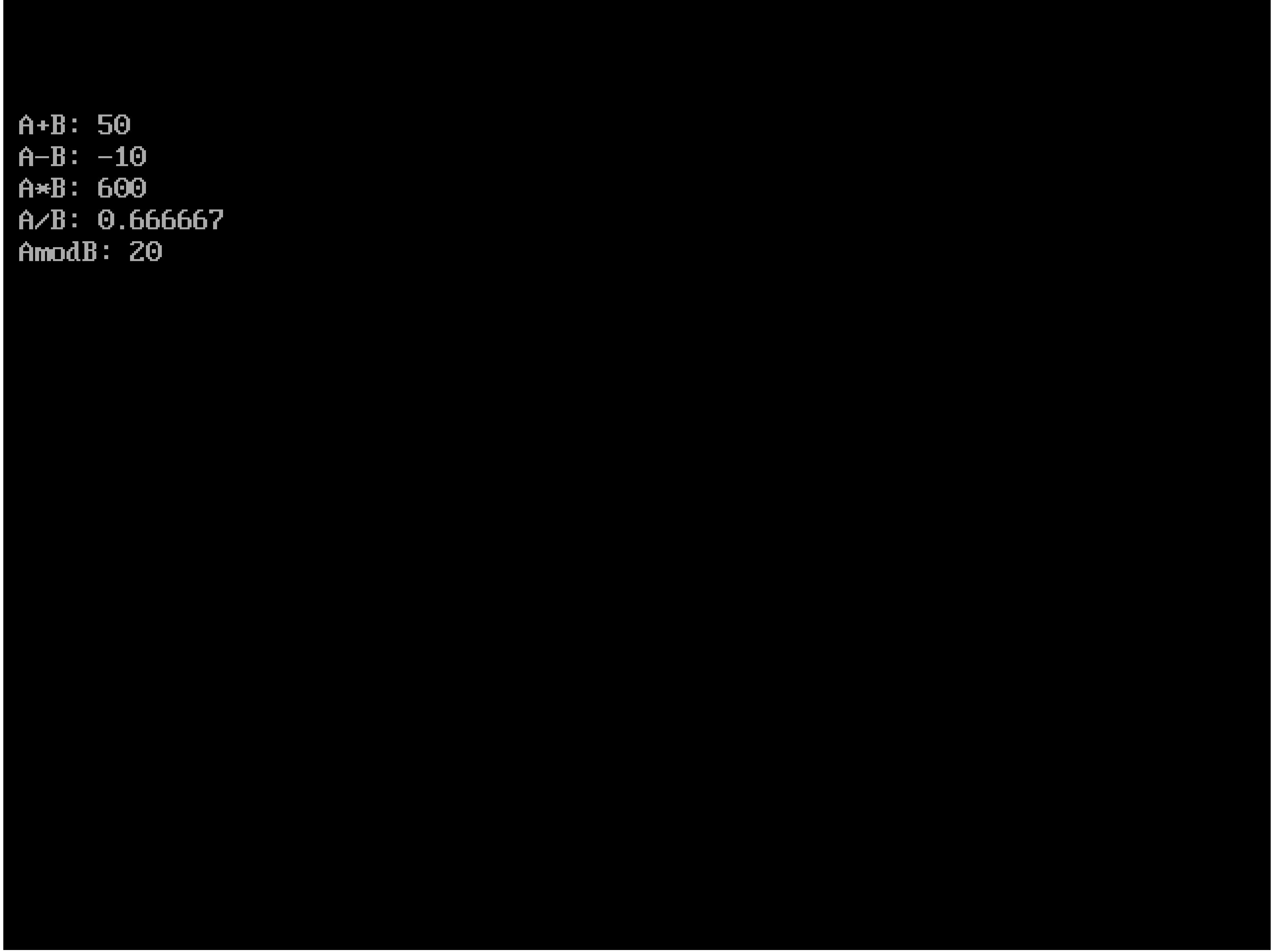
- The arithmetic operators are the symbols that are used to perform basic mathematical operations like addition, subtraction, multiplication, division and percentage modulo. The following table provides information about arithmetic operators.
- The addition operator can be used with numerical data types and character data type. When it is used with numerical values, it performs mathematical addition and when it is used with character data type values, it performs concatenation (appending).
- The remainder of the division operator is used with integer data type only.

Operator	Meaning	Example
+	Addition	10 + 5 = 15
-	Subtraction	10 - 5 = 5
*	Multiplication	10 * 5 = 50
/	Division	10 / 5 = 2
%	Remainder of the Division	5 % 2 = 1



≡ File Edit Search Run Compile Debug Project Options Window Help
[■] ARTHIMET.C 2=[↕]
17:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        a=20;
        b=30;

clrscr();
        /* Example for Arthimetic Operations*/

        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: %f", (float)a/(float)b);
        printf("\n AmodB: %d", a%b);
getch();
}
```

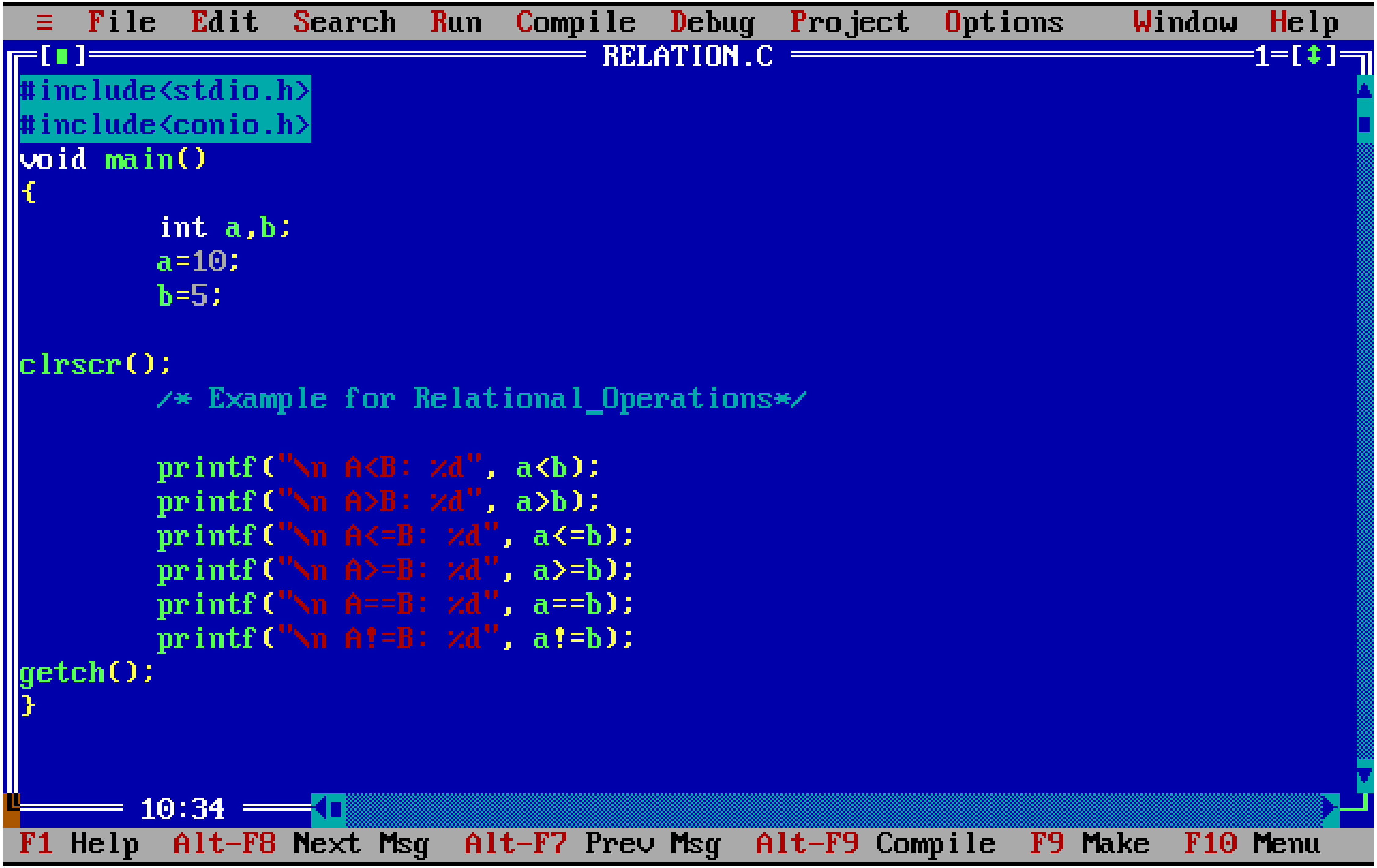


```
A+B: 50
A-B: -10
A*B: 600
A/B: 0.666667
AmodB: 20
```

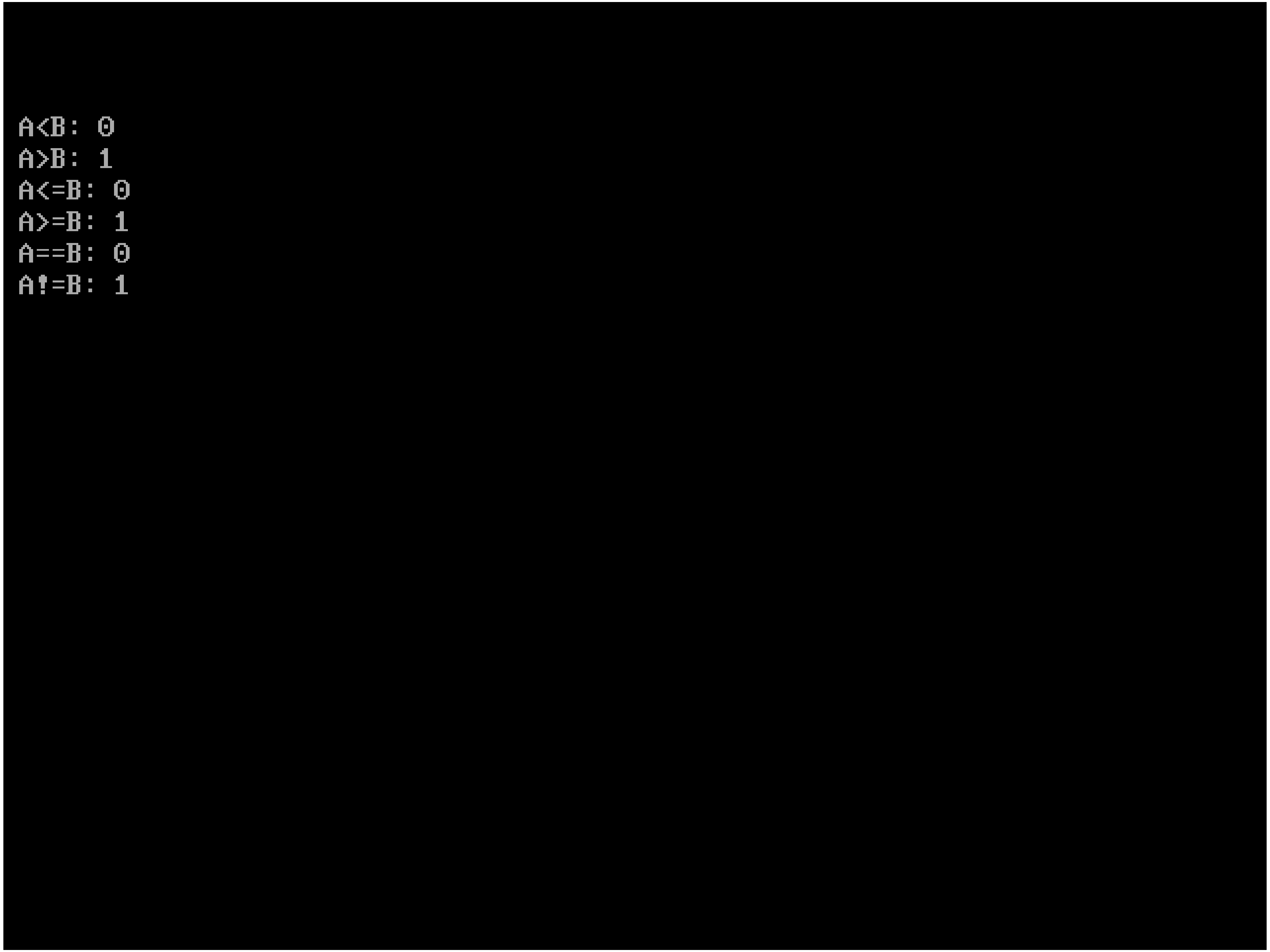
Relational Operators (<, >, <=, >=, ==, !=)

- The relational operators are the symbols that are used to compare two values. That means the relational operators are used to check the relationship between two values. Every relational operator has two results TRUE or FALSE. In simple words, the relational operators are used to define conditions in a program. The following table provides information about relational operators.

Operator	Meaning	Example
<	Returns TRUE if the first value is smaller than second value otherwise returns FALSE	10 < 5 is FALSE
>	Returns TRUE if the first value is larger than second value otherwise returns FALSE	10 > 5 is TRUE
<=	Returns TRUE if the first value is smaller than or equal to second value otherwise returns FALSE	10 <= 5 is FALSE
>=	Returns TRUE if the first value is larger than or equal to second value otherwise returns FALSE	10 >= 5 is TRUE
==	Returns TRUE if both values are equal otherwise returns FALSE	10 == 5 is FALSE
!=	Returns TRUE if both values are not equal otherwise returns FALSE	10 != 5 is TRUE


≡ File Edit Search Run Compile Debug Project Options Window Help
RELATION.C
1
10:34
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        a=10;
        b=5;

clrscr();
        /* Example for Relational_Operations*/

        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);
getch();
}
```

