PROGRAMMING IN C

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

06-12-2021

- III BSC Mathematics

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

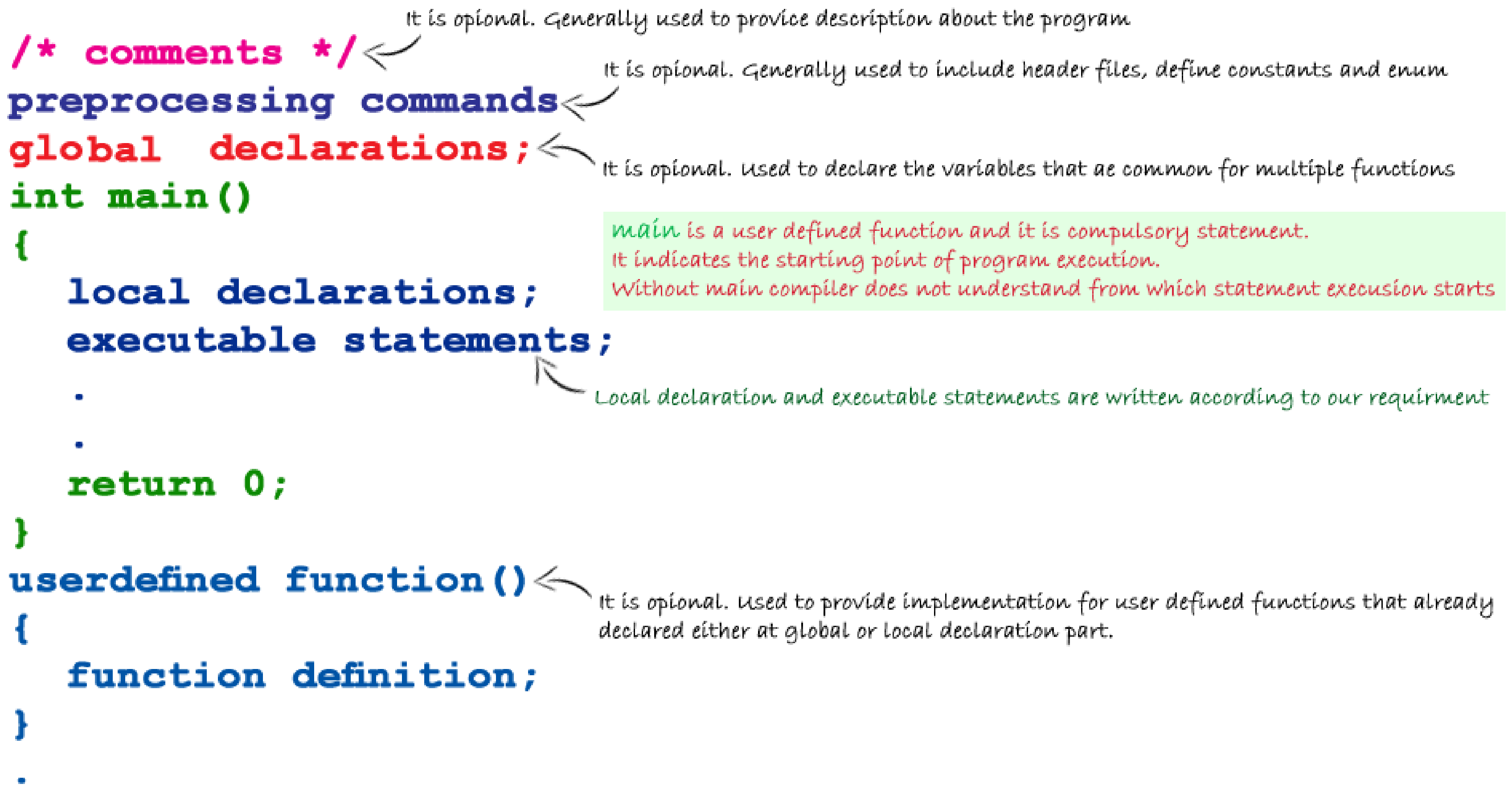
Features

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.

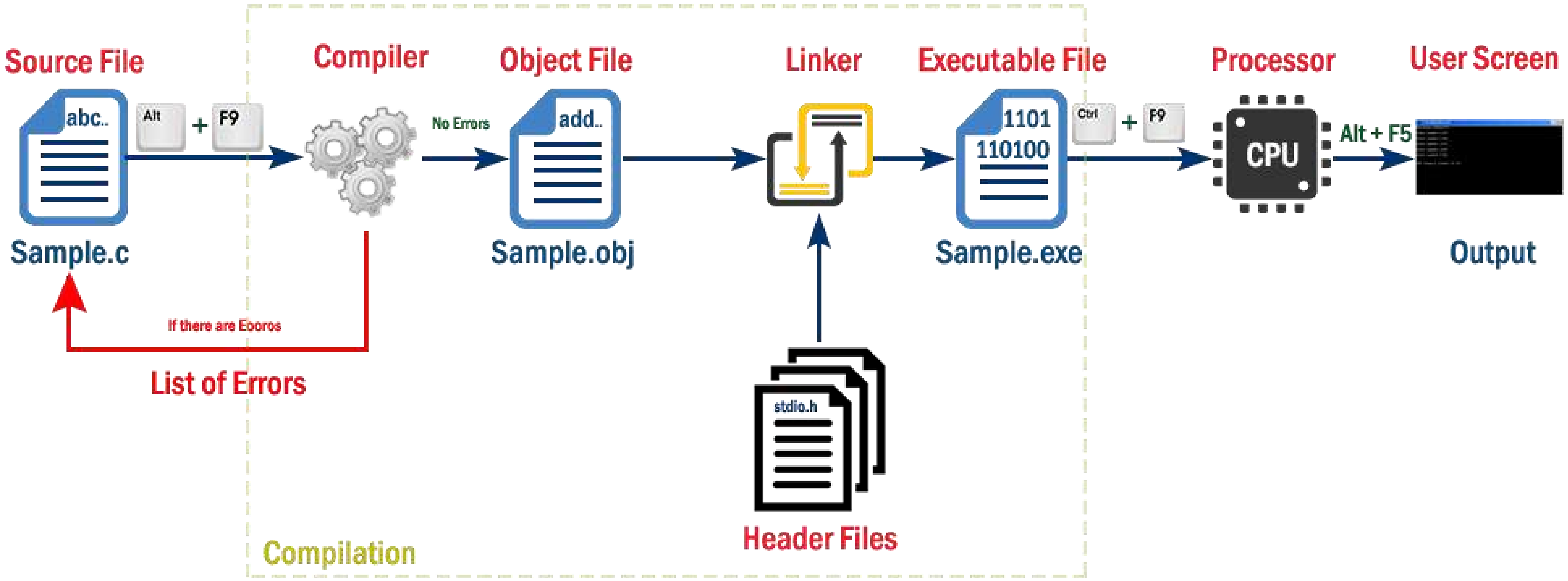
Step 3: Executing / Running Executable File (Ctrl + F9)

- 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	Invalid Identifiers
int a,b;	int a b;
float _a;	float 123a;
char _123;	char str-;
double pi;	double pi, a;
int value,Value,vAlue;	int break;
int Auto;	

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

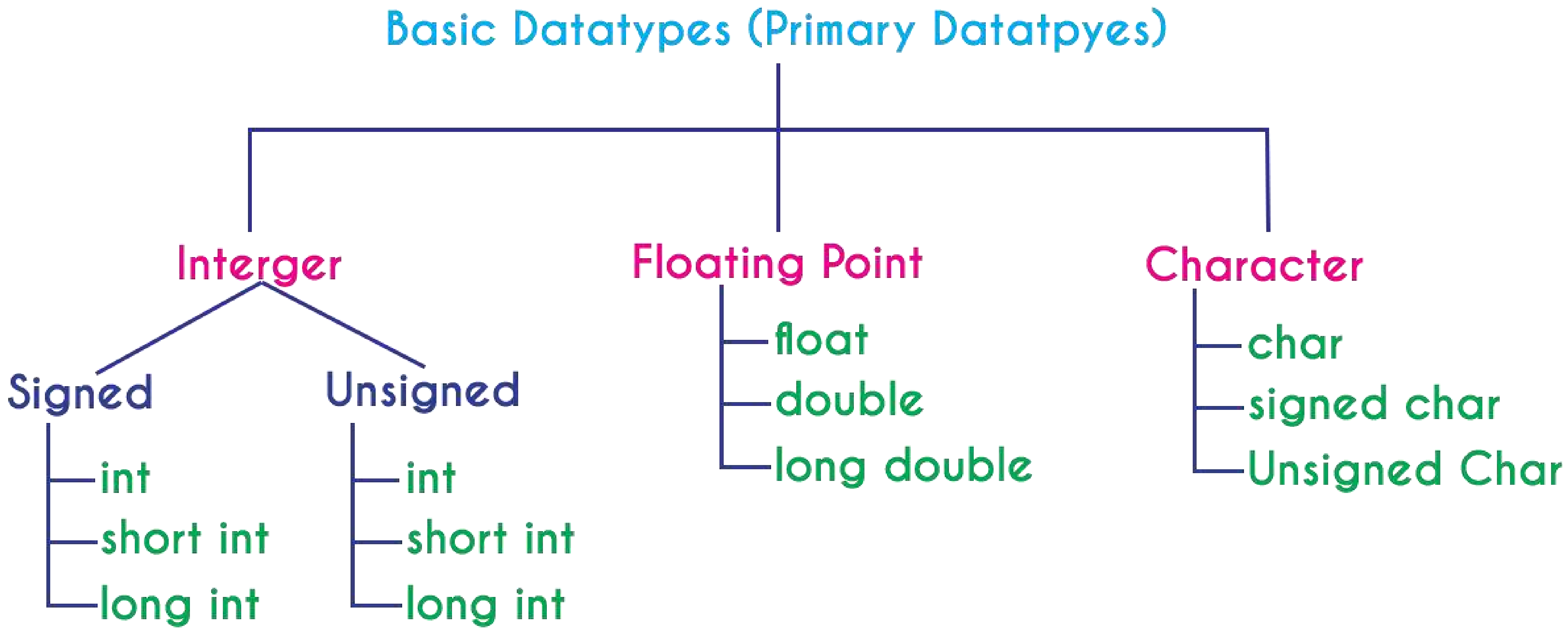
3. void type

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

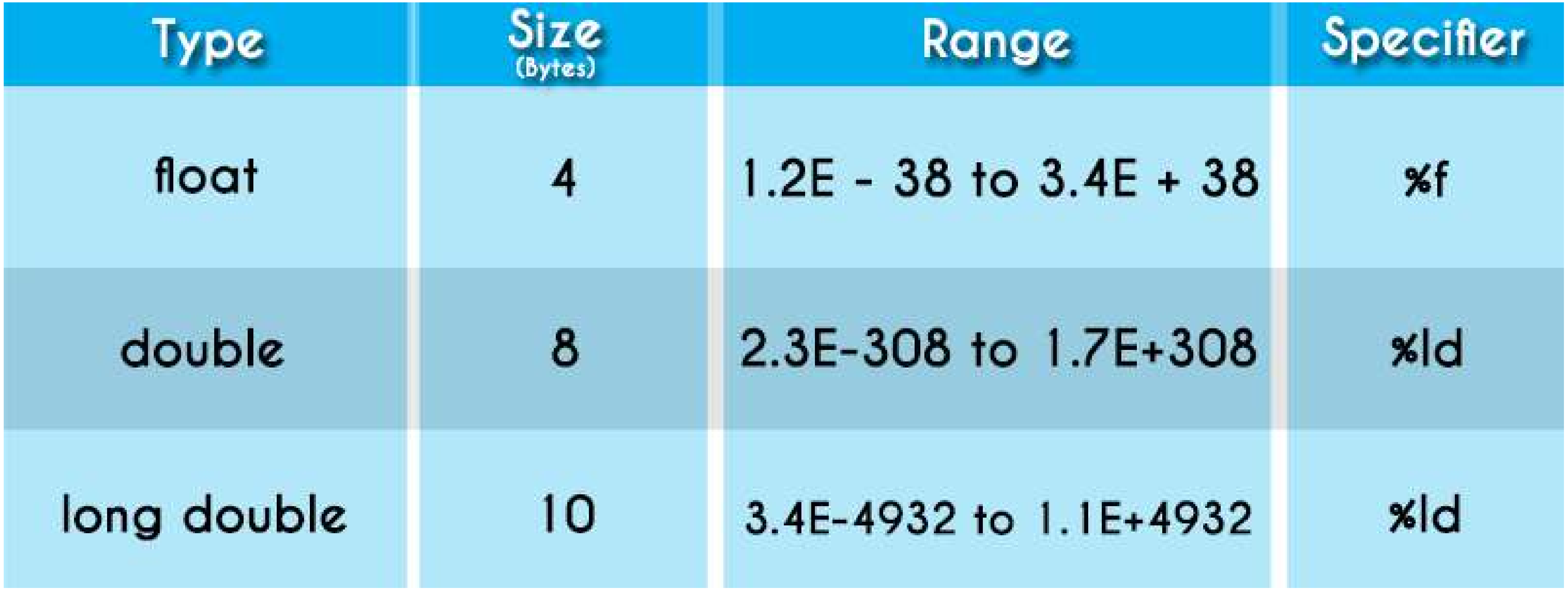
Integer Data type

- 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

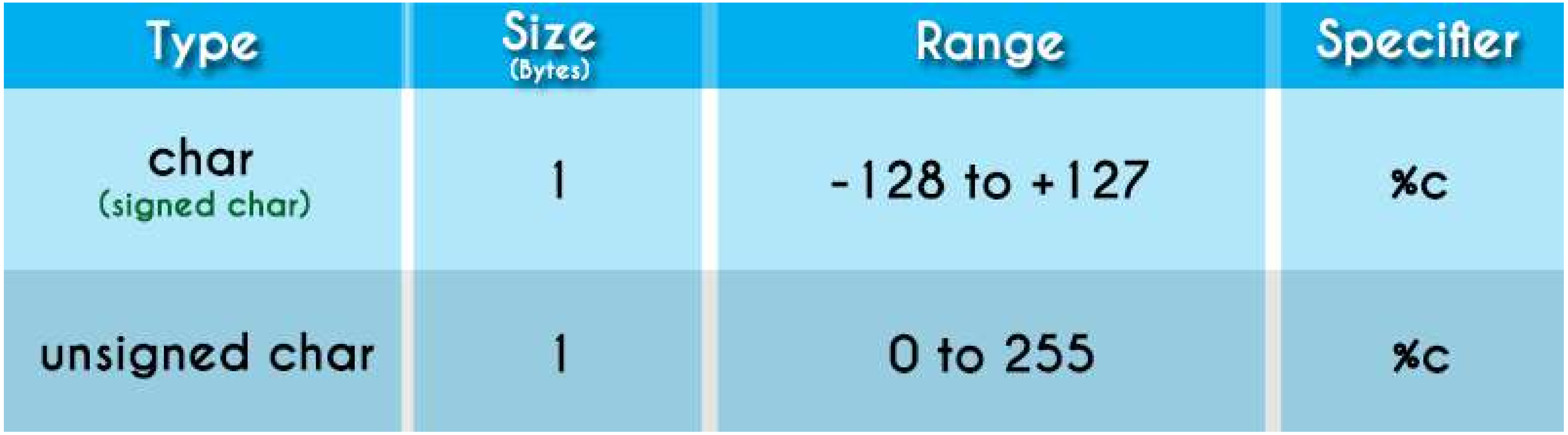
- Floating-point data types are a set of numbers with the decimal value. Every floating-point value must contain the decimal value. The floating-point data type has two variants...
  - float
  - double
- We use the keyword "float" to represent floating-point data type and "double" to represent double 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

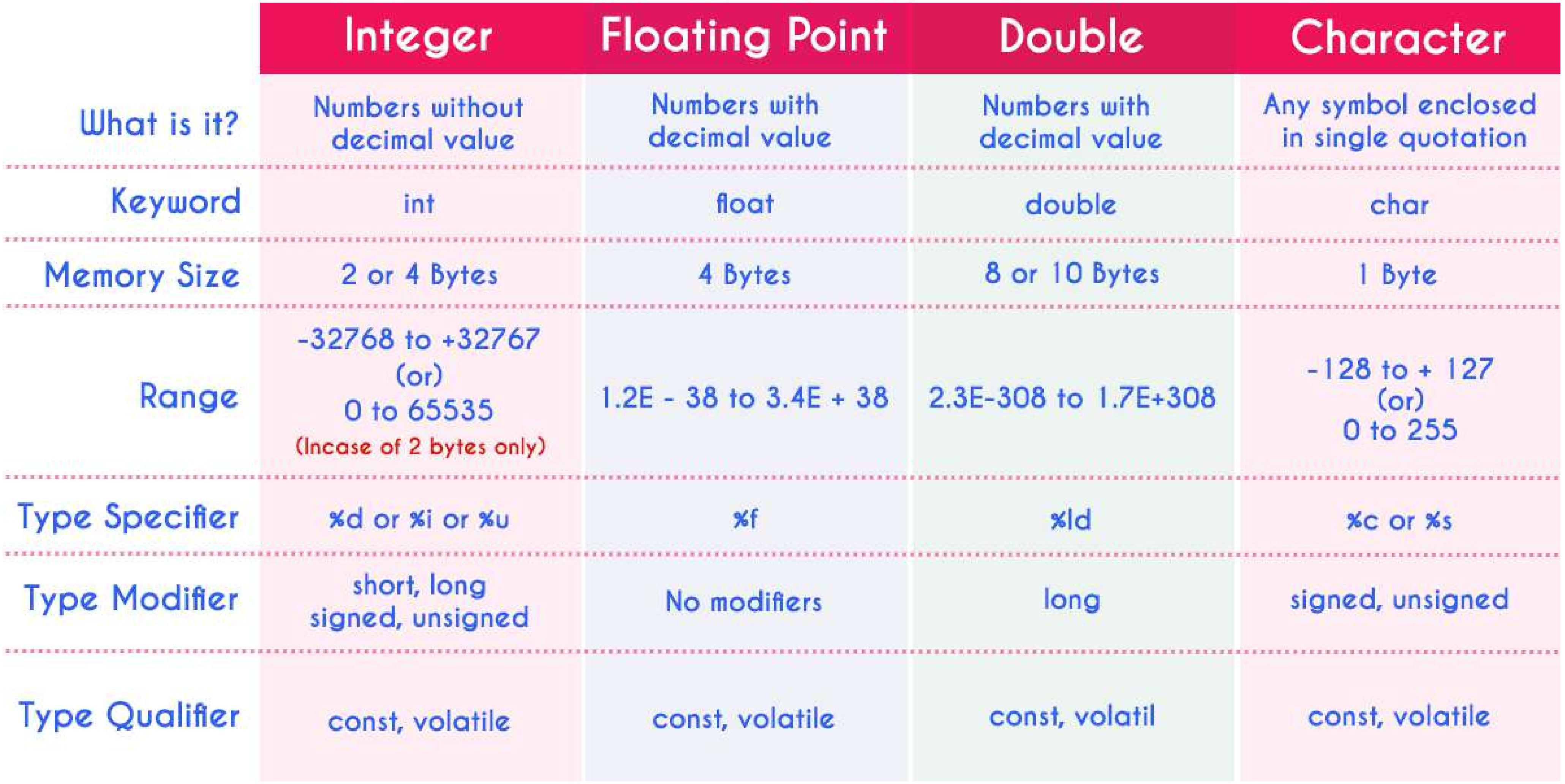
Character Data Type

- 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) ;
}
```

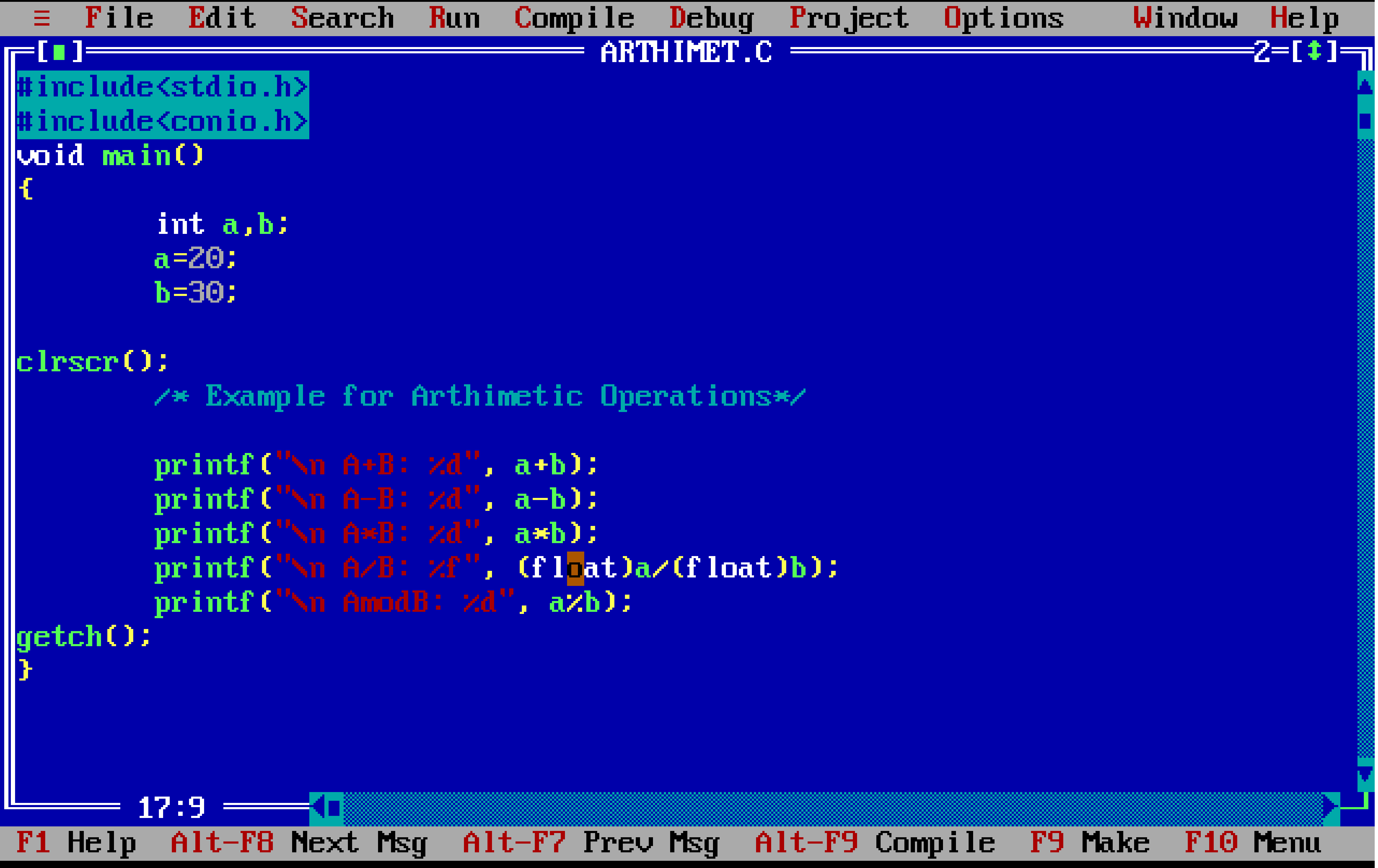
Operators

- 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

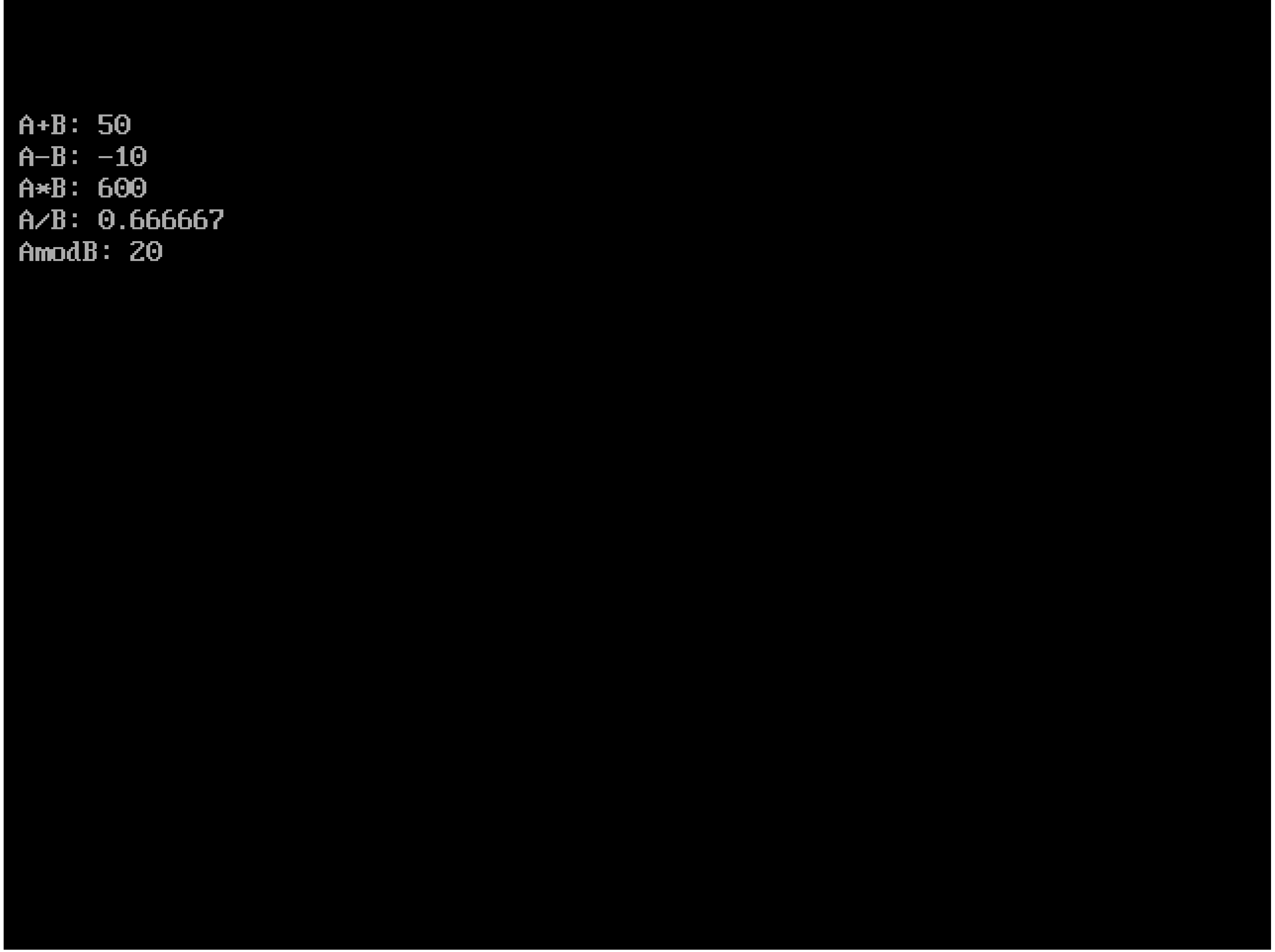
Arithmetic Operators (+, -, *, /, %)

- 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();
}
```

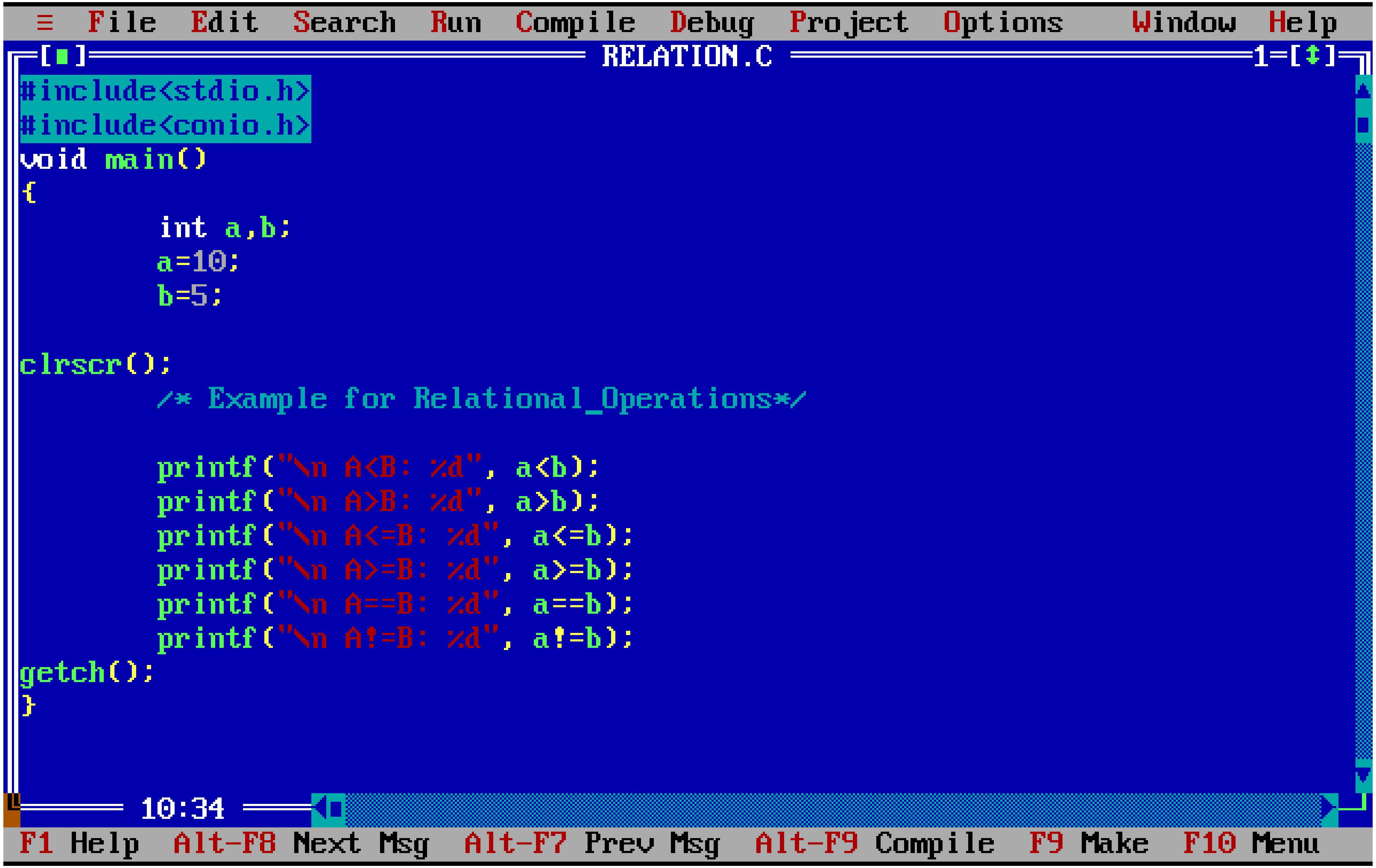


```
A+B: 50
A-B: -10
A*B: 600
A/B: 0.666667
AmodB: 20
```