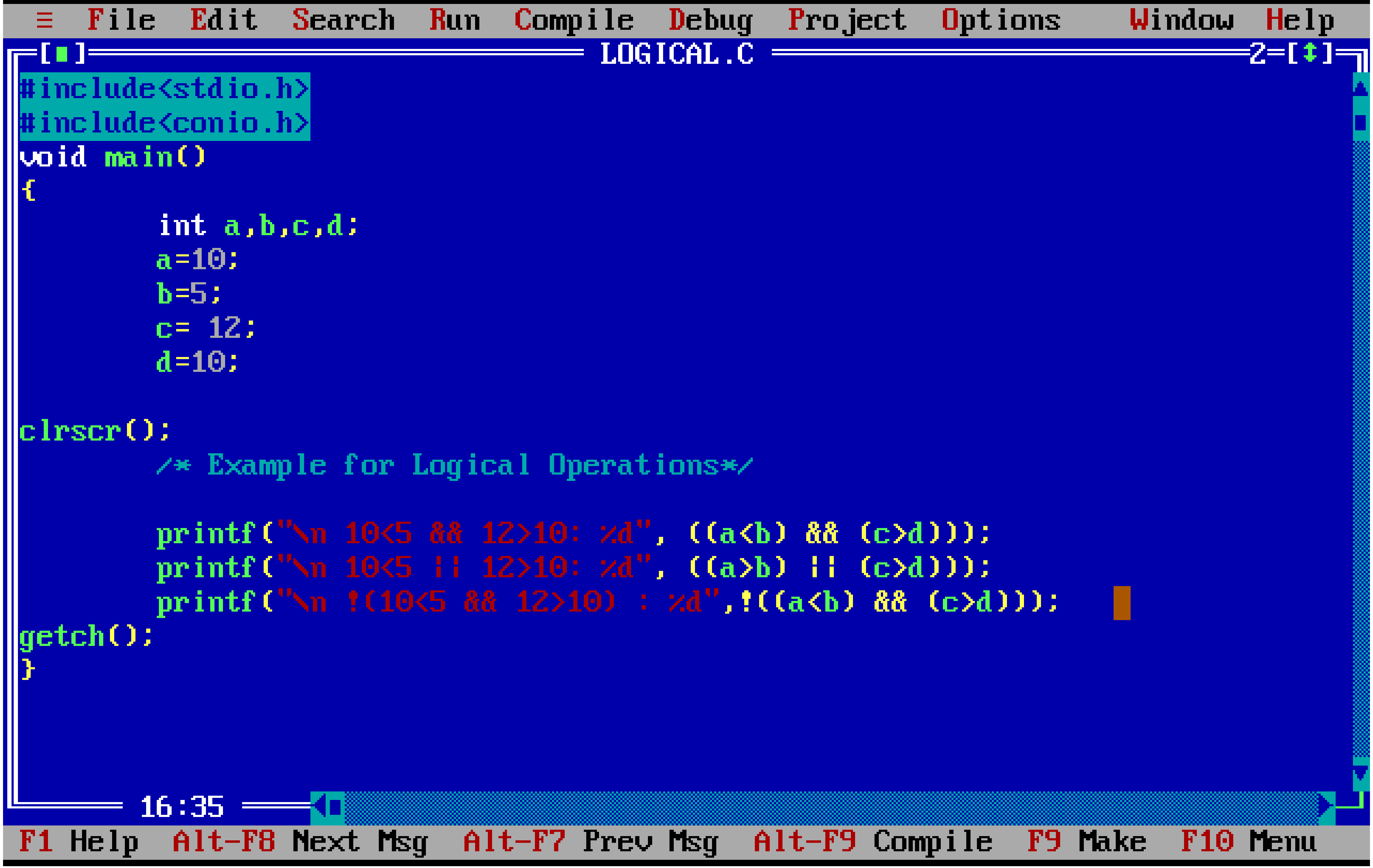


```
A<B: 0
A>B: 1
A<=B: 0
A>=B: 1
A==B: 0
A!=B: 1
```

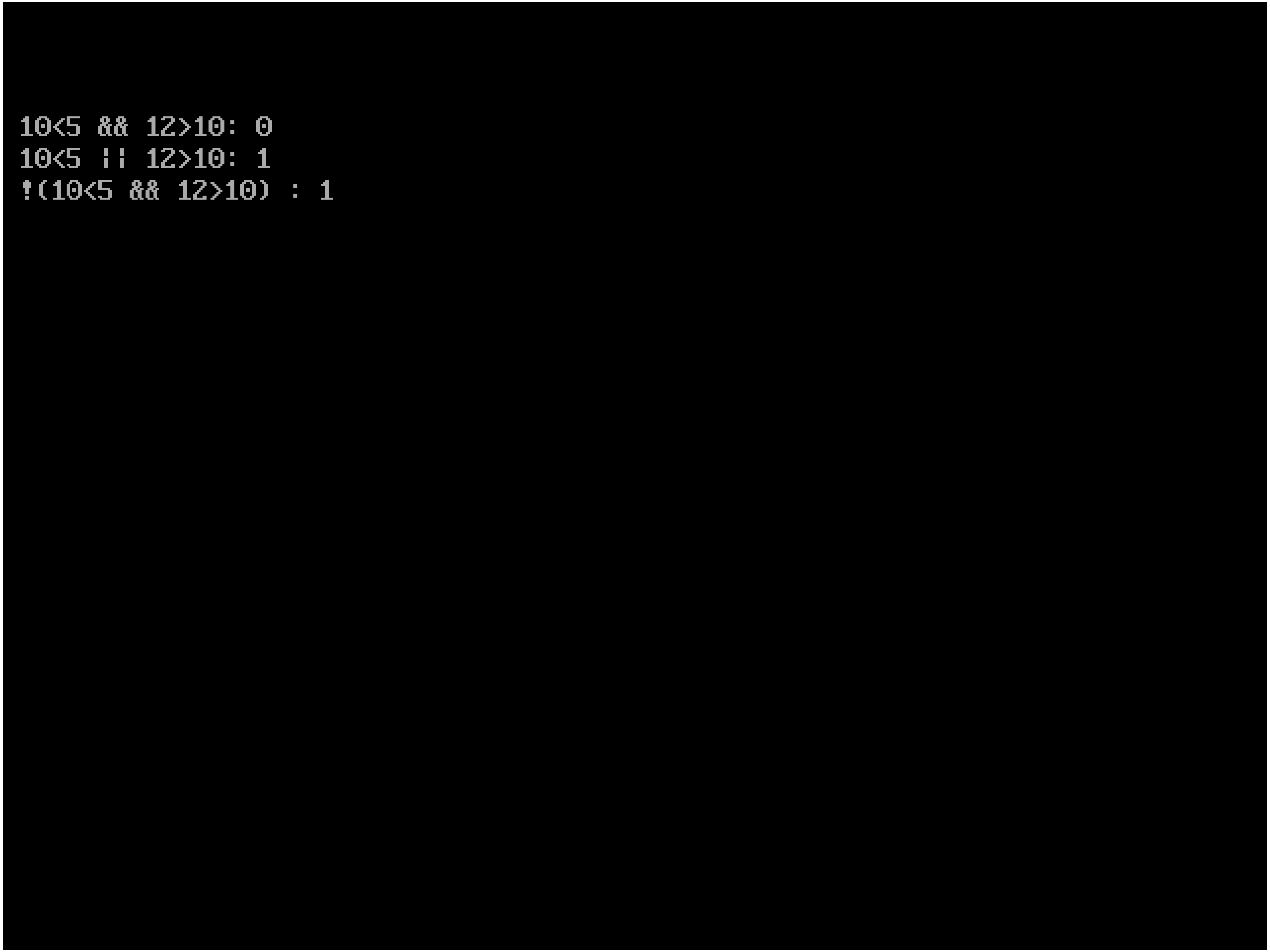
Logical Operators (&&, ||, !)

- The logical operators are the symbols that are used to combine multiple conditions into one condition. The following table provides information about logical operators.
- Logical AND - Returns TRUE only if all conditions are TRUE, if any of the conditions is FALSE then complete condition becomes FALSE.
- Logical OR - Returns FALSE only if all conditions are FALSE, if any of the conditions is TRUE then complete condition becomes TRUE.

Operator	Meaning	Example
&&	**Logical AND** - Returns TRUE if all conditions are TRUE otherwise returns FALSE	10 < 5 && 12 > 10 is FALSE
||	**Logical OR** - Returns FALSE if all conditions are FALSE otherwise returns TRUE	10 < 5 || 12 > 10 is TRUE
!	**Logical NOT** - Returns TRUE if condition is FALSE and returns FALSE if it is TRUE	!(10 < 5 && 12 > 10) is TRUE


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
LOGICAL.C
2
16:35
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b,c,d;
        a=10;
        b=5;
        c= 12;
        d=10;

clrscr();
        /* Example for Logical Operations*/

        printf("\n 10<5 && 12>10: %d", ((a<b) && (c>d)));
        printf("\n 10<5 || 12>10: %d", ((a>b) || (c>d)));
        printf("\n !(10<5 && 12>10) : %d",!((a<b) && (c>d)));
getch();
}
```