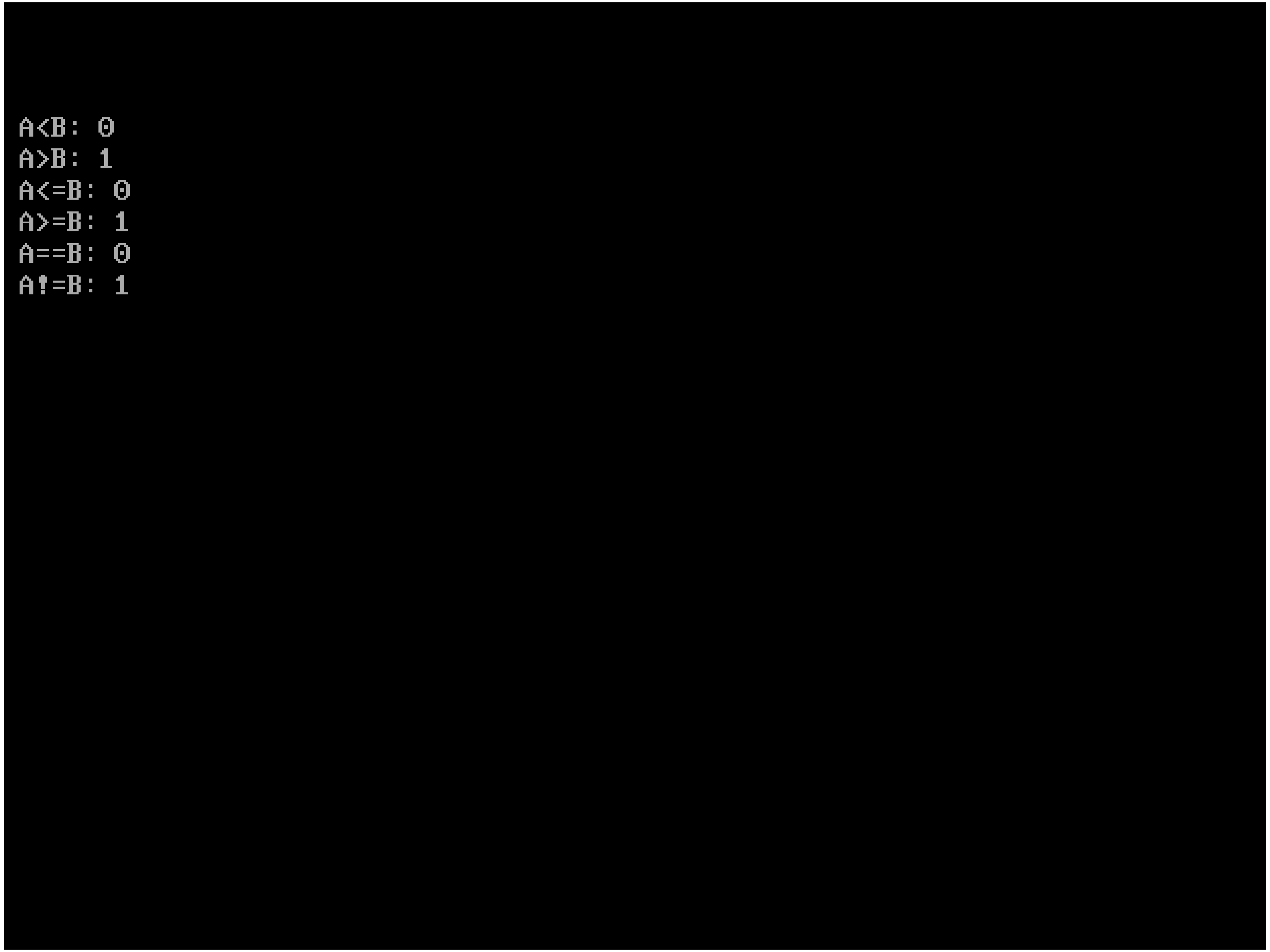
Relational Operators (<, >, <=, >=, ==, !=)

- 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();
}
```

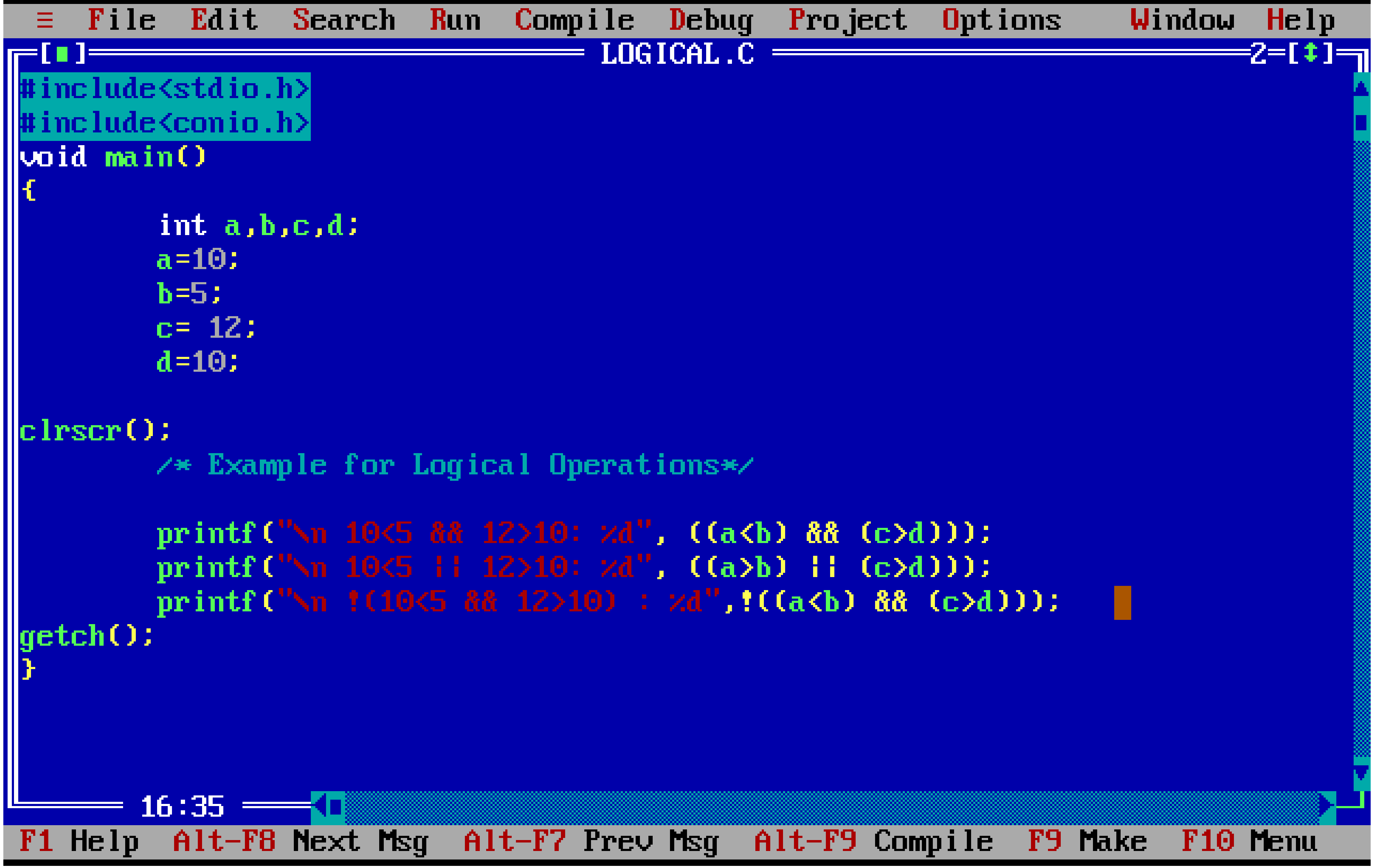


```
A<B: 0
A>B: 1
A<=B: 0
A>=B: 1
A==B: 0
A!=B: 1
```

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

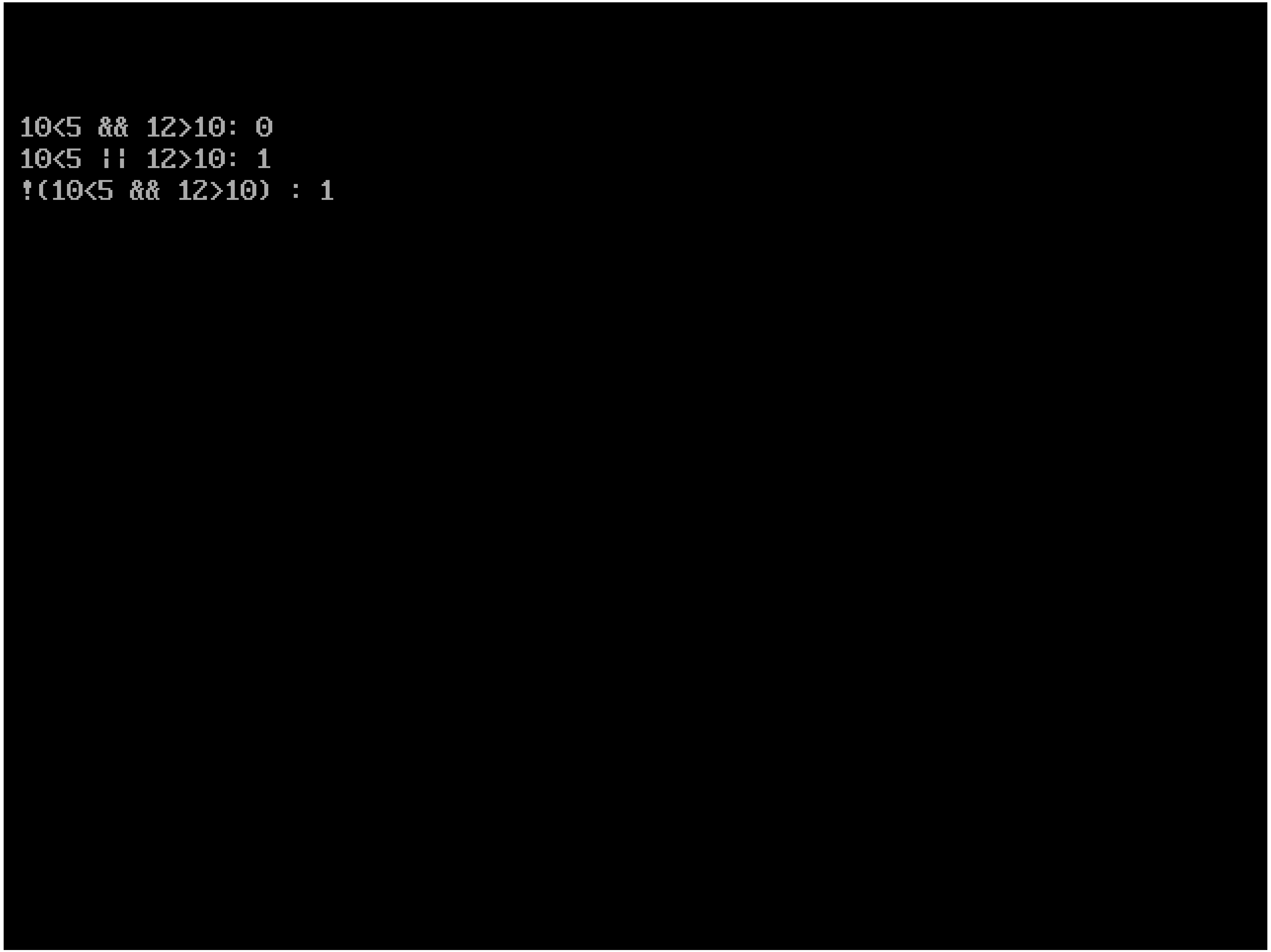

≡ File Edit Search Run Compile Debug Project Options Window Help

LOGICAL.C

```
#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();
}
```

16:35

F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu



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

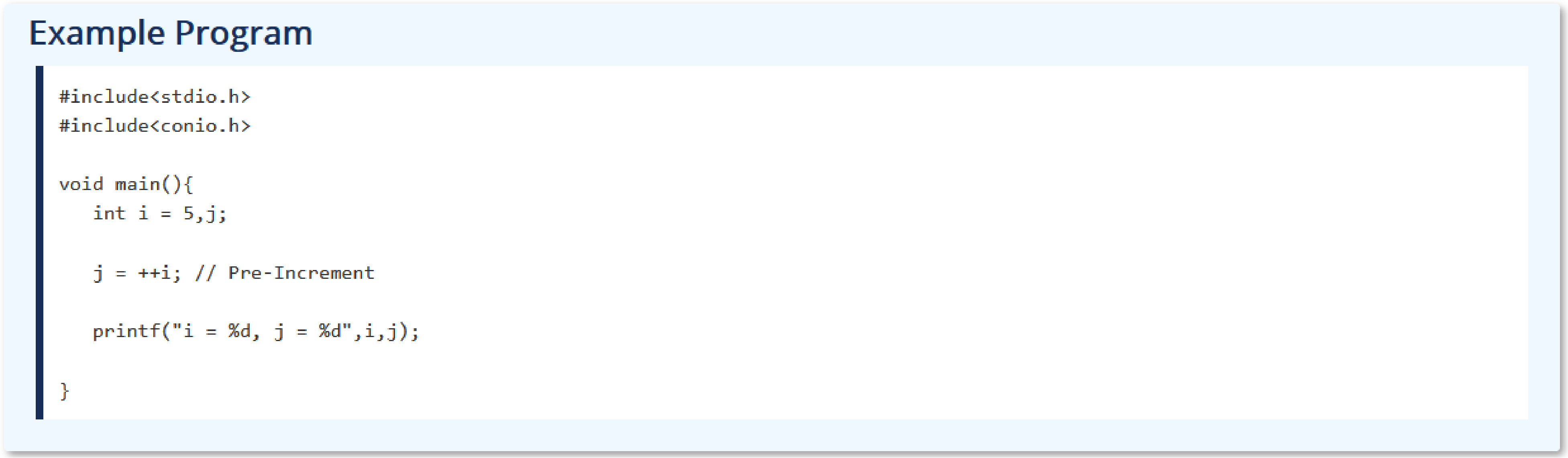
Increment & Decrement Operators (++ & --)

- 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.

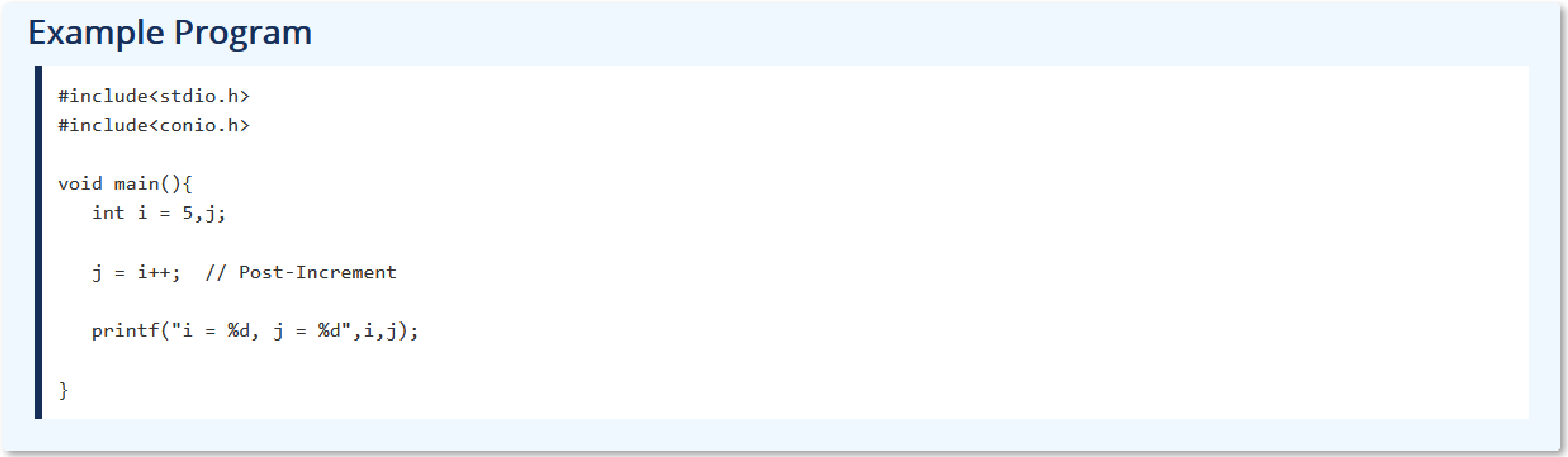


Example Program

```
#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.



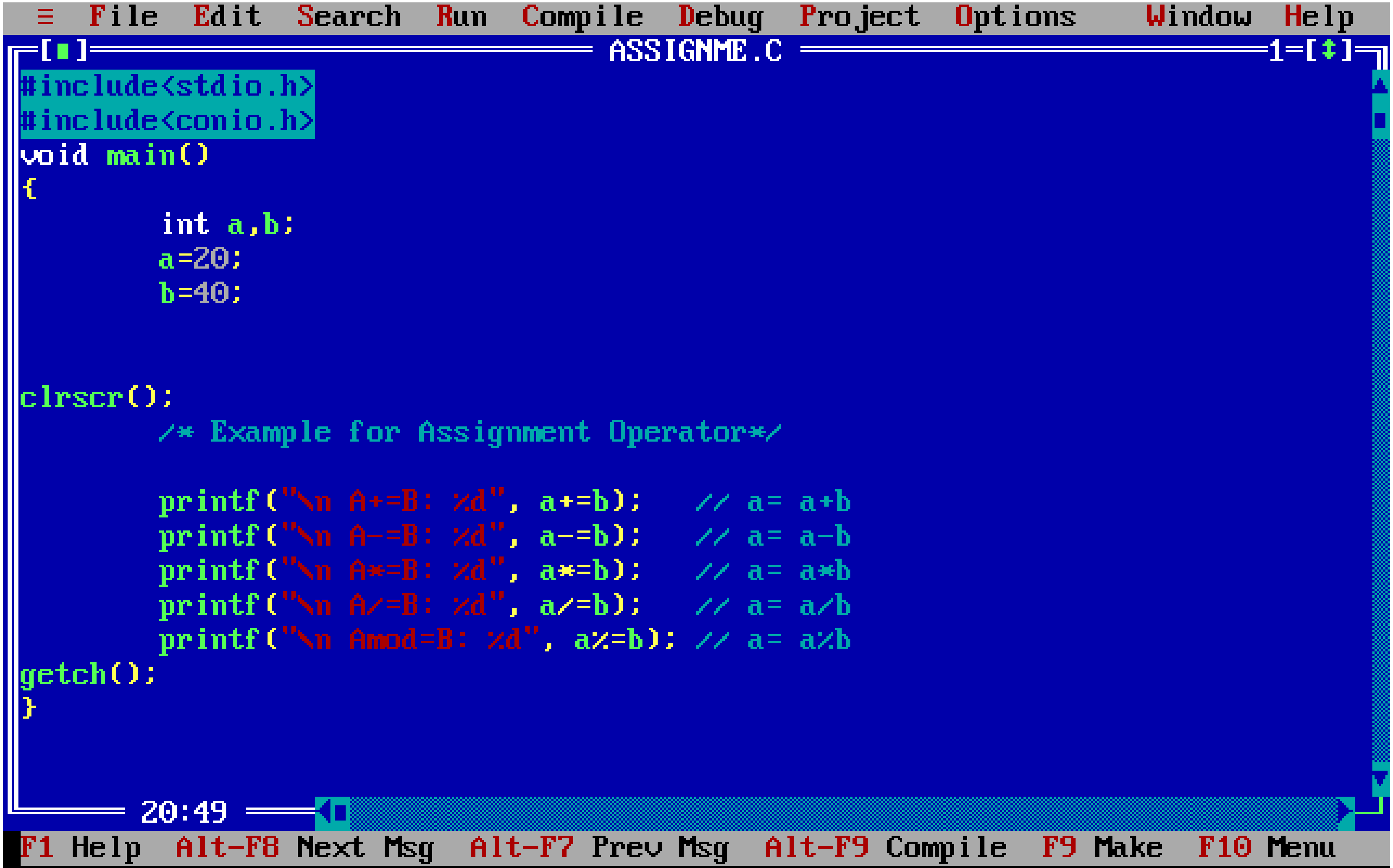
Example Program

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

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

   j = i++;  // Post-Increment

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

}
```

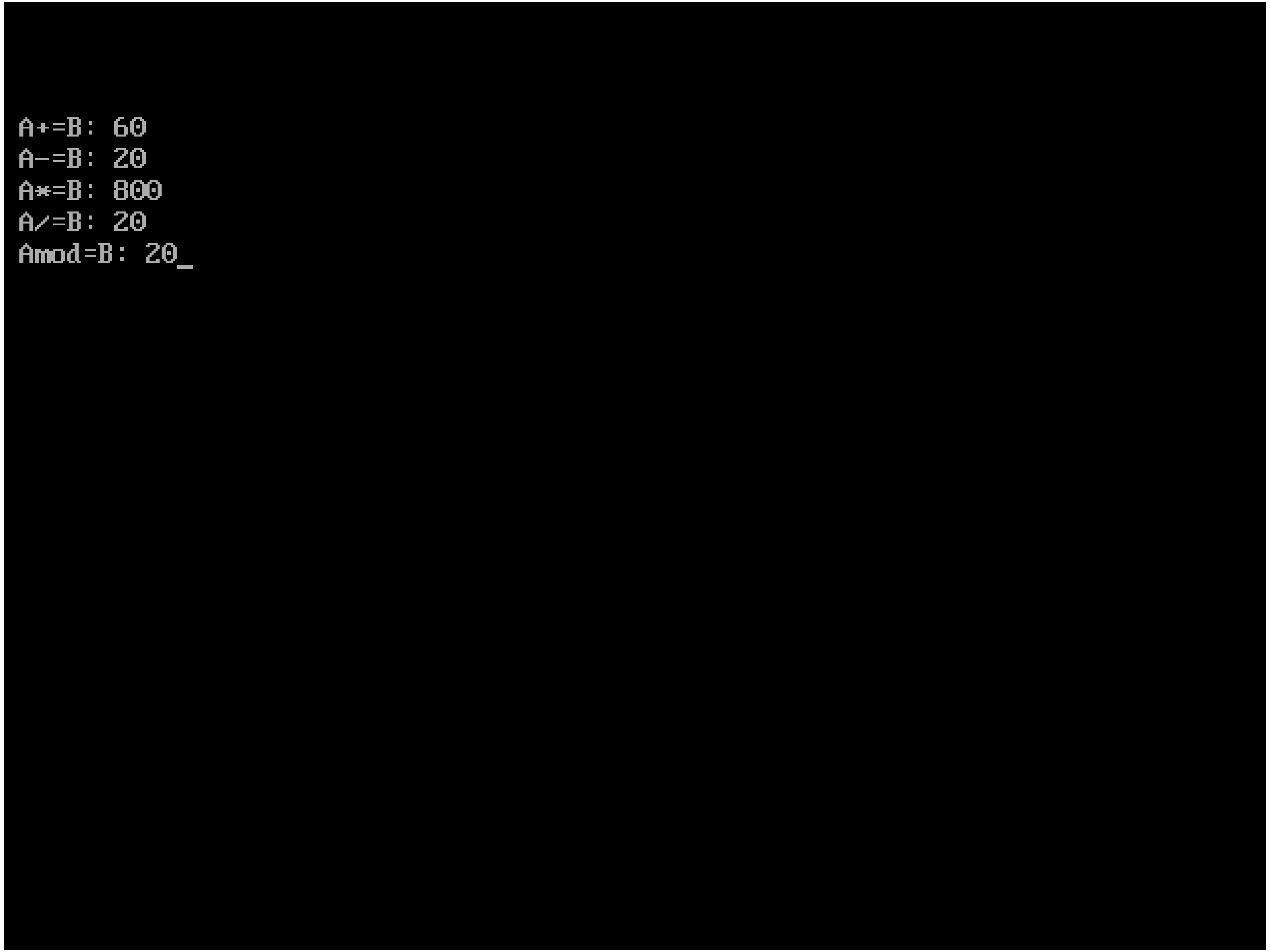
Assignment Operators (=, +=, -=, *=, /=, %=)

- 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();
}
```

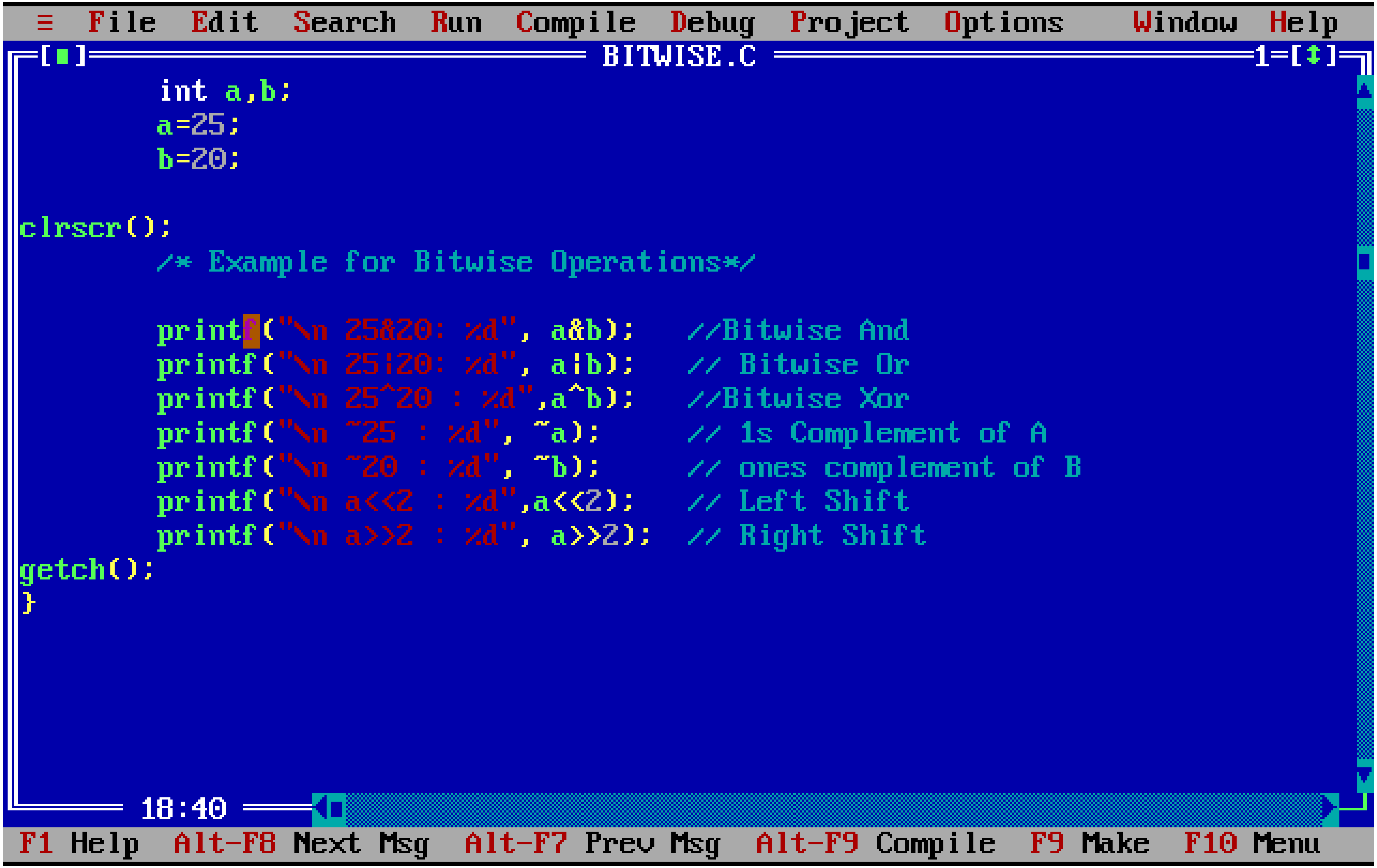


```
A+=B: 60
A-=B: 20
A*=B: 800
A/=B: 20
Amod=B: 20_
```

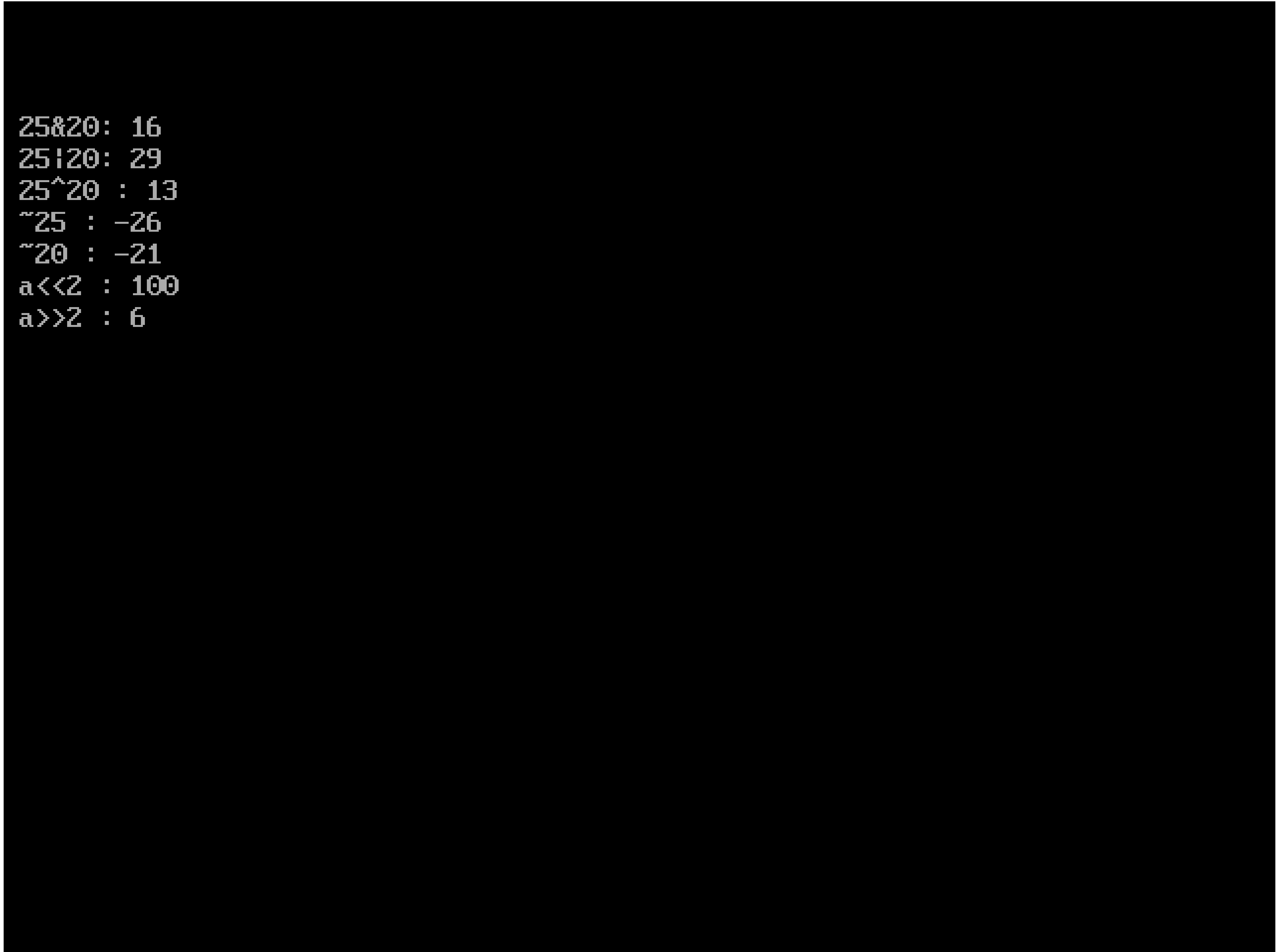
Bitwise Operators (&, |, ^, ~, >>, <<)