```
10<5 && 12>10: 0
10<5 || 12>10: 1
!(10<5 && 12>10) : 1
```

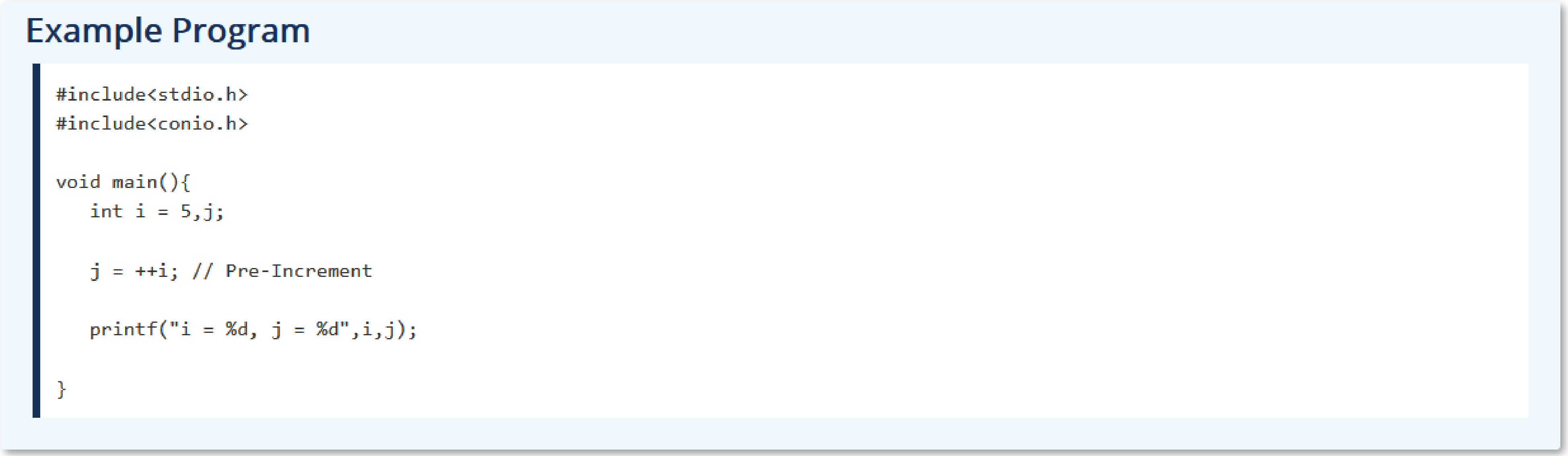
Increment & Decrement Operators (++ & --)

- The increment and decrement operators are called unary operators because both need only one operand. The increment operators adds one to the existing value of the operand and the decrement operator subtracts one from the existing value of the operand. The following table provides information about increment and decrement operators.
- The increment and decrement operators are used Infront of the operand (++a) or after the operand (a++). If it is used in front of the operand, we call it as pre-increment or pre-decrement and if it is used after the operand, we call it as post-increment or post-decrement.

Operator	Meaning	Example
++	**Increment** - Adds one to existing value	int a = 5; a++; ⇒ a = 6
--	**Decrement** - Subtracts one from existing value	int a = 5; a--; ⇒ a = 4

Pre-Increment or Pre-Decrement

- In the case of pre-increment, the value of the variable is increased by one before the expression evaluation. In the case of pre-decrement, the value of the variable is decreased by one before the expression evaluation. That means, when we use pre-increment or pre-decrement, first the value of the variable is incremented or decremented by one, then the modified value is used in the expression evaluation.

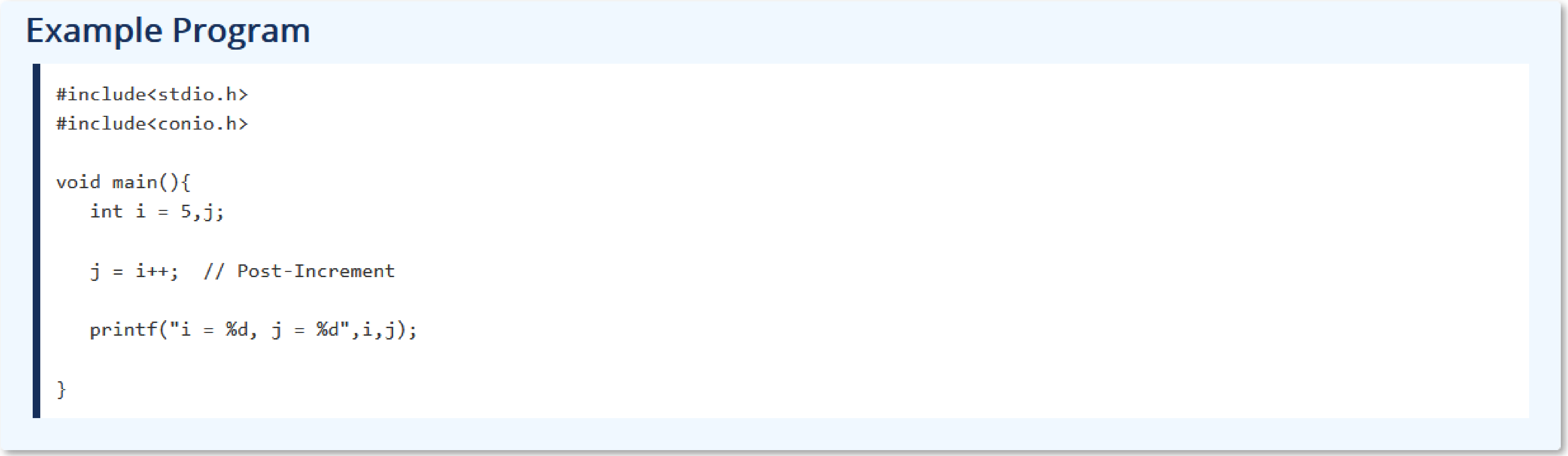


Example Program

```
#include<stdio.h>
#include<conio.h>

void main(){
   int i = 5,j;

   j = ++i; // Pre-Increment

   printf("i = %d, j = %d",i,j);

}
```

Post-Increment or Post-Decrement

□ In the case of post-increment, the value of the variable is increased by one after the expression evaluation. In the case of post-decrement, the value of the variable is decreased by one after the expression evaluation. That means, when we use post-increment or post-decrement, first the expression is evaluated with existing value, then the value of the variable is incremented or decremented by one.



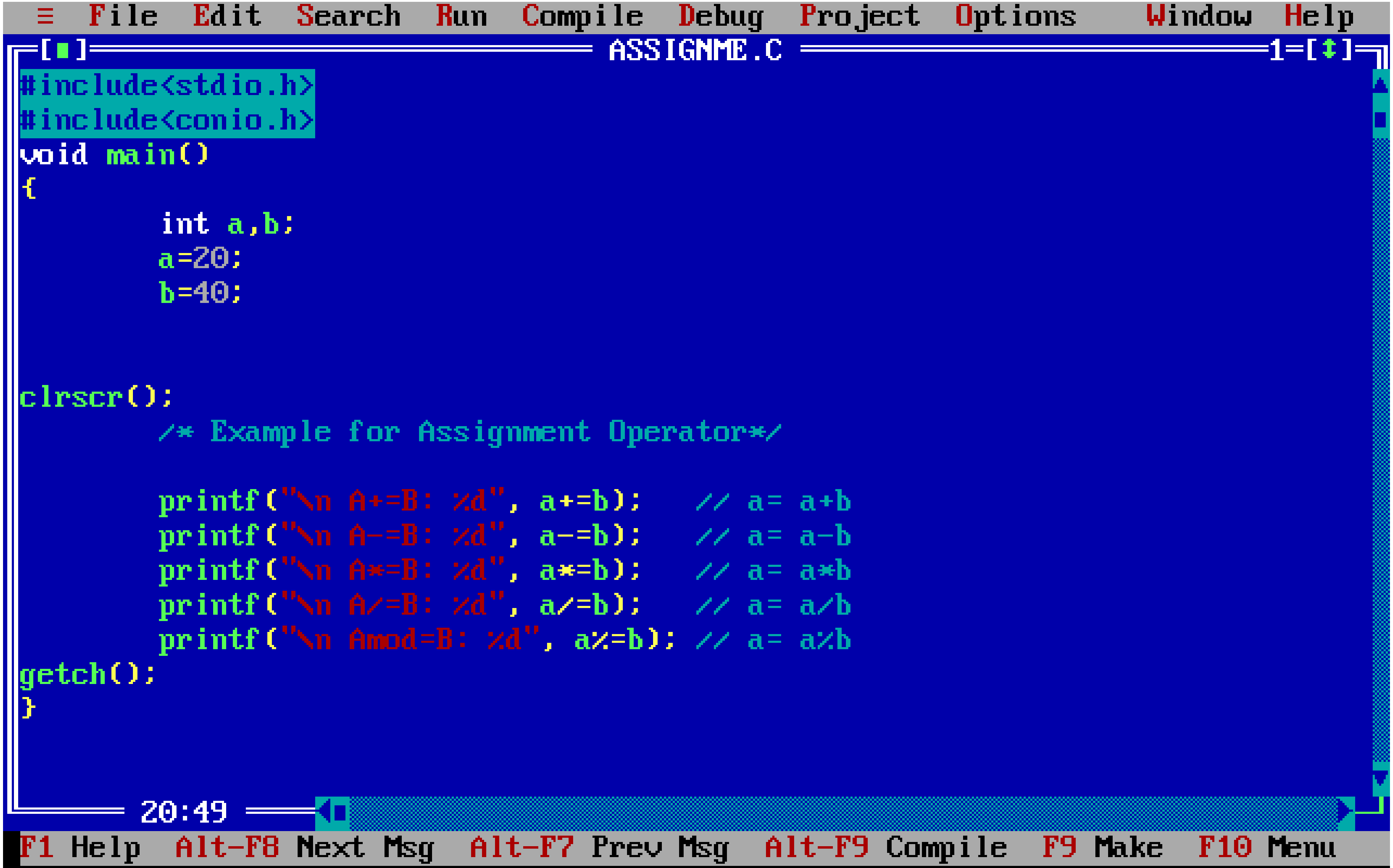
Example Program

```
#include<stdio.h>
#include<conio.h>

void main(){
   int i = 5,j;

   j = i++;  // Post-Increment

   printf("i = %d, j = %d",i,j);

}
```

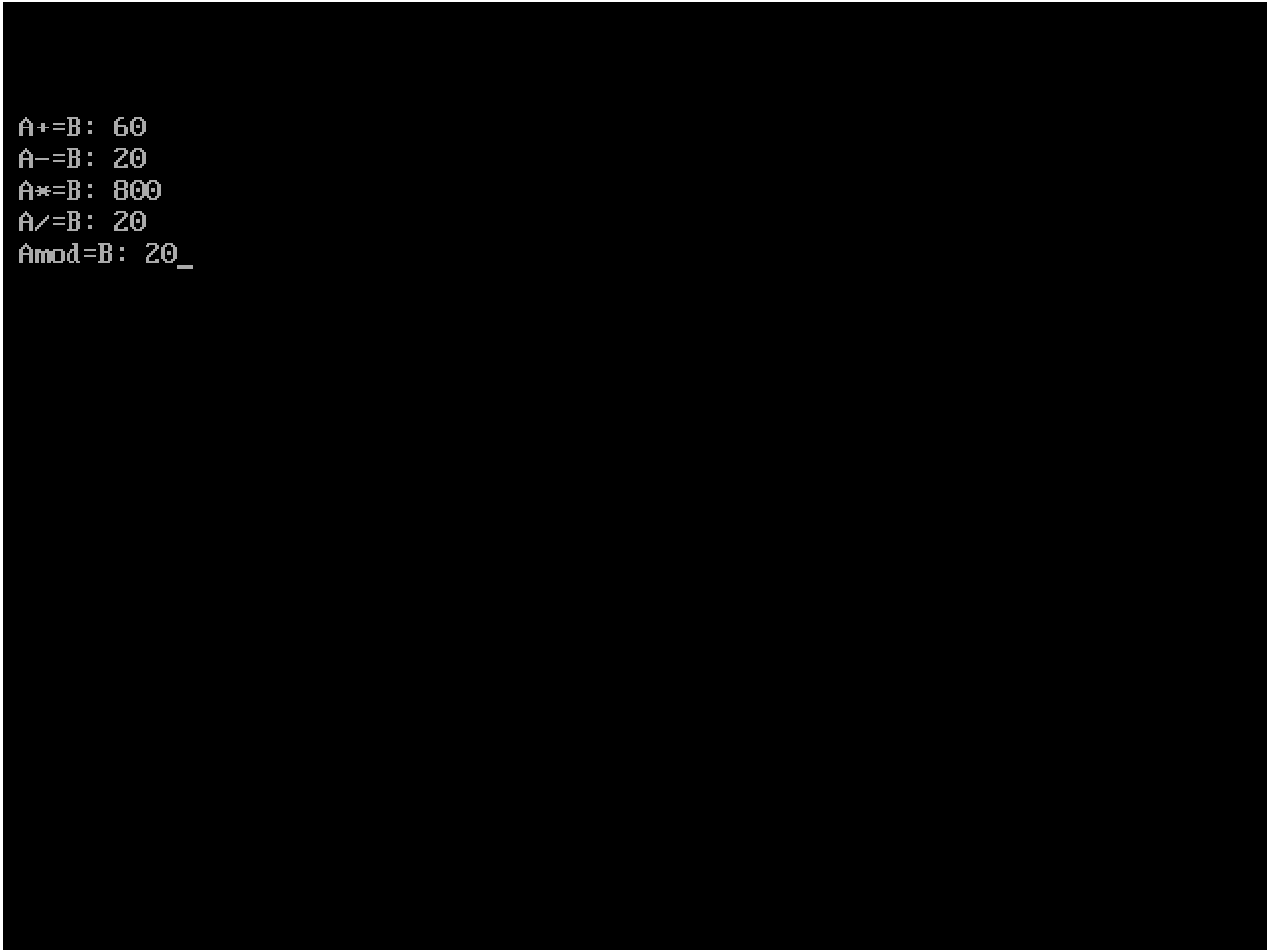
Assignment Operators (=, +=, -=, *=, /=, %=)

- The assignment operators are used to assign right-hand side value (Rvalue) to the left-hand side variable (Lvalue). The assignment operator is used in different variants along with arithmetic operators. The following table describes all the assignment operators in the C programming language.

Operator	Meaning	Example
=	Assign the right-hand side value to left-hand side variable	A = 15
+=	Add both left and right-hand side values and store the result into left-hand side variable	A += 10 ⇒ A = A+10
-=	Subtract right-hand side value from left-hand side variable value and store the result into left-hand side variable	A -= B ⇒ A = A-B
*=	Multiply right-hand side value with left-hand side variable value and store the result into left-hand side variable	A *= B ⇒ A = A*B
/=	Divide left-hand side variable value with right-hand side variable value and store the result into the left-hand side variable	A /= B ⇒ A = A/B
%=	Divide left-hand side variable value with right-hand side variable value and store the remainder into the left-hand side variable	A %= B ⇒ A = A%B


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
ASSIGNME.C
1=[↕]
20:49
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        a=20;
        b=40;


clrscr();
        /* Example for Assignment Operator*/

        printf("\n A+=B: %d", a+=b);   // a= a+b
        printf("\n A-=B: %d", a-=b);   // a= a-b
        printf("\n A*=B: %d", a*=b);   // a= a*b
        printf("\n A/=B: %d", a/=b);   // a= a/b
        printf("\n Amod=B: %d", a%=b); // a= a%b
getch();
}
```