- 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();
}
```



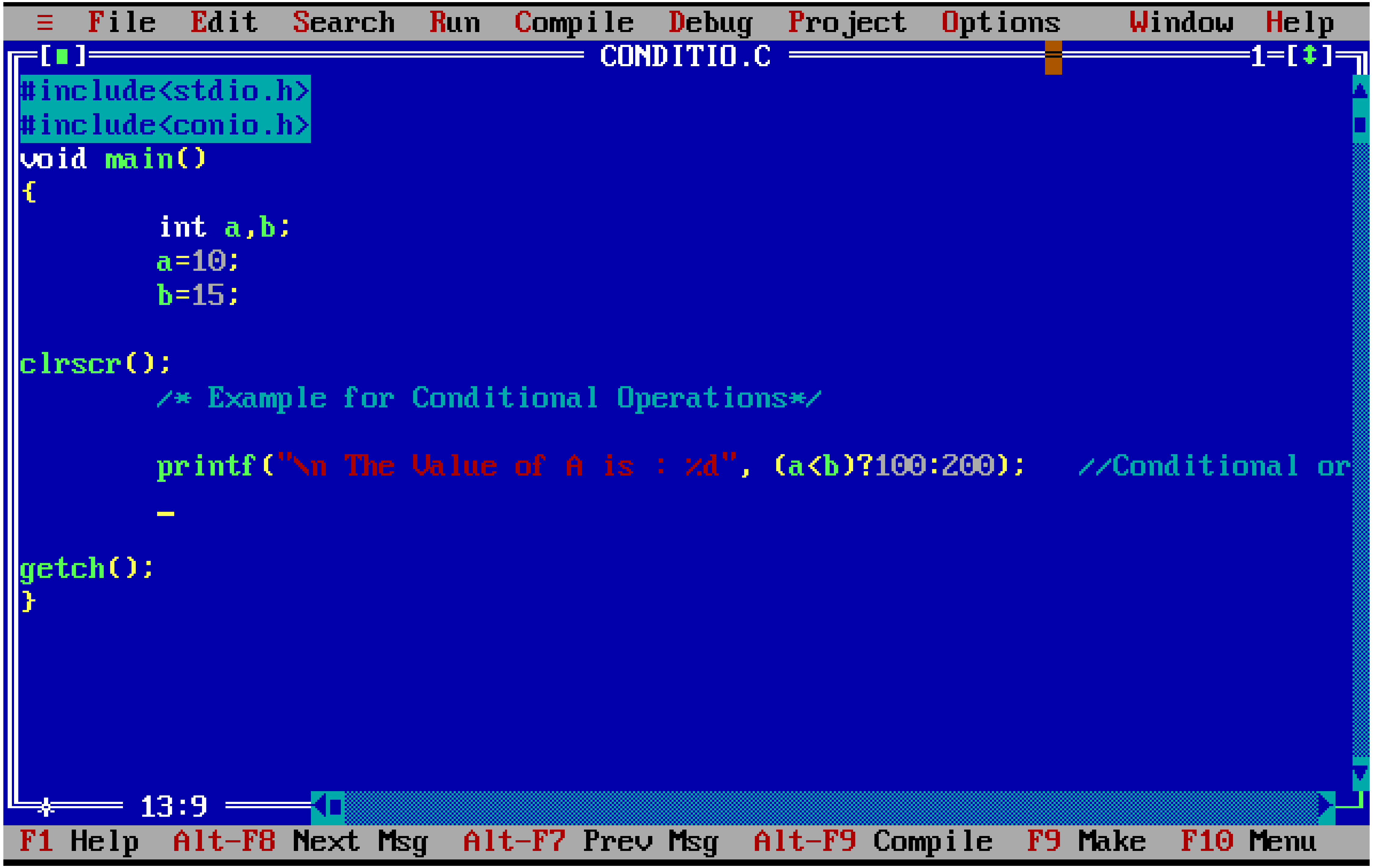
```
25&20: 16
25¦20: 29
25^20 : 13
~25 : -26
~20 : -21
a<<2 : 100
a>>2 : 6
```

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;

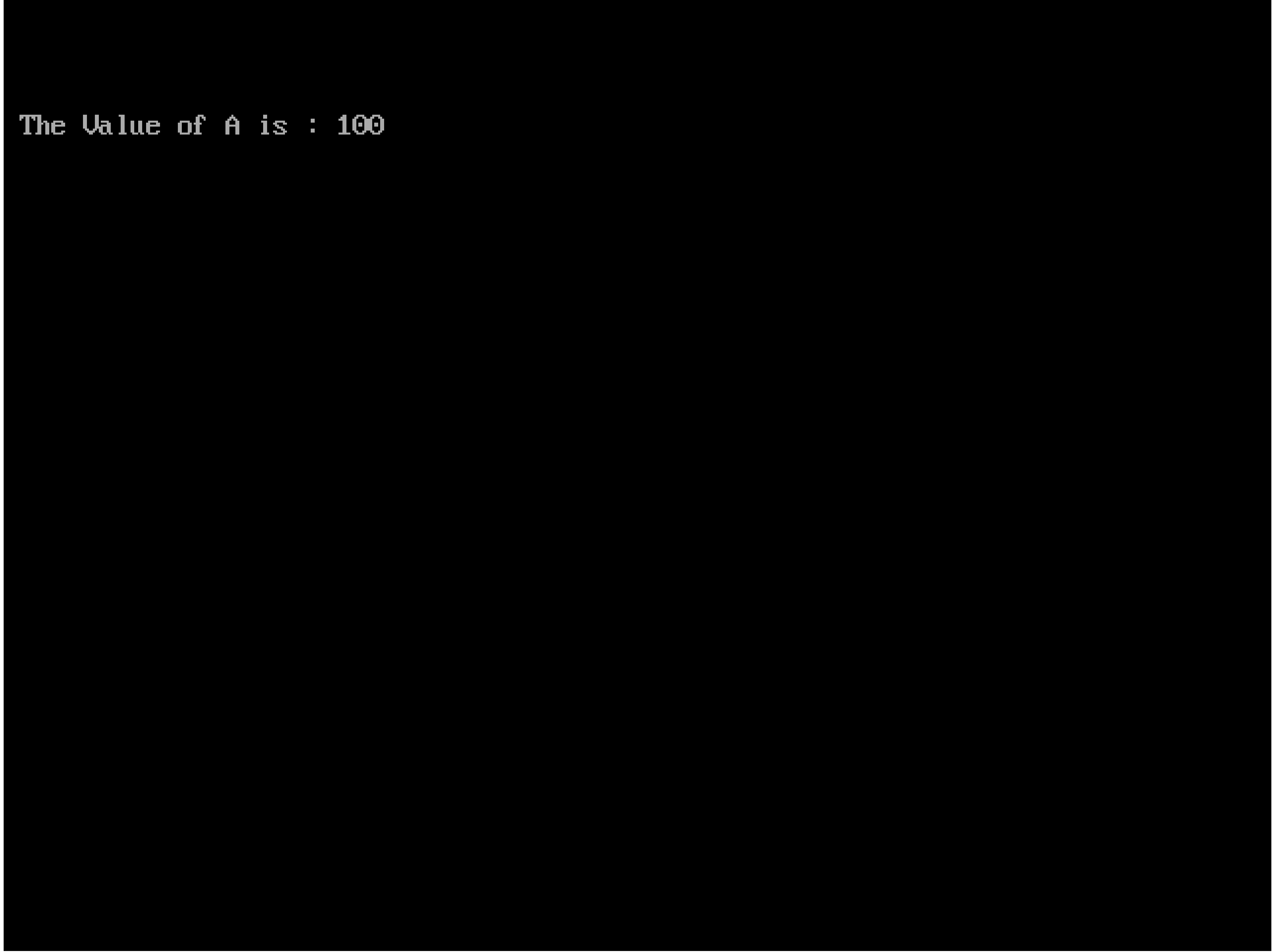
Example

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

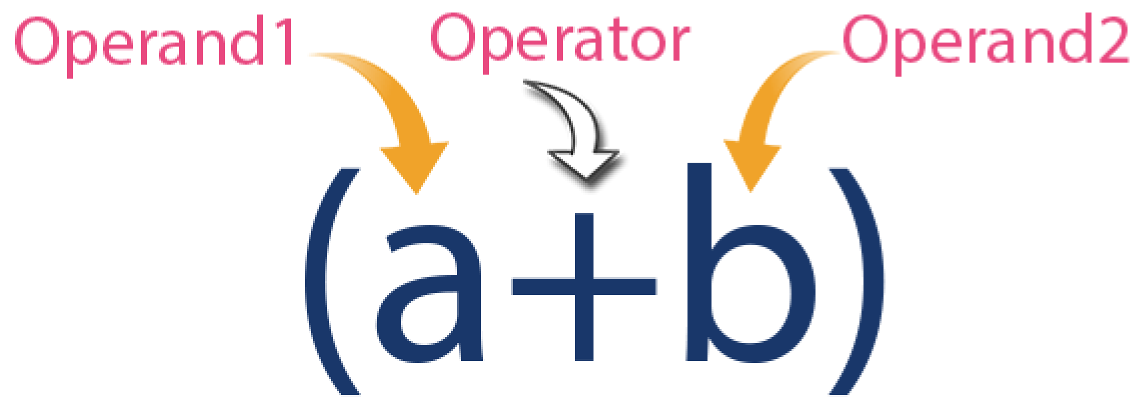
- In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.
- In the C programming language, an expression is defined as follows.
- **An expression is a collection of operators 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.

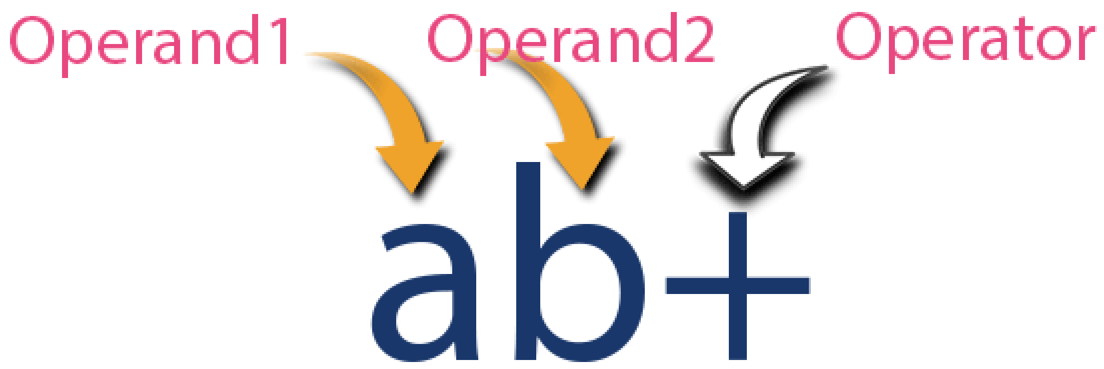
  Operand1 Operator Operand2

  Example

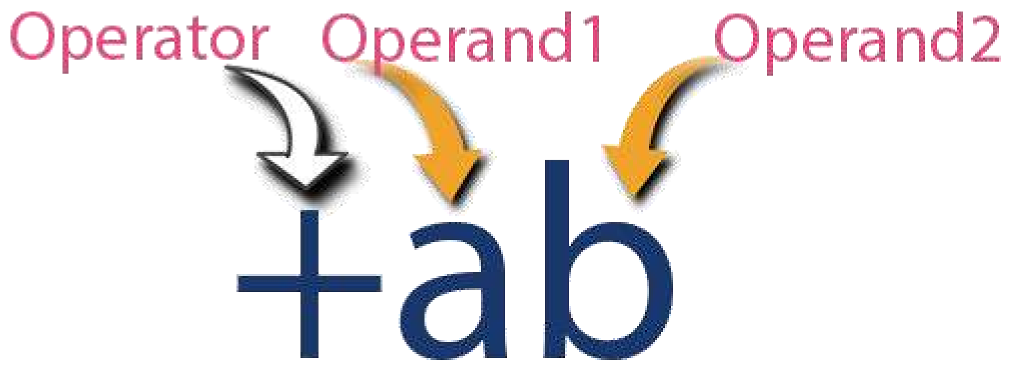


- **Postfix Expression**
- The expression in which the operator is used after operands is called postfix expression.
- The postfix expression has the following general structure.

  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

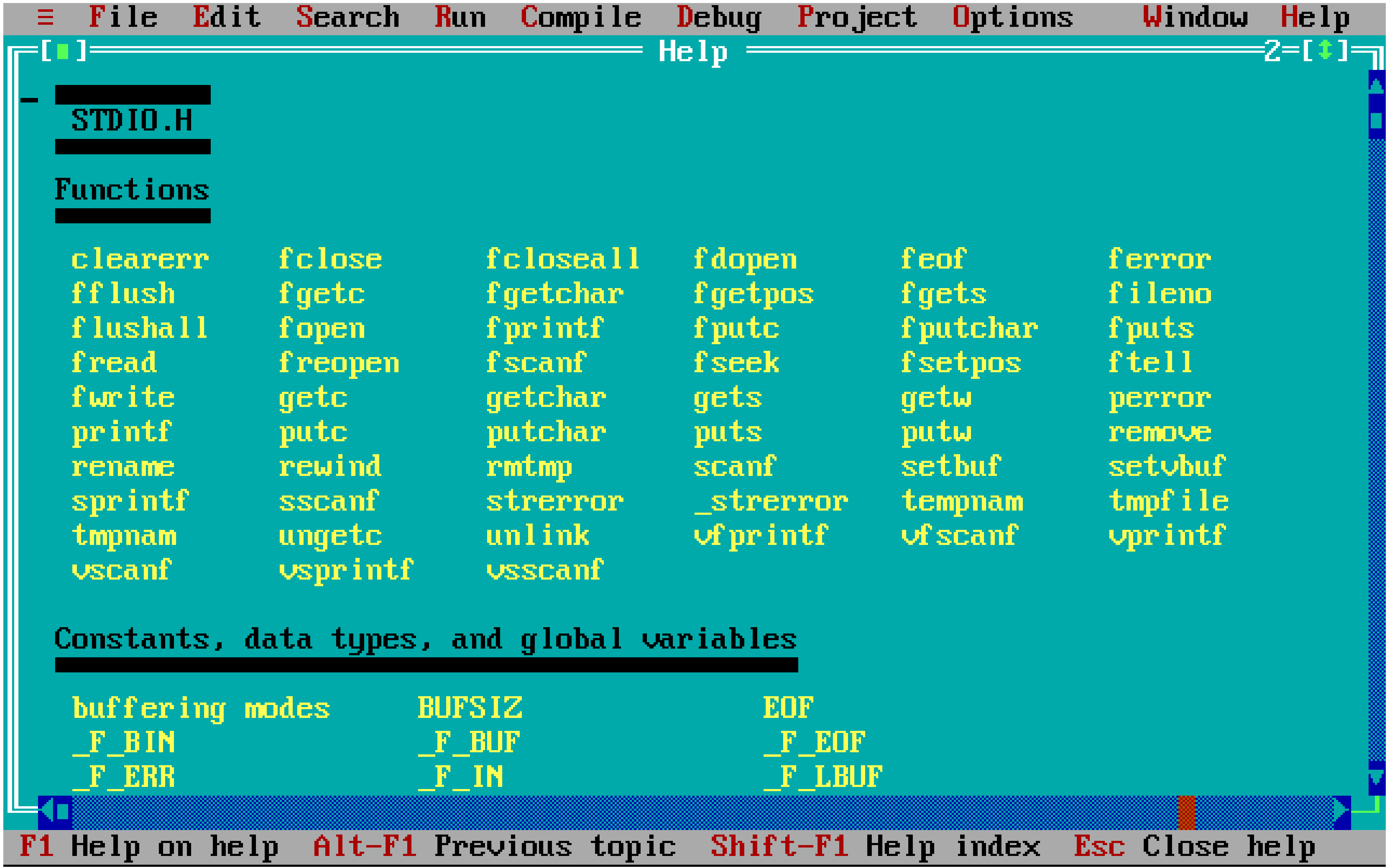
- Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least 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	


≡ 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

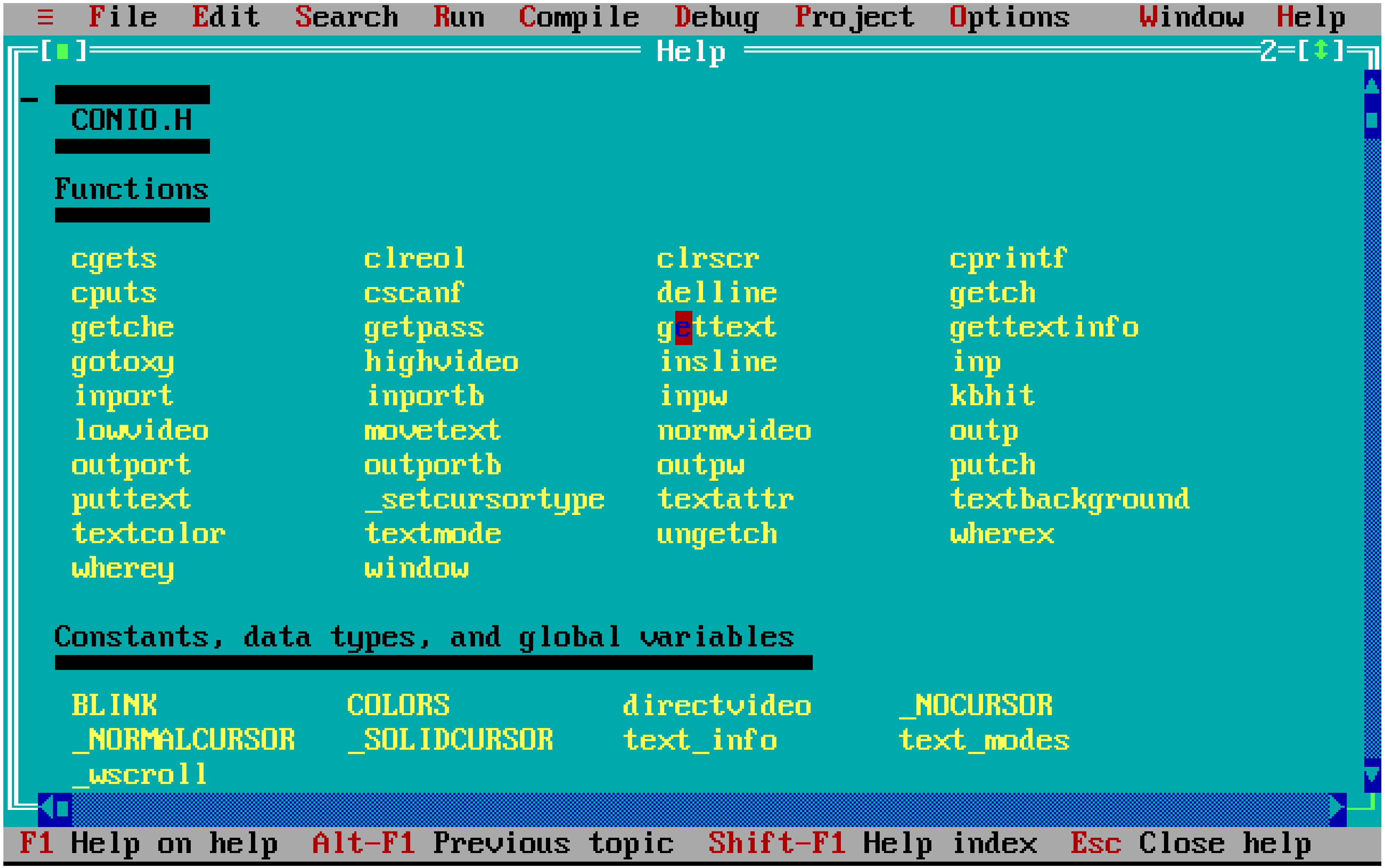
STDIO.H

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
```