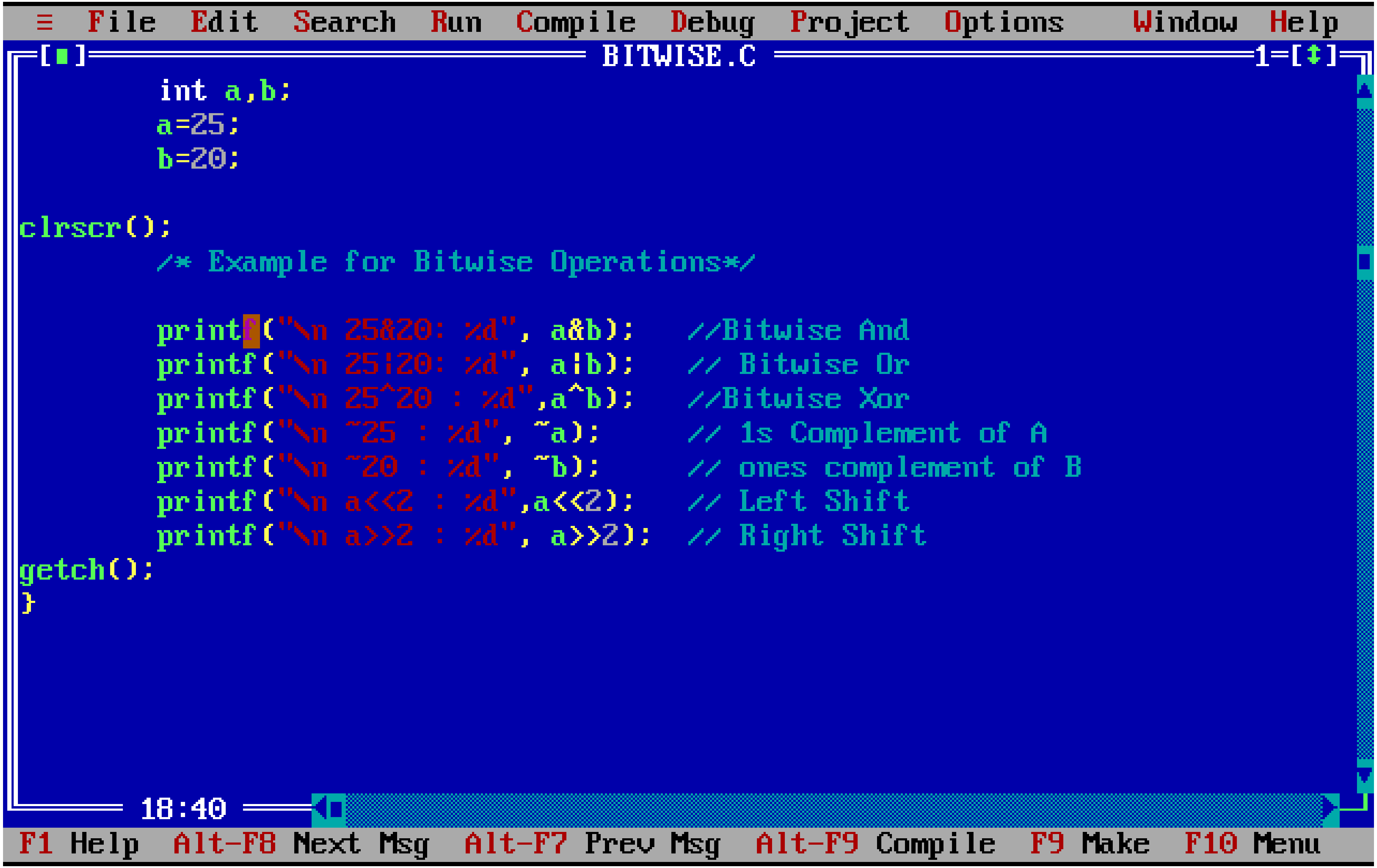


```
A+=B: 60
A-=B: 20
A*=B: 800
A/=B: 20
Amod=B: 20_
```

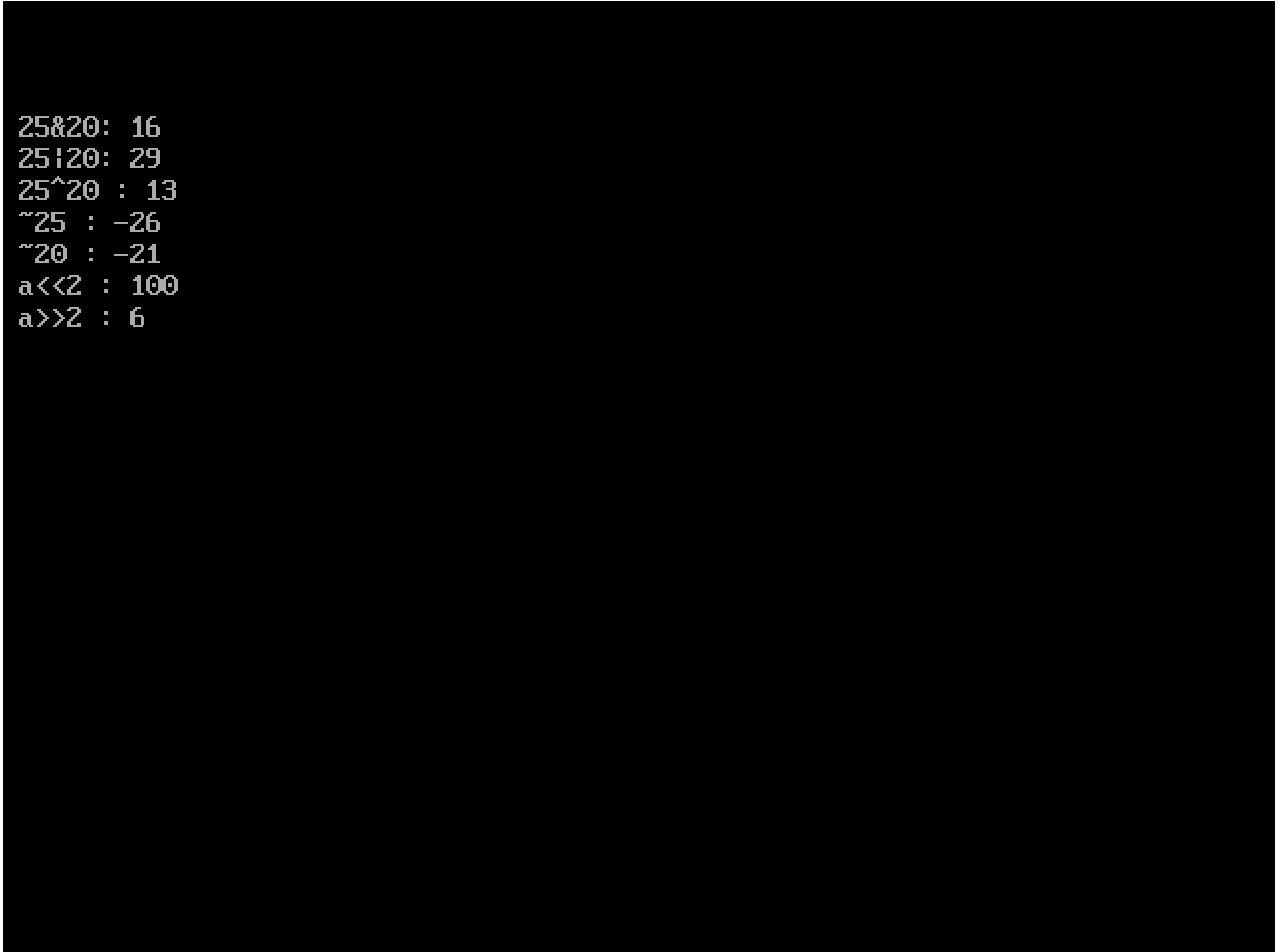
Bitwise Operators (&, |, ^, ~, >>, <<)

- The bitwise operators are used to perform bit-level operations in the c programming language. When we use the bitwise operators, the operations are performed based on the binary values. The following table describes all the bitwise operators in the C programming language. Let us consider two variables A and B as A = 25 (11001) and B = 20 (10100).

Operator	Meaning	Example
&	the result of Bitwise AND is 1 if all the bits are 1 otherwise it is 0	A & B ⇒ 16 (10000)
|	the result of Bitwise OR is 0 if all the bits are 0 otherwise it is 1	A | B ⇒ 29 (11101)
^	the result of Bitwise XOR is 0 if all the bits are same otherwise it is 1	A ^ B ⇒ 13 (01101)
~	the result of Bitwise once complement is negation of the bit (Flipping)	~A ⇒ 6 (00110)
<<	the Bitwise left shift operator shifts all the bits to the left by the specified number of positions	A << 2 ⇒ 100 (1100100)
>>	the Bitwise right shift operator shifts all the bits to the right by the specified number of positions	A >> 2 ⇒ 6 (00110)


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
BITWISE.C
1=[↕]
18:40
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
        int a,b;
        a=25;
        b=20;

clrscr();
        /* Example for Bitwise Operations*/

        printf("\n 25&20: %d", a&b);   //Bitwise And
        printf("\n 25|20: %d", a|b);   // Bitwise Or
        printf("\n 25^20 : %d",a^b);   //Bitwise Xor
        printf("\n ~25 : %d", ~a);     // 1s Complement of A
        printf("\n ~20 : %d", ~b);     // ones complement of B
        printf("\n a<<2 : %d",a<<2);   // Left Shift
        printf("\n a>>2 : %d", a>>2);  // Right Shift
getch();
}
```



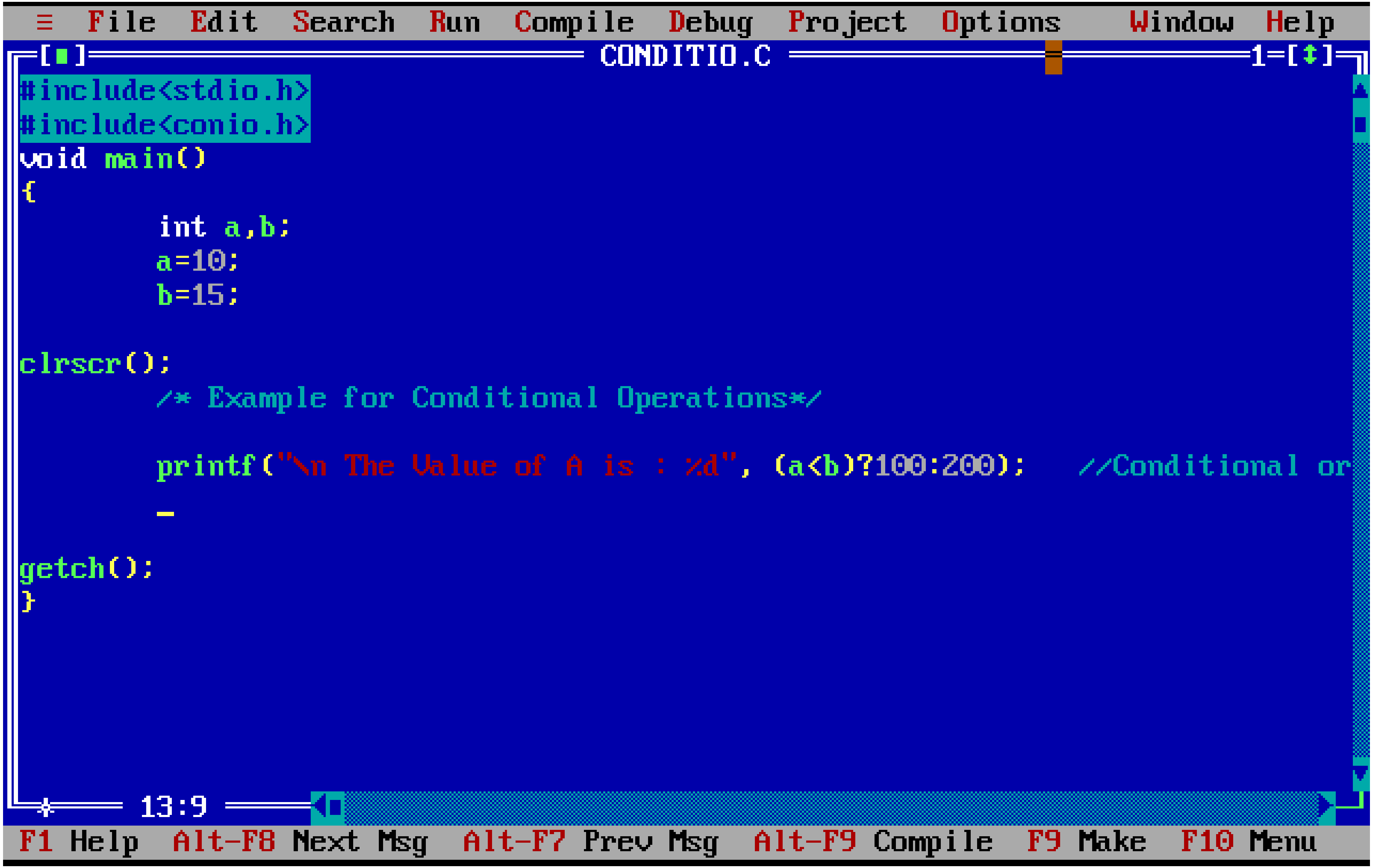
```
25&20: 16
25¦20: 29
25^20 : 13
~25 : -26
~20 : -21
a<<2 : 100
a>>2 : 6
```

Conditional Operator (?:)

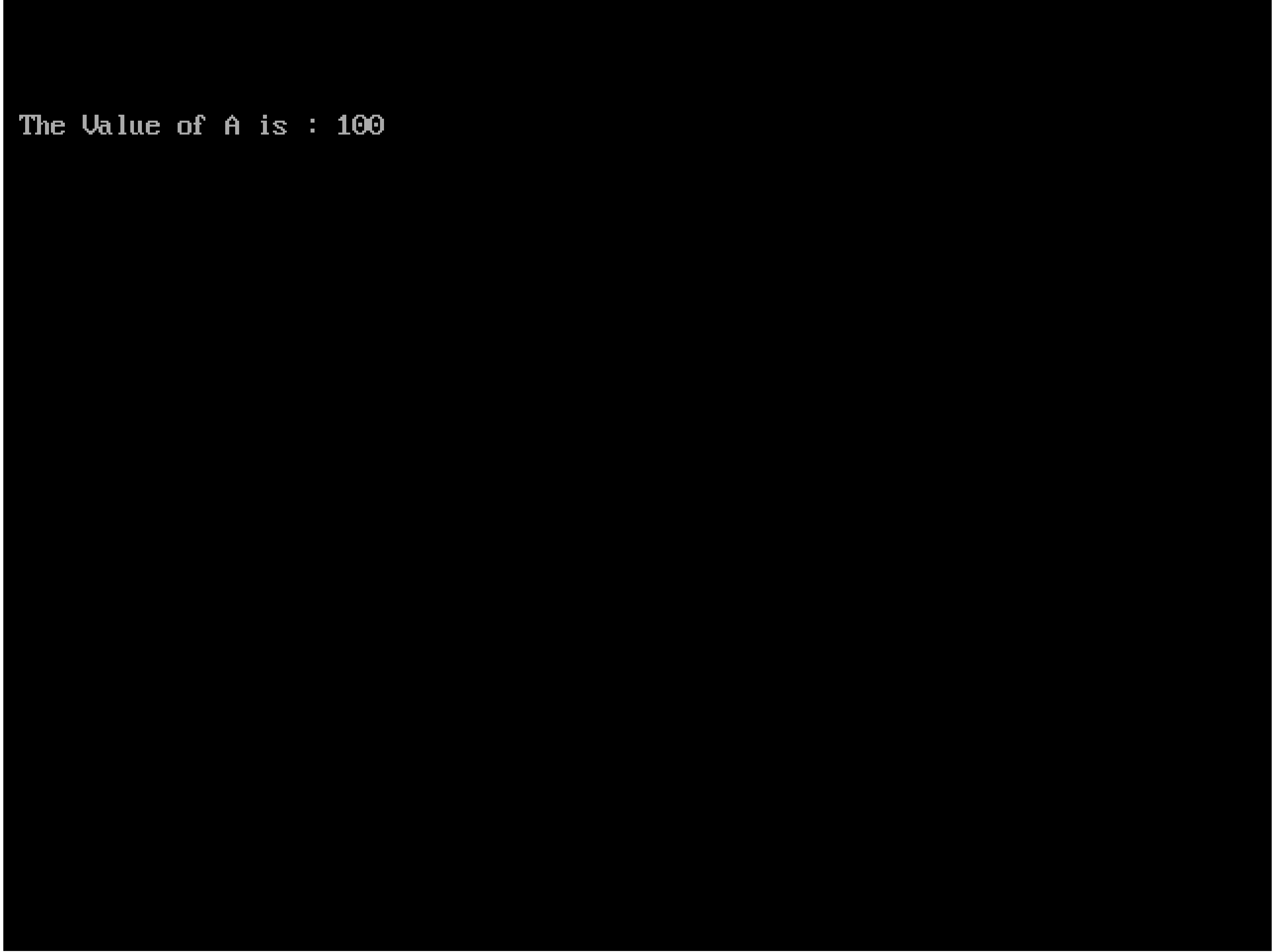
- The conditional operator is also called a ternary operator because it requires three operands. This operator is used for decision making. In this operator, first we verify a condition, then we perform one operation out of the two operations based on the condition result. If the condition is TRUE the first option is performed, if the condition is FALSE the second option is performed. The conditional operator is used with the following syntax.
- Condition ? TRUE Part : FALSE Part;

Example

A = (10<15)?100:200; ⇒ A value is 100


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
CONDITIO.C
1=[↕]
13:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a,b;
        a=10;
        b=15;

clrscr();
        /* Example for Conditional Operations*/

        printf("\n The Value of A is : %d", (a<b)?100:200);   //Conditional or

        _

getch();
}
```



```
The Value of A is : 100
```

Special Operators (sizeof, pointer, comma, dot, etc.)

- The following are the special operators in c programming language.

sizeof operator

- This operator is used to find the size of the memory (in bytes) allocated for a variable. This operator is used with the following syntax.
- sizeof(variableName);

  Example
- sizeof(A); ⇒ the result is 2 if A is an integer

Pointer operator (*)

- This operator is used to define pointer variables in c programming language.
- Comma operator (,)
- This operator is used to separate variables while they are declaring, separate the expressions in function calls, etc.

Dot operator (.)

- This operator is used to access members of structure or union.

Expression

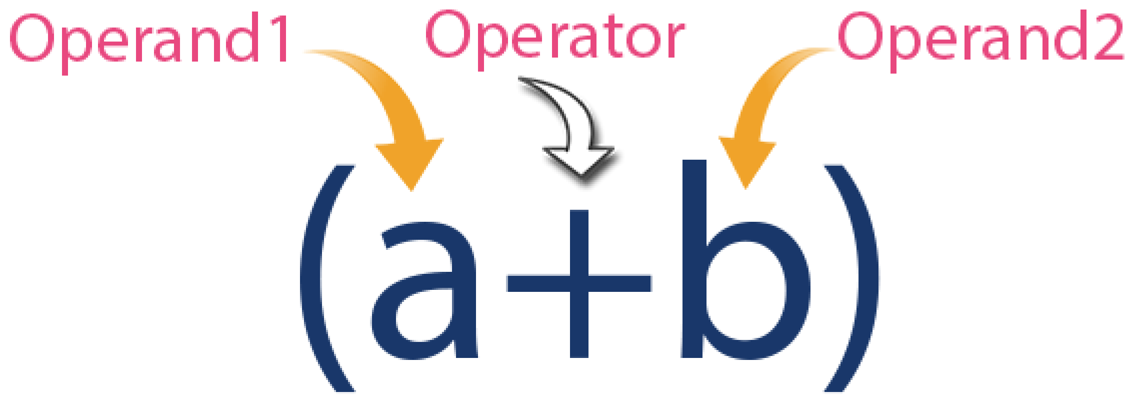
- In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.
- In the C programming language, an expression is defined as follows.
- **An expression is a collection of operators and operands that represents a specific value.**
- In the above definition, an operator is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.
- Operands are the values on which the operators perform the task. Here operand can be a direct value or variable or address of memory location.
- In the C programming language, expressions are divided into THREE types. They are as follows...
  - Infix Expression
  - Postfix Expression
  - Prefix Expression
- The above classification is based on the operator position in the expression.

Expression Types in C

- **Infix Expression**
  - The expression in which the operator is used between operands is called infix expression.
- The infix expression has the following general structure.

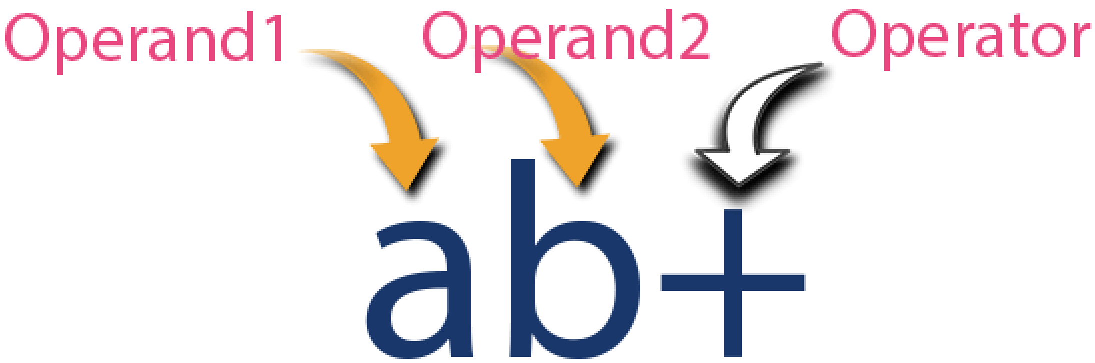
Operand1 Operator Operand2

Example

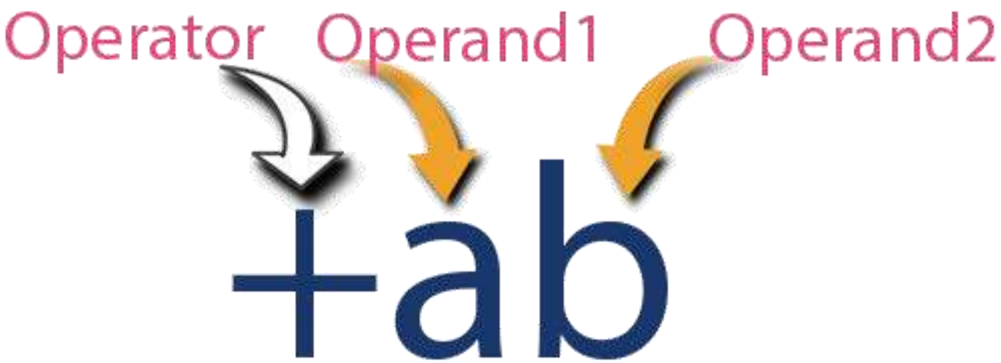


- **Postfix Expression**
- The expression in which the operator is used after operands is called postfix expression.
- The postfix expression has the following general structure.