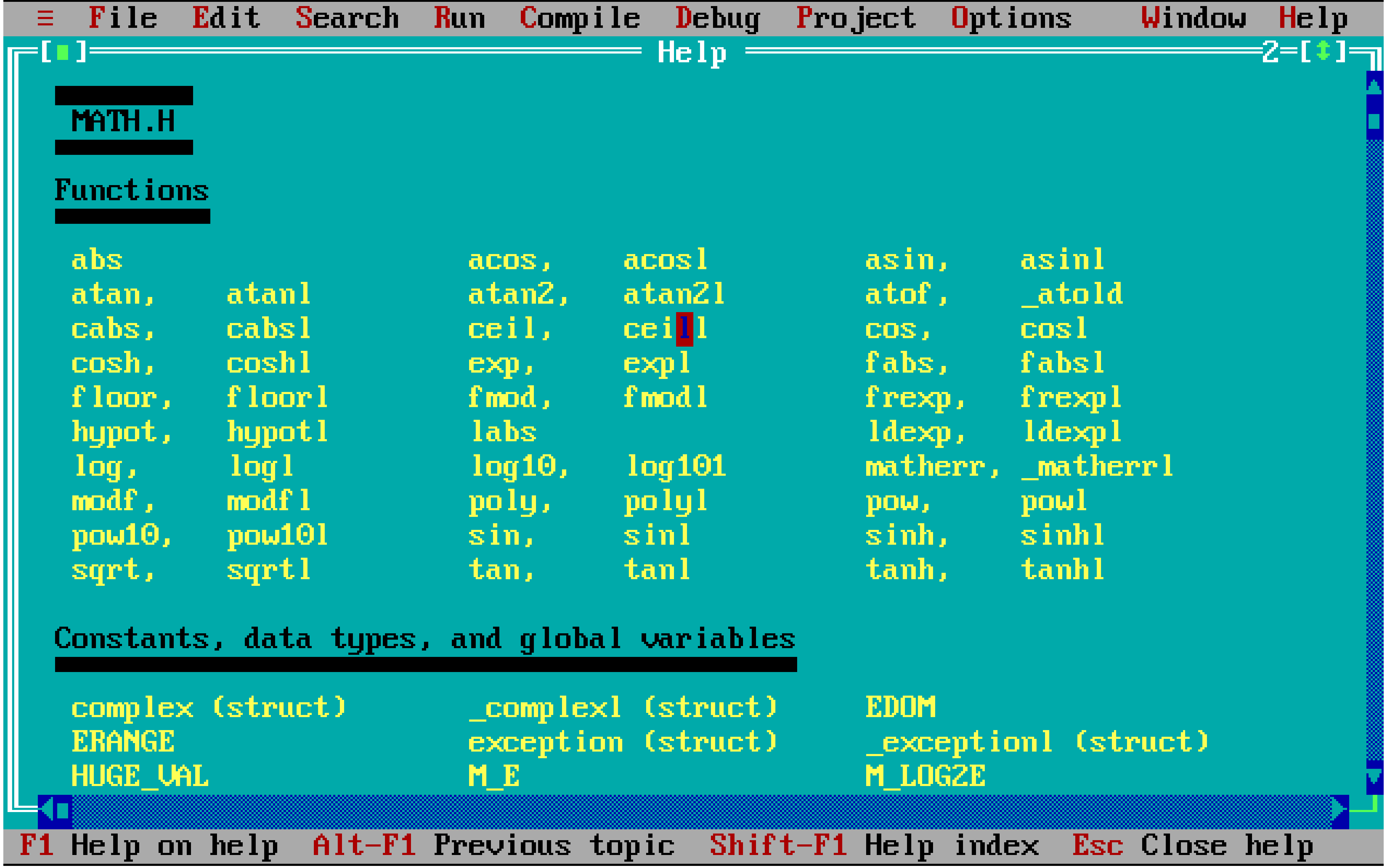
Constants, data types, and global variables

```
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=[↕]

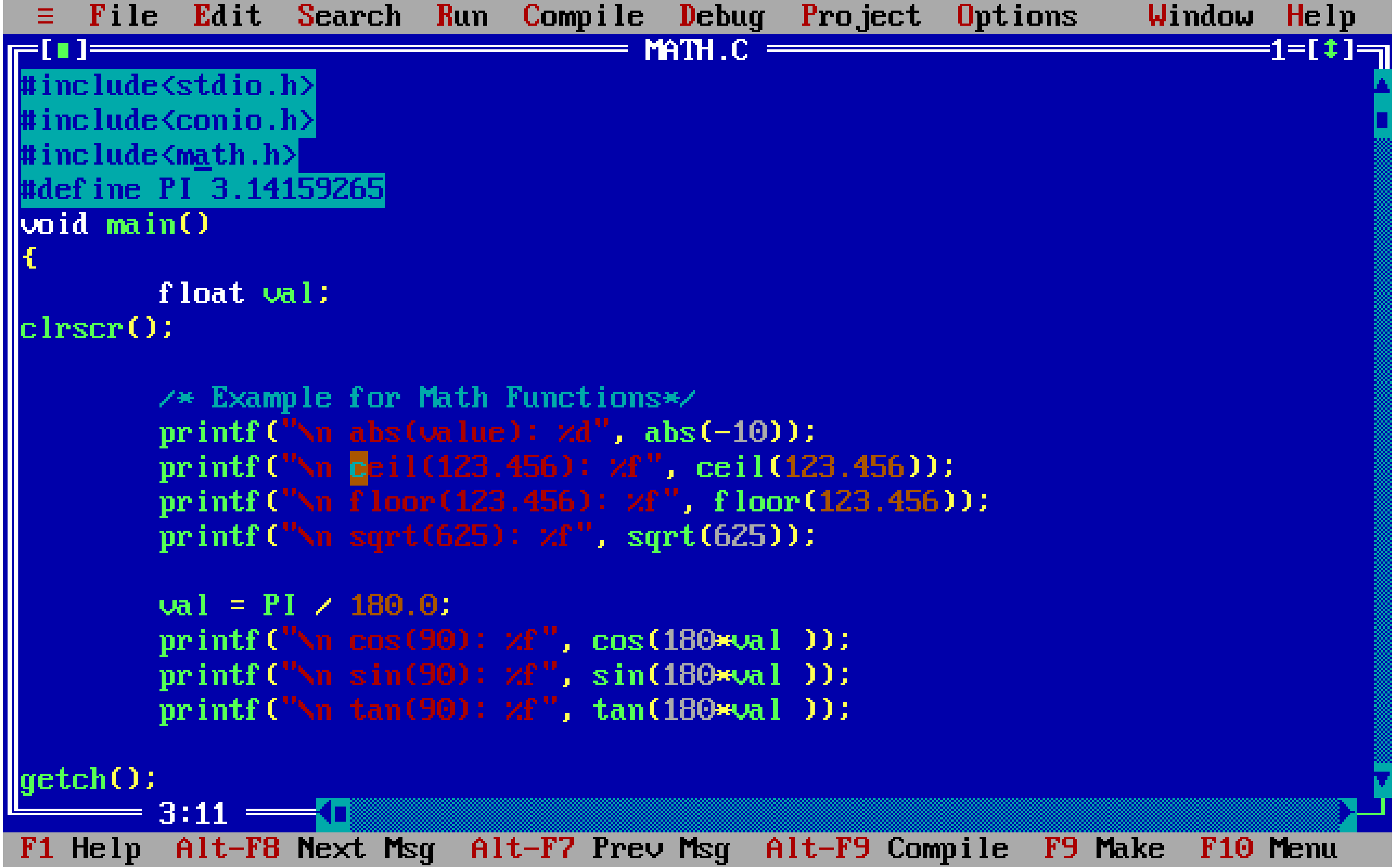
```
CONIO.H

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
```

F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help

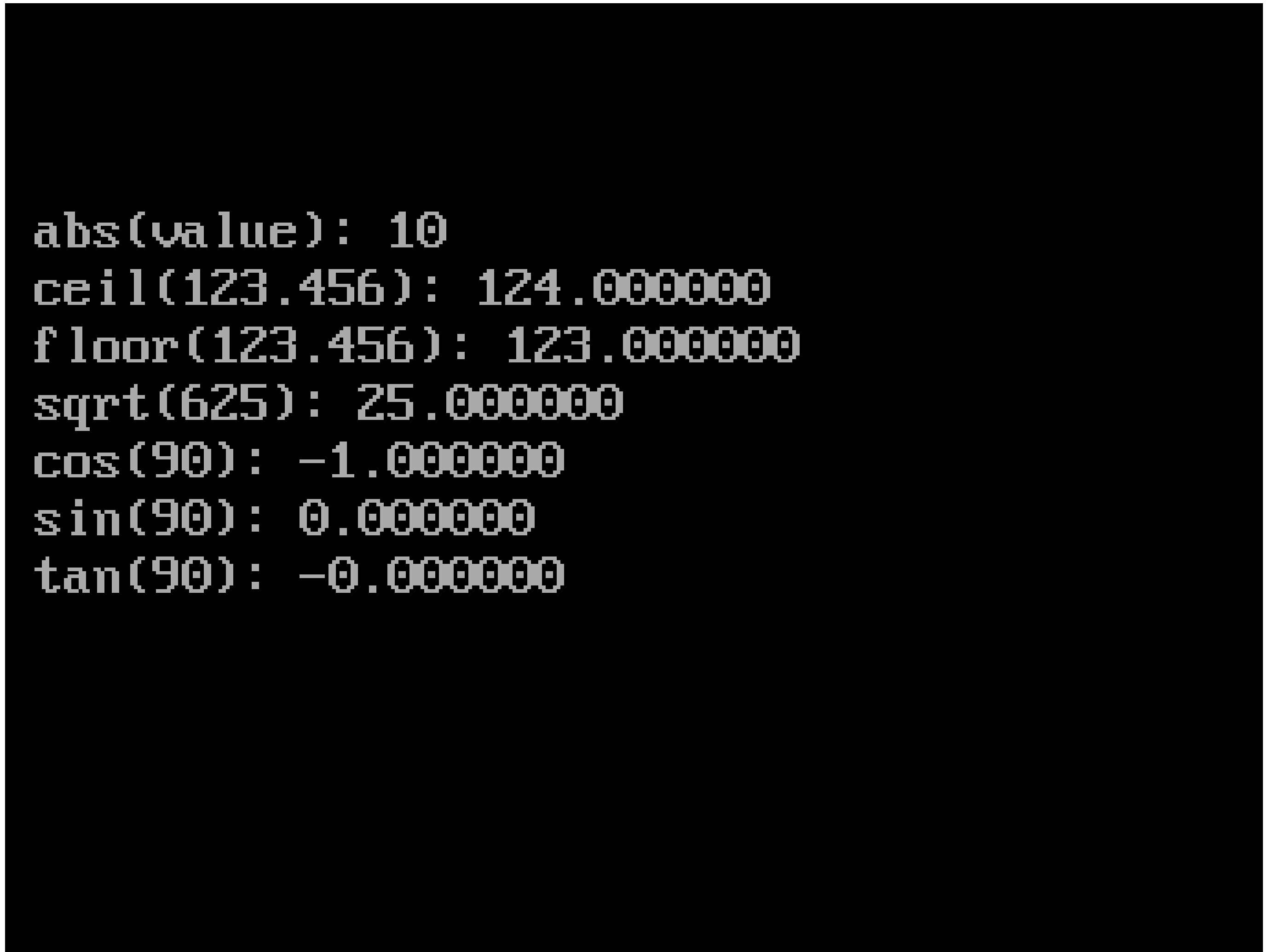

≡ File Edit Search Run Compile Debug Project Options Window Help
[■] Help 2=[↕]