  Operand1 Operand2 Operator
- Example



- Prefix Expression
- The expression in which the operator is used before operands is called a prefix expression.
- The prefix expression has the following general structure.
- Operator Operand1 Operand2
- Example



Expression Evaluation

- In the C programming language, an expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their precedence and associativity. The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.
- To understand expression evaluation in c, let us consider the following simple example expression...

  10 + 4 * 3 / 2
- In the above expression, there are three operators +, * and /. Among these three operators, both multiplication and division have the same higher precedence and addition has lower precedence. So, according to the operator precedence both multiplication and division are evaluated first and then the addition is evaluated. As multiplication and division have the same precedence they are evaluated based on the associativity. Here, the associativity of multiplication and division is left to right. So, multiplication is performed first, then division and finally addition. So, the above expression is evaluated in the order of * / and +. It is evaluated as follows...

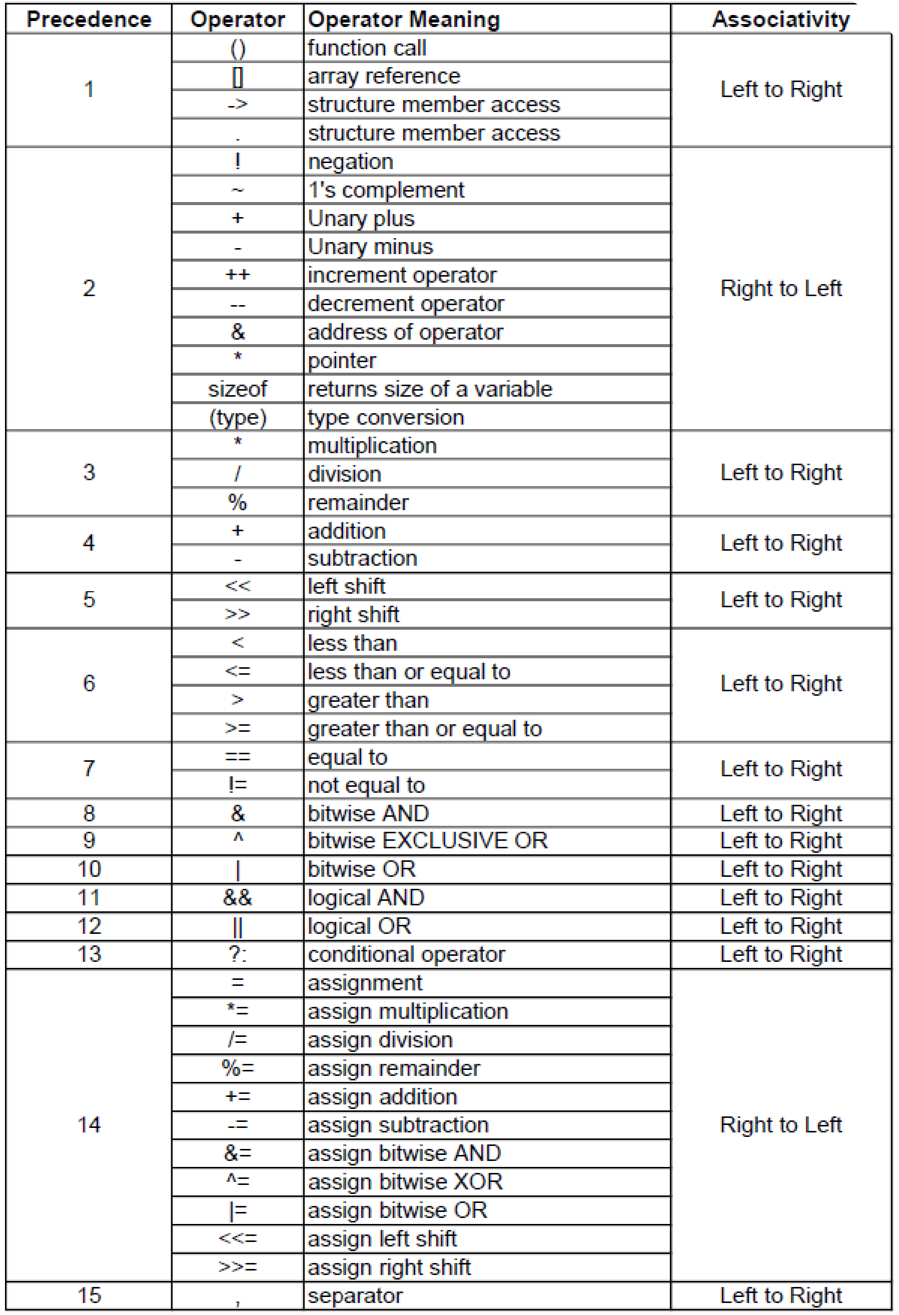
  4 * 3 ====> 12

  12 / 2 ===> 6

  10 + 6 ===> 16
- The expression is evaluated to 16.

Operator Precedence and Associativity

- Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.
- Operator associativity is used to determine the order of operators with equal precedence evaluated in an expression. In the c programming language, when an expression contains multiple operators with equal precedence, we use associativity to determine the order of evaluation of those operators.
- In c programming language the operator precedence and associativity are as shown in the following table.

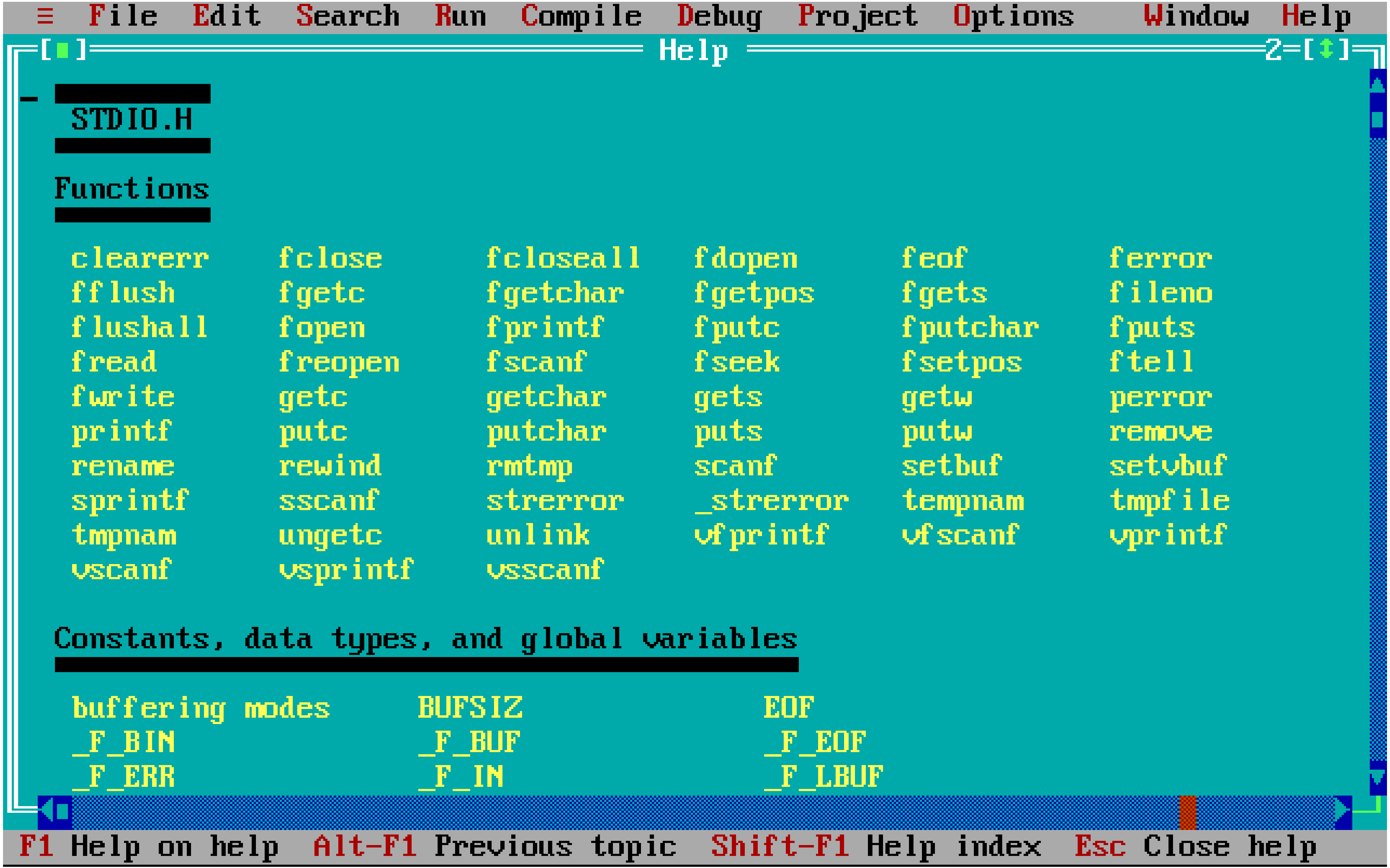


Precedence	Operator	Operator Meaning	Associativity
1	()	function call	Left to Right
	[]	array reference	
	->	structure member access	
	.	structure member access	
2	!	negation	Right to Left
	~	1's complement	
	+	Unary plus	
	-	Unary minus	
	++	increment operator	
	--	decrement operator	
	&	address of operator	
	*	pointer	
	sizeof	returns size of a variable	
	(type)	type conversion	
3	*	multiplication	Left to Right
	/	division	
	%	remainder	
4	+	addition	Left to Right
	-	subtraction	
5	<<	left shift	Left to Right
	>>	right shift	
6	<	less than	Left to Right
	<=	less than or equal to	
	>	greater than	
	>=	greater than or equal to	
7	==	equal to	Left to Right
	!=	not equal to	
8	&	bitwise AND	Left to Right
9	^	bitwise EXCLUSIVE OR	Left to Right
10	|	bitwise OR	Left to Right
11	&&	logical AND	Left to Right
12	||	logical OR	Left to Right
13	?:	conditional operator	Left to Right
14	=	assignment	Right to Left
	*=	assign multiplication	
	/=	assign division	
	%=	assign remainder	
	+=	assign addition	
	-=	assign subtraction	
	&=	assign bitwise AND	
	^=	assign bitwise XOR	
	|=	assign bitwise OR	
	<<=	assign left shift	
	>>=	assign right shift	
15	,	separator	Left to Right

Library Functions

- The standard functions are built-in functions. In C programming language, the standard functions are declared in header files. The standard functions are also called as library functions or pre-defined functions.
- In C when we use standard functions, we must include the respective header file using #include statement. For example, the function printf() is defined in header file stdio.h (Standard Input Output header file). When we use printf() in our program, we must include stdio.h header file using #include<stdio.h> statement.
- The C standard library provides macros, type definitions and functions for tasks such as string handling, mathematical computations, input/output processing, memory management, and several other operating system services.
- C Programming Language provides the following header files with standard functions.

Header File	Purpose	Example Functions
stdio.h	Provides functions to perform standard I/O operations	printf(), scanf()
conio.h	Provides functions to perform console I/O operations	clrscr(), getch()
math.h	Provides functions to perform mathematical operations	sqrt(), pow()
string.h	Provides functions to handle string data values	strlen(), strcpy()
stdlib.h	Provides functions to perform general functions/td>	calloc(), malloc()
time.h	Provides functions to perform operations on time and date	time(), localtime()
ctype.h	Provides functions to perform - testing and mapping of character data values	isalpha(), islower()
setjmp.h	Provides functions that are used in function calls	setjump(), longjump()
signal.h	Provides functions to handle signals during program execution	signal(), raise()
assert.h	Provides Macro that is used to verify assumptions made by the program	assert()
locale.h	Defines the location specific settings such as date formats and currency symbols	setlocale()
stdarg.h	Used to get the arguments in a function if the arguments are not specified by the function	va_start(), va_end()
errno.h	Provides macros to handle the system calls	Error, errno
graphics.h	Provides functions to draw graphics.	circle(), rectangle()
float.h	Provides constants related to floating point data values	
stddef.h	Defines various variable types	
limits.h	Defines the maximum and minimum values of various variable types like char, int and long	



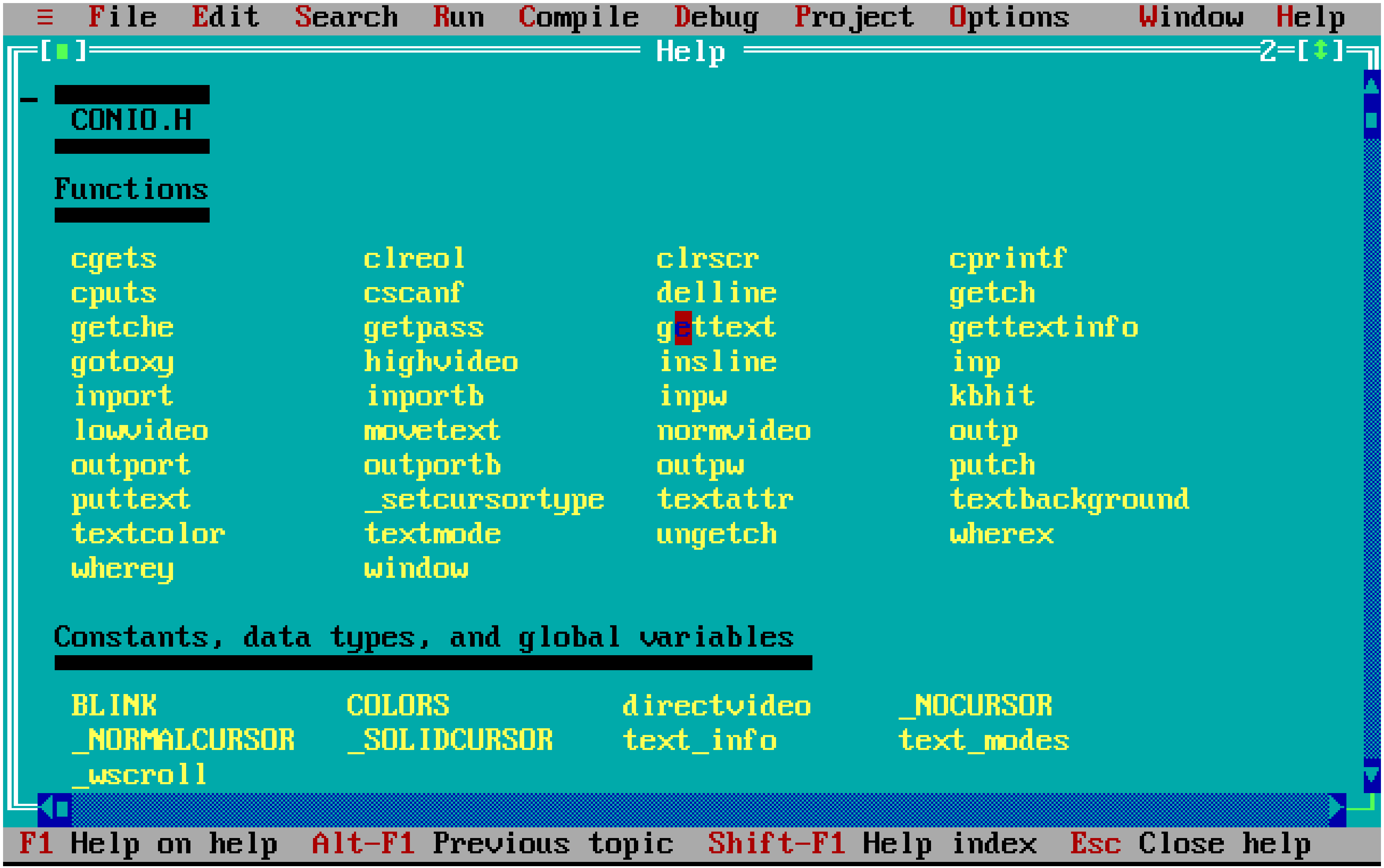
STDIO.H

Functions

```
 clearerr    fclose      fcloseall   fdopen      feof        ferror
 fflush      fgetc       fgetchar    fgetpos     fgets       fileno
 flushall    fopen       fprintf     fputc       fputchar    fputs
 fread       freopen     fscanf      fseek       fsetpos     ftell
 fwrite      getc        getchar     gets        getw        perror
 printf      putc        putchar     puts        putw        remove
 rename      rewind      rmtmp       scanf       setbuf      setvbuf
 sprintf     sscanf      strerror    _strerror   tempnam     tmpfile
 tmpnam      ungetc      unlink      vfprintf    vfscanf     vprintf
 vscanf      vsprintf    vsscanf
```

Constants, data types, and global variables

```
 buffering modes     BUFSIZ              EOF
 _F_BIN              _F_BUF              _F_EOF
 _F_ERR              _F_IN               _F_LBUF
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] Help 2=[↕]
F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help

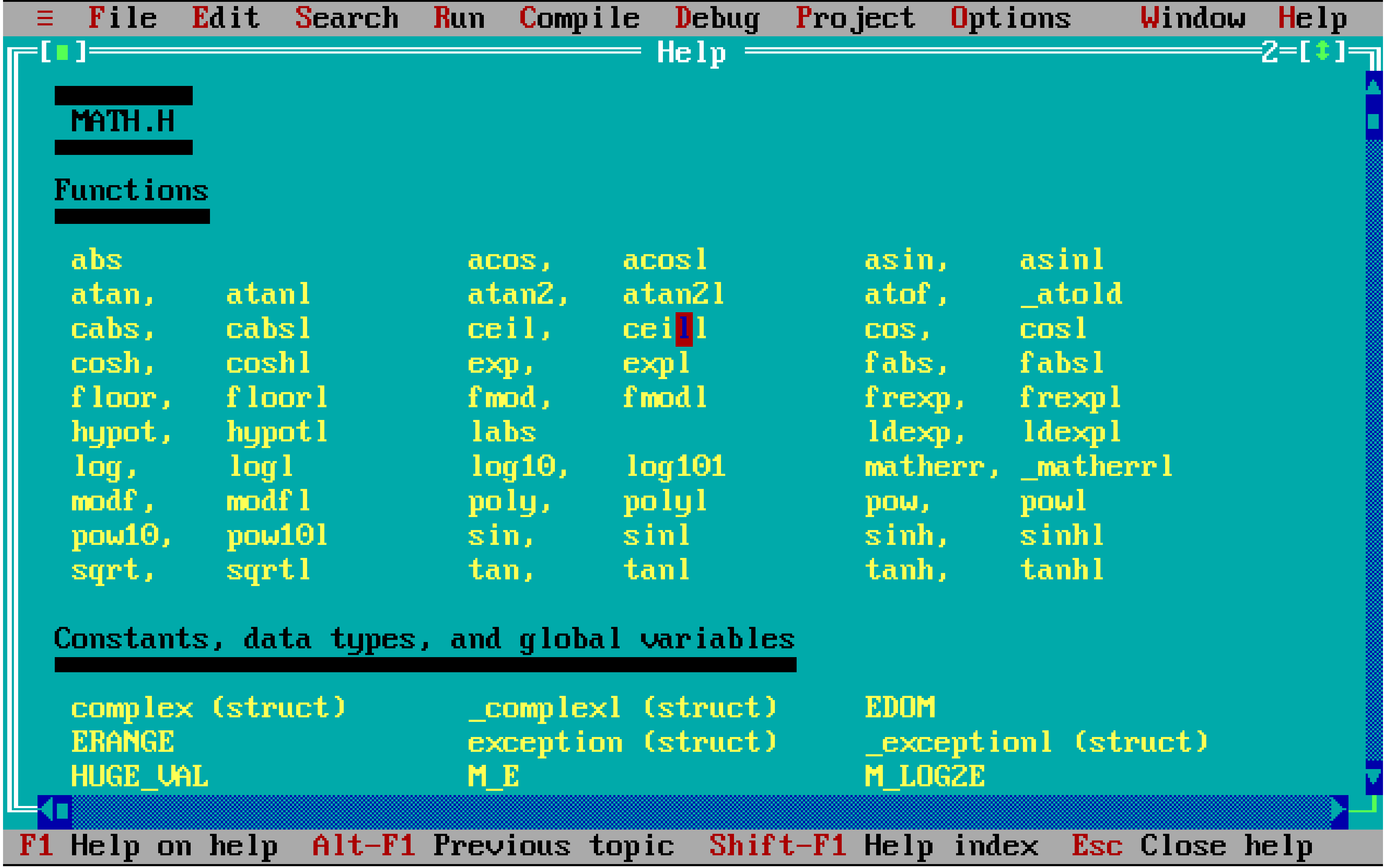
CONIO.H

Functions

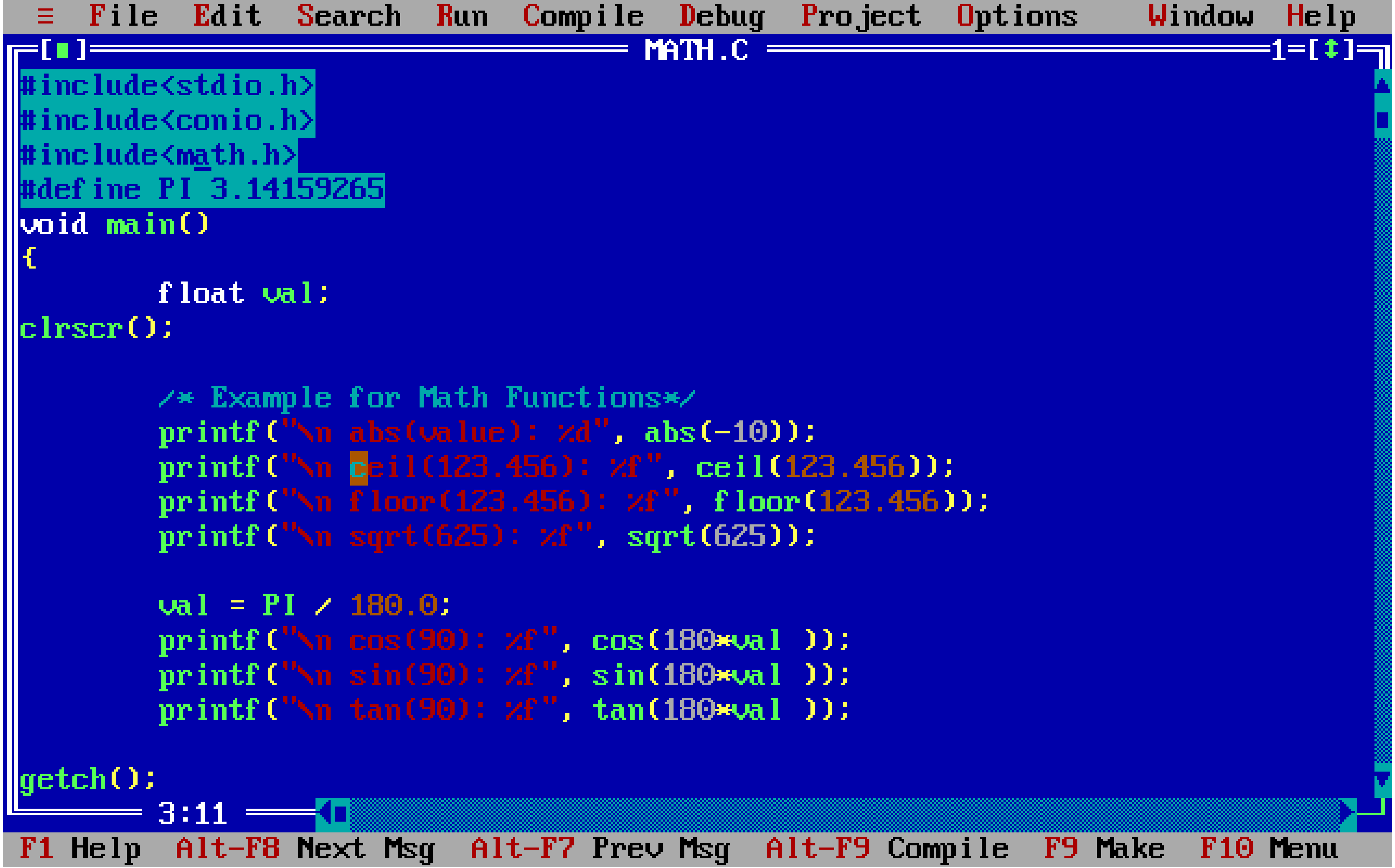
```
cgets       clreol          clrscr      cprintf
cputs       cscanf          delline     getch
getche      getpass         gettext     gettextinfo
gotoxy      highvideo       insline     inp
inport      inportb         inpw        kbhit
lowvideo    movetext        normvideo   outp
outport     outportb        outpw       putch
puttext     _setcursortype  textattr    textbackground
textcolor   textmode        ungetch     wherex
wherey      window
```

Constants, data types, and global variables

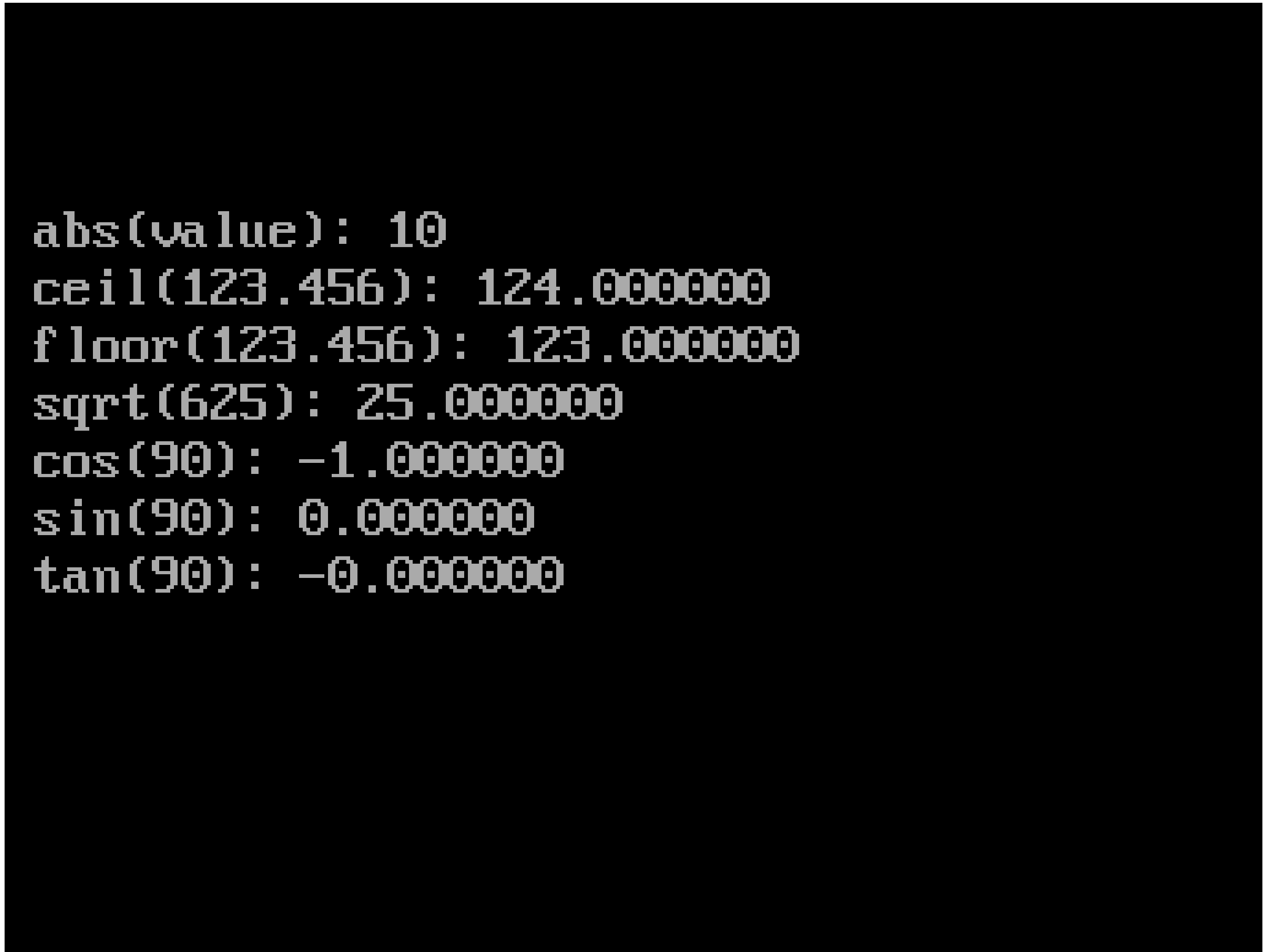
```
BLINK            COLORS          directvideo   _NOCURSOR
_NORMALCURSOR    _SOLIDCURSOR    text_info     text_modes
_wscroll
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] Help 2 [↕]
F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help