```
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
```

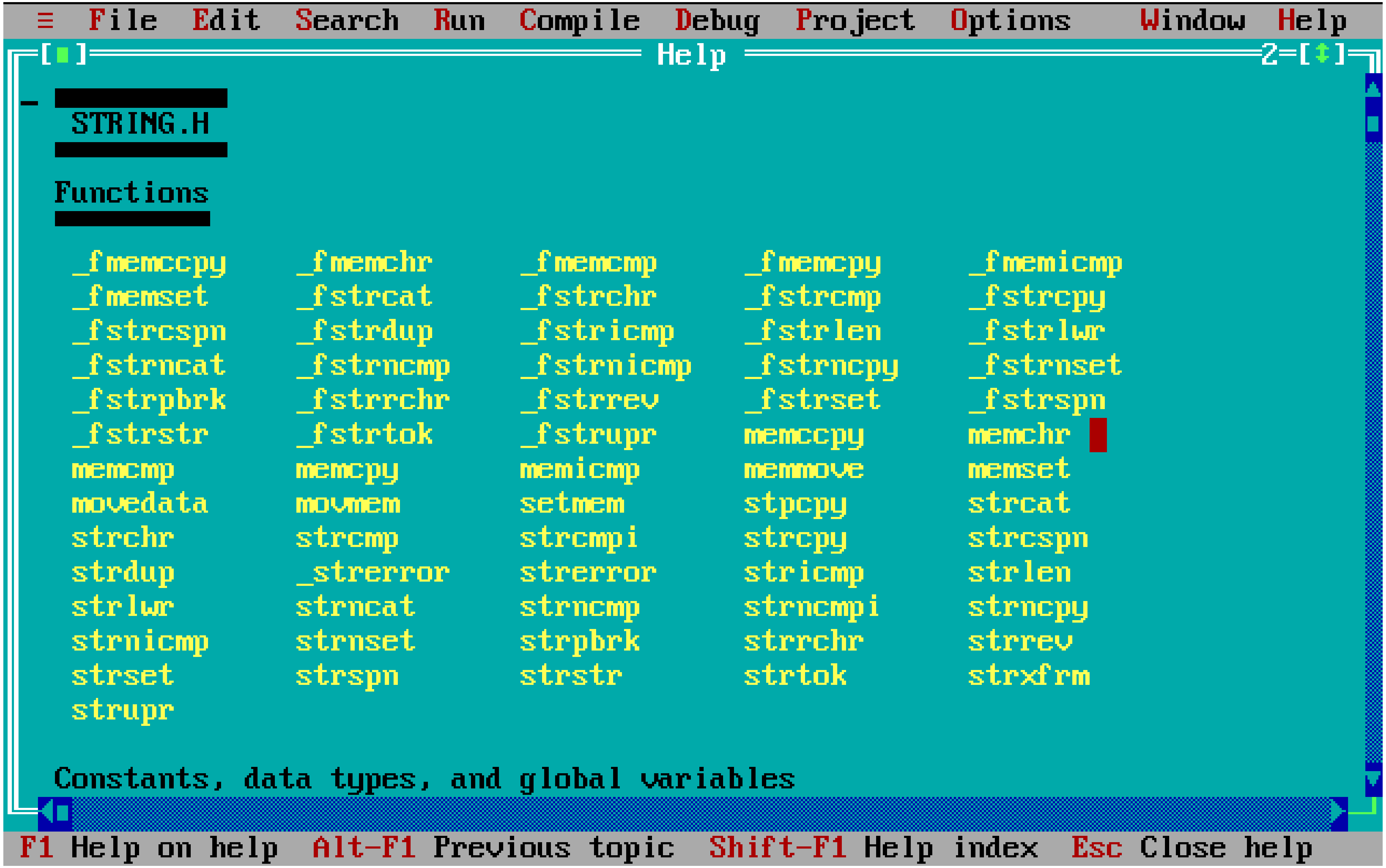
F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help


≡ 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

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

F1 Help on help Alt-F1 Previous topic Shift-F1 Help index Esc Close help

DATA INPUT AND OUTPUT FUNCTIONS

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

03-01-2022

- III Bsc Maths

Data Input and Output functions

- C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result. All these built-in functions are present in C header files.
- The c programming language provides the following basic built-in input functions.
  - getchar() and putchar()
  - getch() and putch()
  - getche()
  - getc() and putc()
  - gets() and puts()
  - scanf() and printf()
  - fscanf() and fprintf()

getchar()

- The getchar() function is used to read a character from the keyboard and return it to the program.
- This function is used to read a single character. To read multiple characters we need to write multiple times or use a looping statement.
- Syntax of getchar()

```
int getchar(void)
```

- The function does not have any parameters. However, it returns the read characters as an unsigned char in an int.

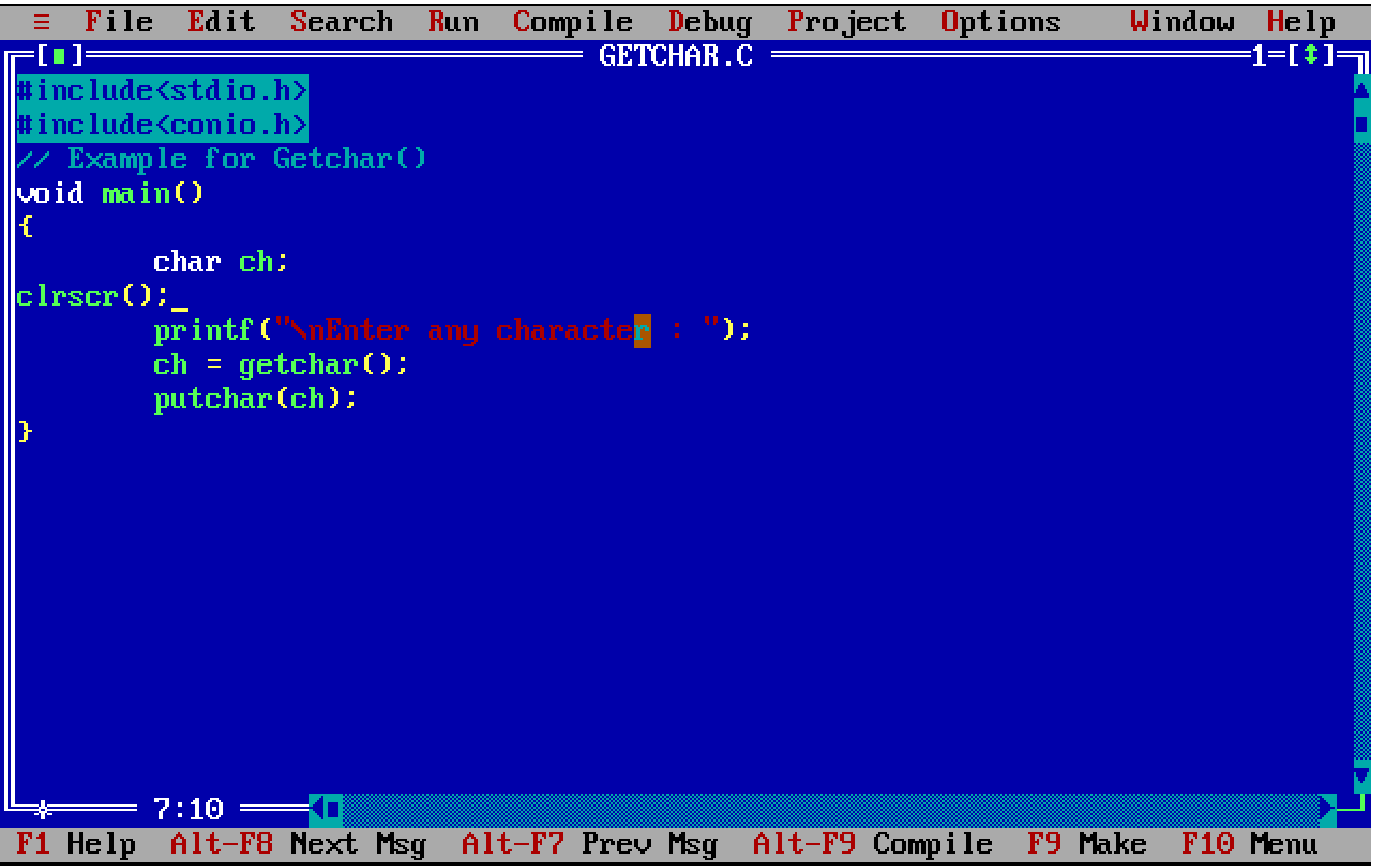
putchar()

- The putchar(int char) method in C is used to write a character, of unsigned char type, to stdout. This character is passed as the parameter to this method.

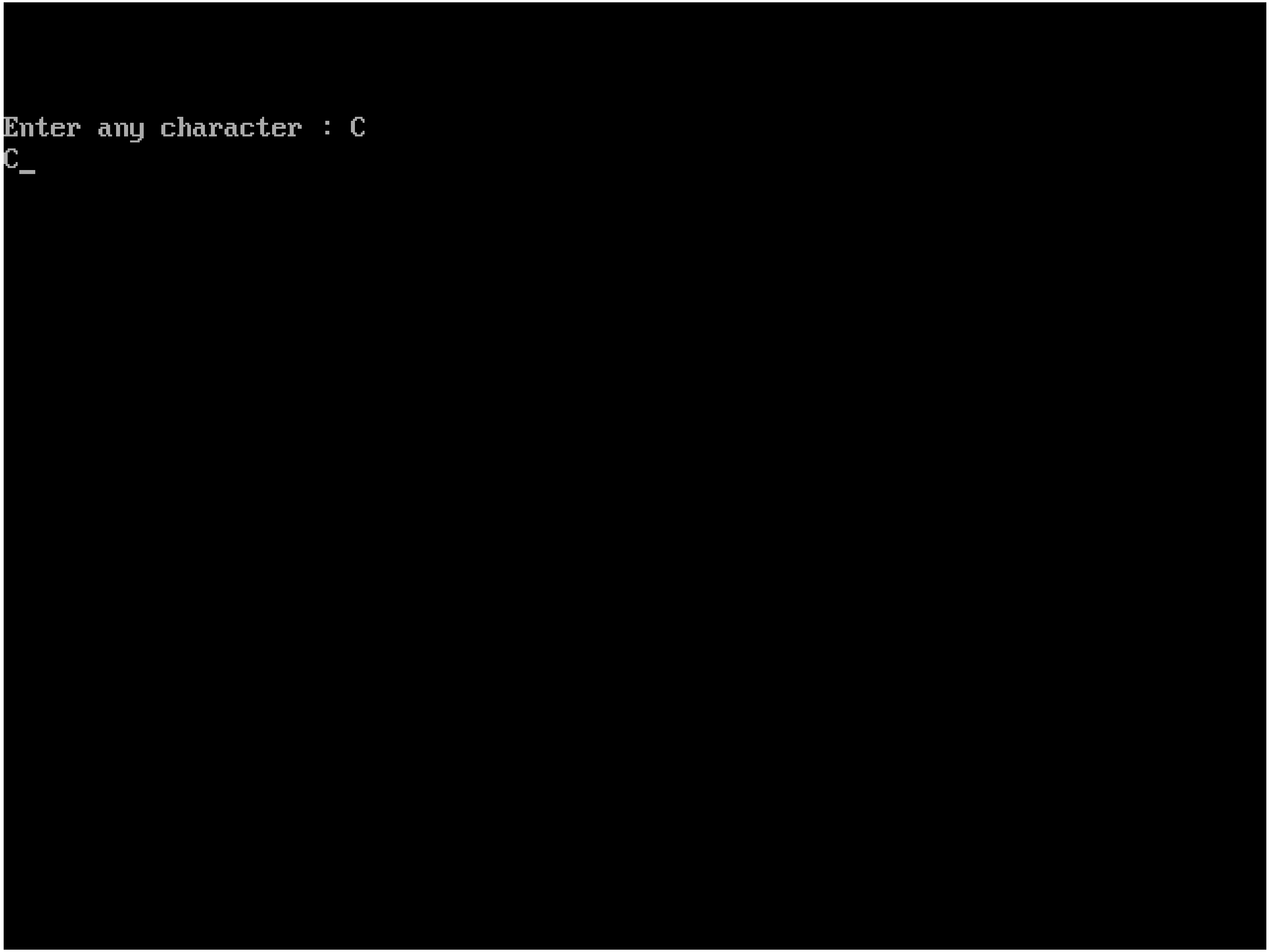
Syntax:

```
int putchar(int char)
```

- Parameters: This method accepts a mandatory parameter char which is the character to be written to stdout.
- Return Value: This function returns the character written on the stdout as an unsigned char. It also returns EOF when some error occurs.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] GETCHAR.C 1=[↕]
7:10
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
// Example for Getchar()
void main()
{
        char ch;
clrscr();_
        printf("\nEnter any character : ");
        ch = getchar();
        putchar(ch);
}
```



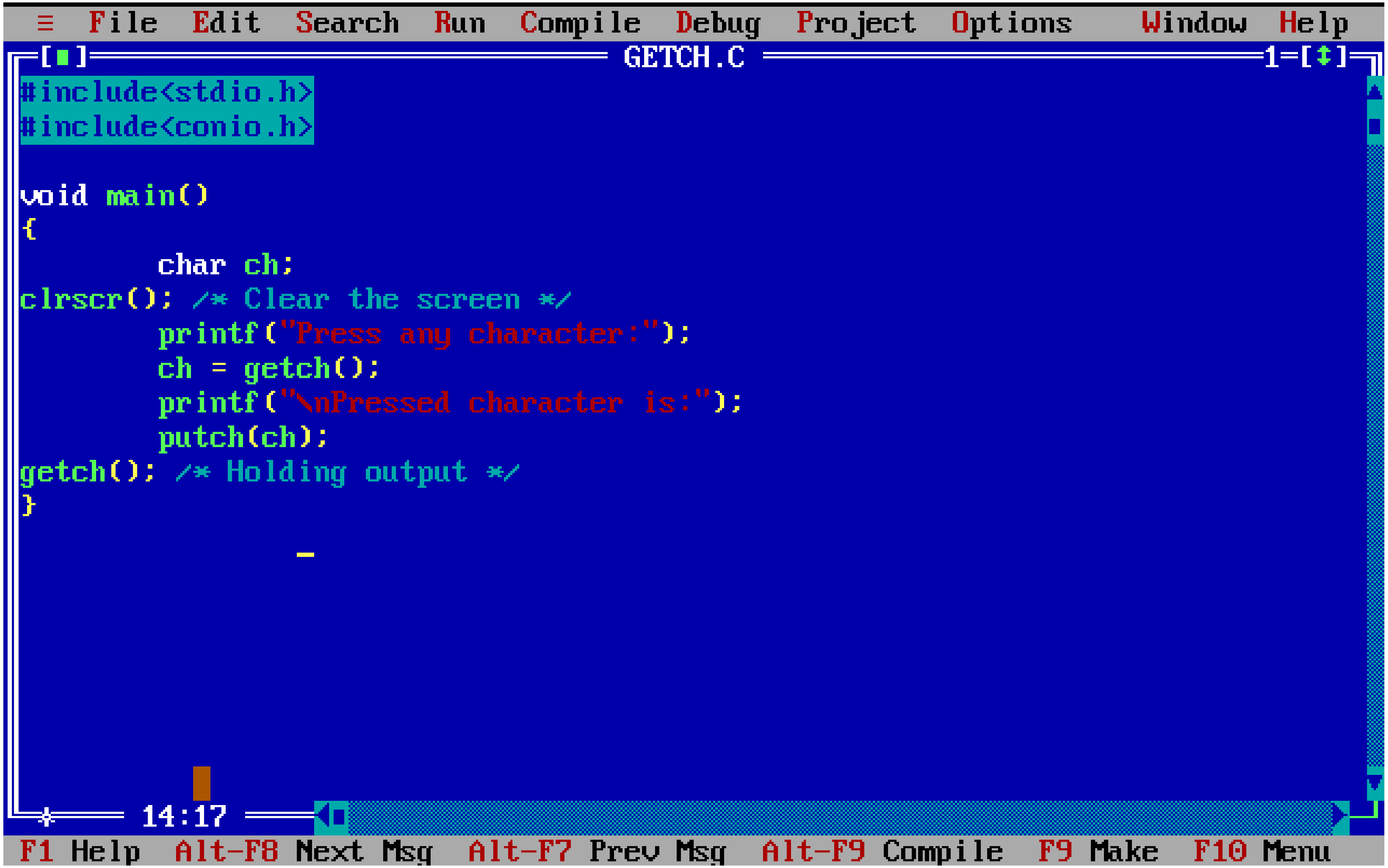
```
Enter any character : C
C_
```

getch() and putch()