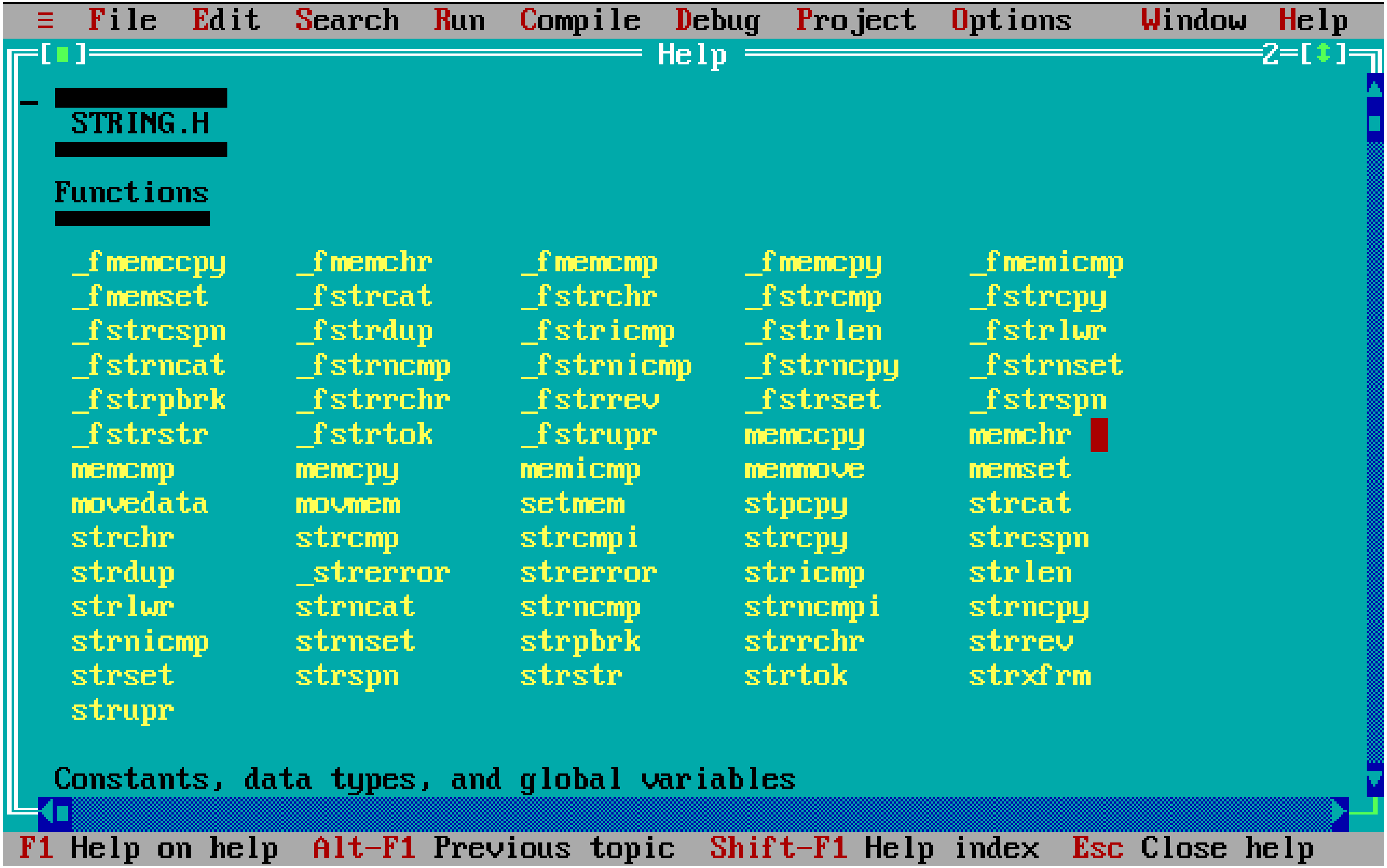
```
MATH.H

Functions

 abs                    acos,    acosl         asin,    asinl
 atan,    atanl         atan2,   atan2l        atof,    _atold
 cabs,    cabsl         ceil,    ceill         cos,     cosl
 cosh,    coshl         exp,     expl          fabs,    fabsl
 floor,   floorl        fmod,    fmodl         frexp,   frexpl
 hypot,   hypotl        labs                   ldexp,   ldexpl
 log,     logl          log10,   log10l        matherr, _matherrl
 modf,    modfl         poly,    polyl         pow,     powl
 pow10,   pow10l        sin,     sinl          sinh,    sinhl
 sqrt,    sqrtl         tan,     tanl          tanh,    tanhl

Constants, data types, and global variables

 complex (struct)       _complexl (struct)     EDOM
 ERANGE                 exception (struct)     _exceptionl (struct)
 HUGE_VAL               M_E                    M_LOG2E
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
MATH.C
1=[↕]
3:11
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define PI 3.14159265
void main()
{
        float val;
clrscr();

        /* Example for Math Functions*/
        printf("\n abs(value): %d", abs(-10));
        printf("\n ceil(123.456): %f", ceil(123.456));
        printf("\n floor(123.456): %f", floor(123.456));
        printf("\n sqrt(625): %f", sqrt(625));

        val = PI / 180.0;
        printf("\n cos(90): %f", cos(180*val ));
        printf("\n sin(90): %f", sin(180*val ));
        printf("\n tan(90): %f", tan(180*val ));


getch();
```



```
abs(value): 10
ceil(123.456): 124.000000
floor(123.456): 123.000000
sqrt(625): 25.000000
cos(90): -1.000000
sin(90): 0.000000
tan(90): -0.000000
```


≡ File Edit Search Run Compile Debug Project Options Window Help
Help
2
F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help

```
STRING.H

Functions

  _fmemccpy    _fmemchr     _fmemcmp     _fmemcpy     _fmemicmp
  _fmemset     _fstrcat     _fstrchr     _fstrcmp     _fstrcpy
  _fstrcspn    _fstrdup     _fstricmp    _fstrlen     _fstrlwr
  _fstrncat    _fstrncmp    _fstrnicmp   _fstrncpy    _fstrnset
  _fstrpbrk    _fstrrchr    _fstrrev     _fstrset     _fstrspn
  _fstrstr     _fstrtok     _fstrupr     memccpy      memchr
  memcmp       memcpy       memicmp      memmove      memset
  movedata     movmem       setmem       stpcpy       strcat
  strchr       strcmp       strcmpi      strcpy       strcspn
  strdup       _strerror    strerror     stricmp      strlen
  strlwr       strncat      strncmp      strncmpi     strncpy
  strnicmp     strnset      strpbrk      strrchr      strrev
  strset       strspn       strstr       strtok       strxfrm
  strupr

Constants, data types, and global variables
```

Thank you