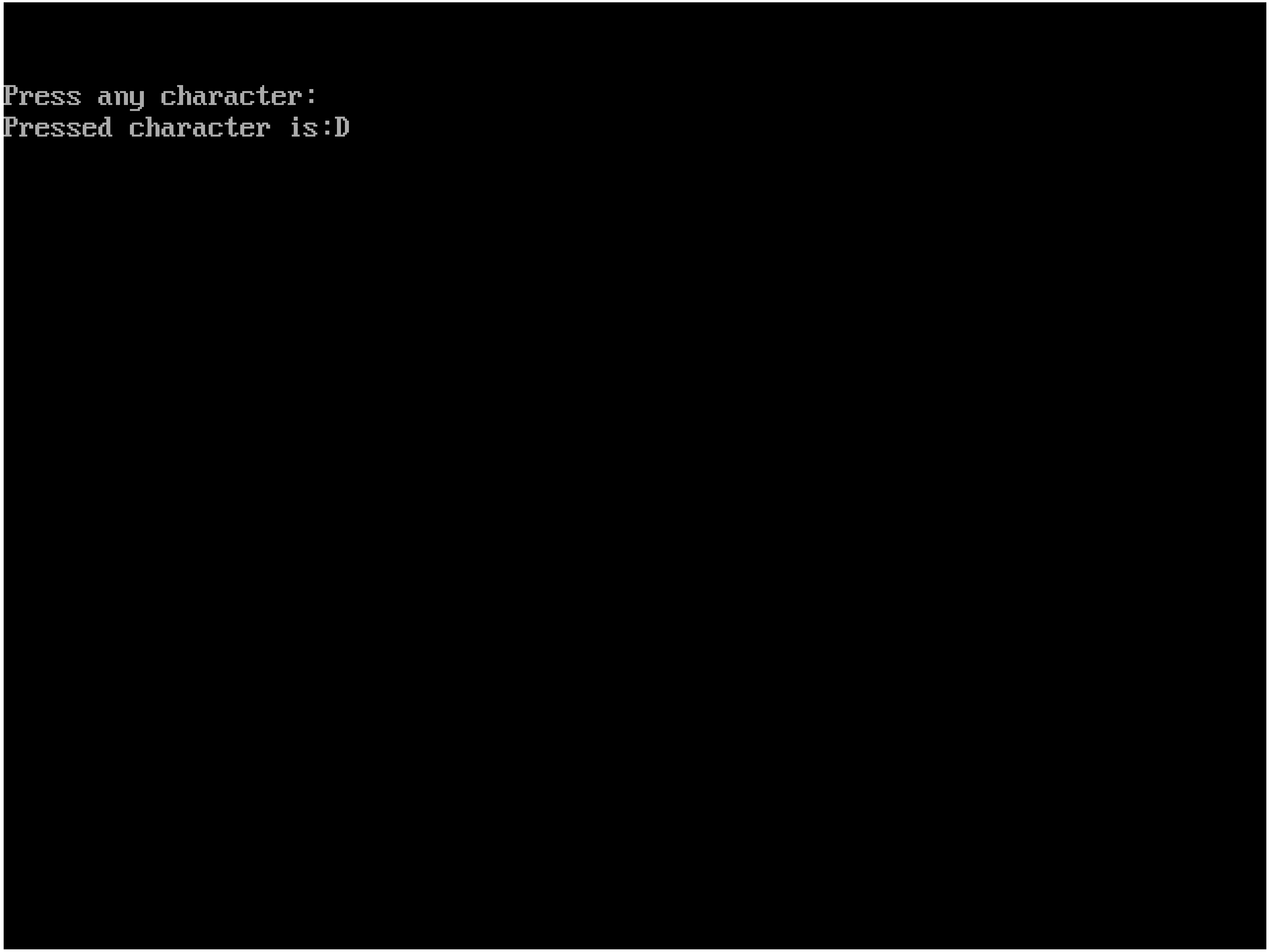
- getch() is a nonstandard function and is present in conio.h header file which is mostly used by MS-DOS compilers like Turbo C.
- Like these functions, getch() also reads a single character from the keyboard. But it does not use any buffer, so the entered character is immediately returned without waiting for the enter key.
- Syntax:

```
int getch(void);
```

- Parameters: This method does not accept any parameters.
- Return value: This method returns the ASCII value of the key pressed.
- The putch() function is used for printing character to a screen at current cursor location. It is unformatted character output functions. It is defined in header file conio.h.


≡ File Edit Search Run Compile Debug Project Options Window Help
GETCH.C
1
14:17
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()
{
        char ch;
clrscr(); /* Clear the screen */
        printf("Press any character:");
        ch = getch();
        printf("\nPressed character is:");
        putch(ch);
getch(); /* Holding output */
}
```

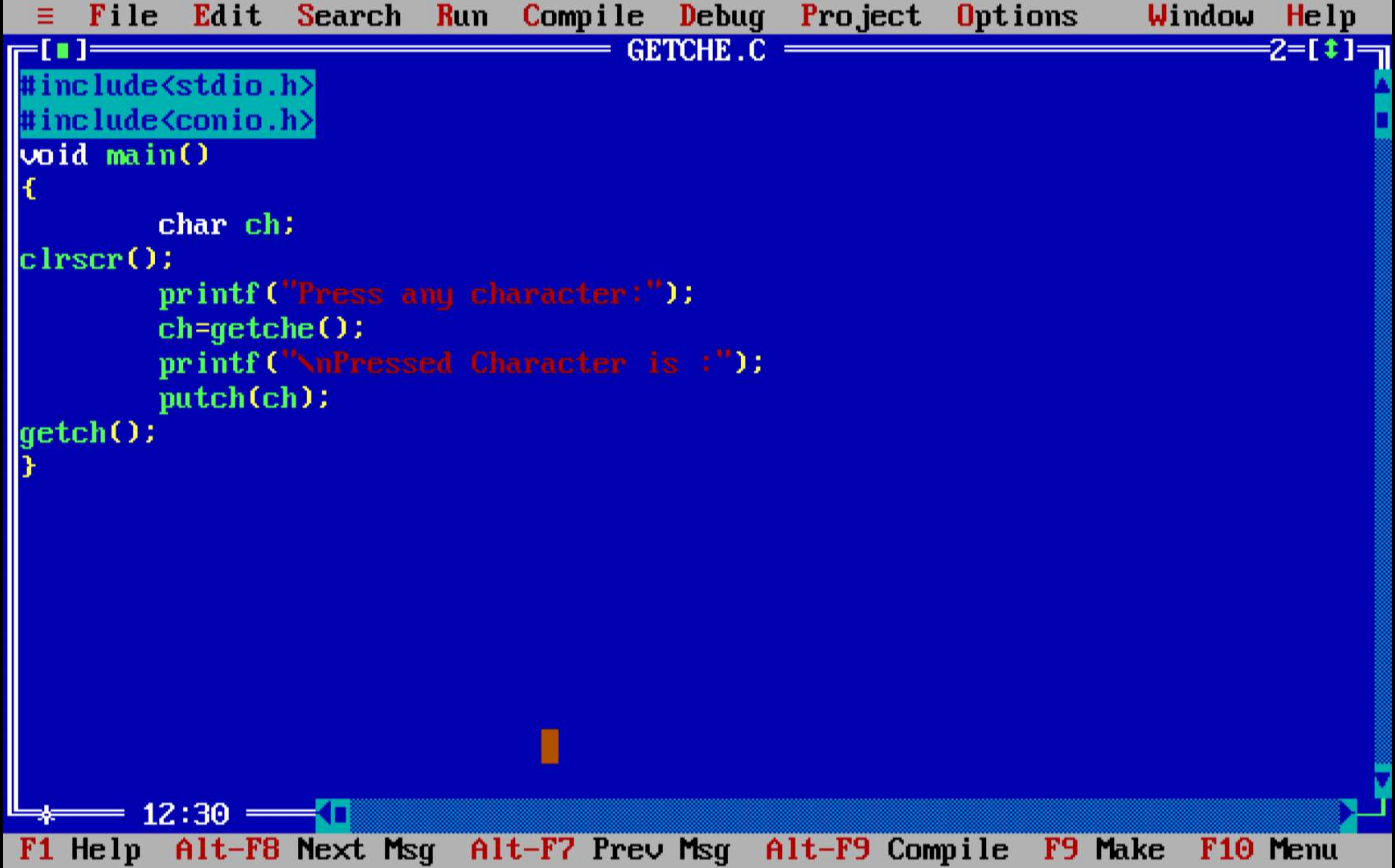


```
Press any character:
Pressed character is:D
```

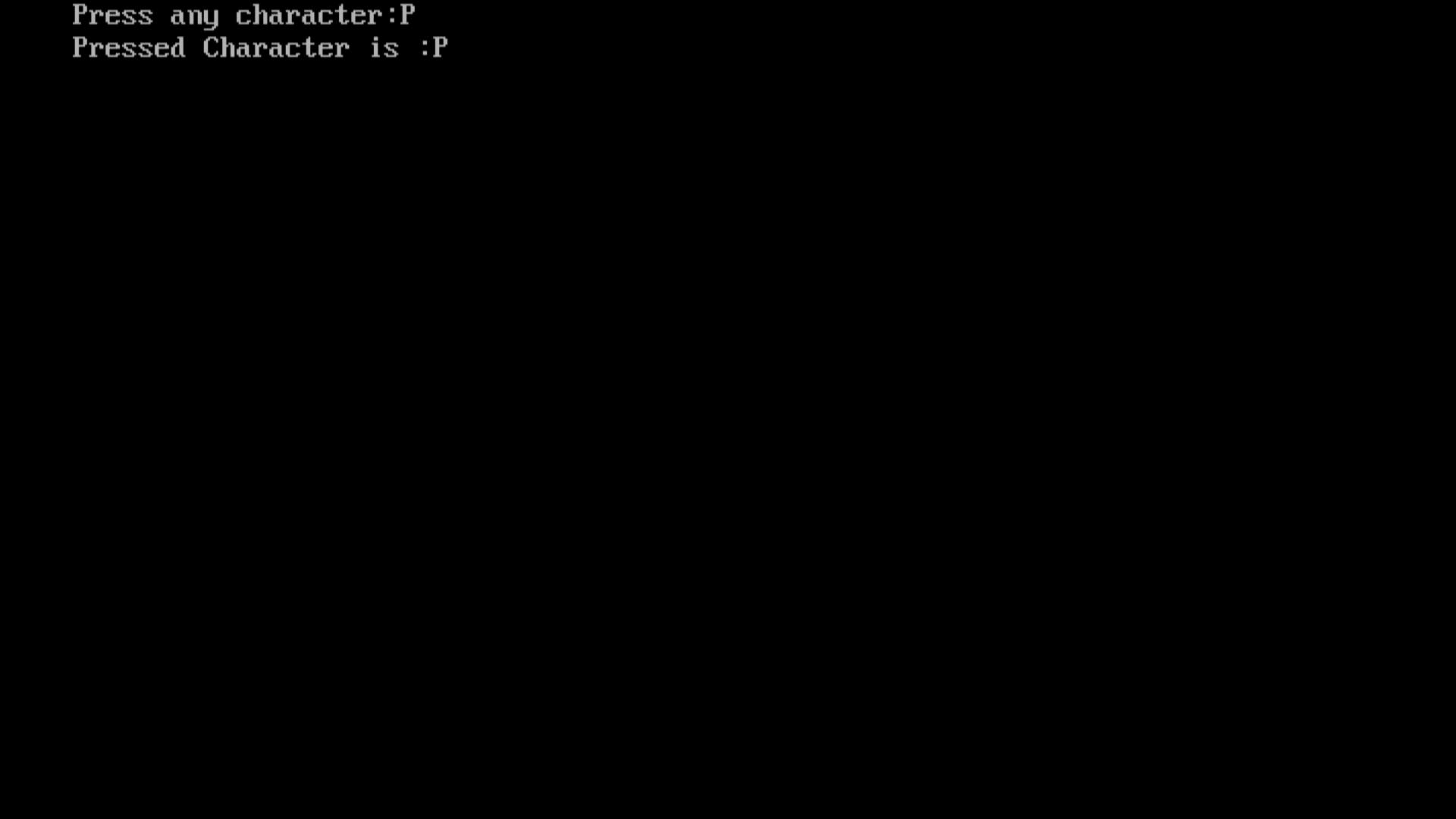
getche()

- Like getch(), getche() is also character input functions.
- It is unformatted input function meaning it does not allow user to read input in their format.
- Difference between getch() and getche() is that getche() echoes pressed character.
- getche() also returns character pressed like getch(). It is also defined in header file conio.h.

```
int getche(void);
```


≡ File Edit Search Run Compile Debug Project Options Window Help
GETCHE.C
2
12:30
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()
{
        char ch;
clrscr();
        printf("Press any character:");
        ch=getche();
        printf("\nPressed Character is :");
        putch(ch);
getch();
}
```



```
Press any character:P
Pressed Character is :P
```

getc and putc()

- int getc(FILE *stream) gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream.
- Syntax getc() function.

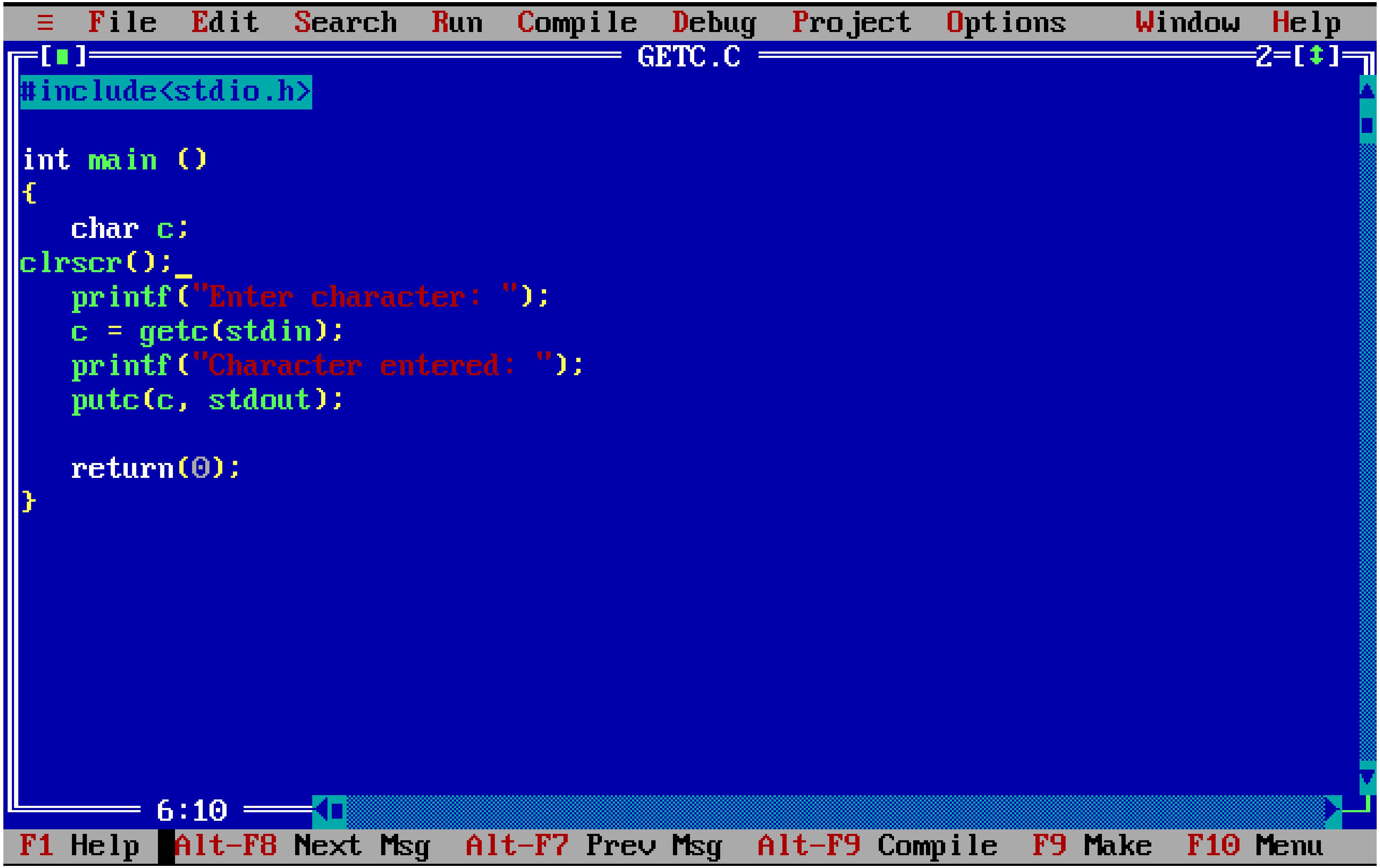
```
int getc(FILE *stream)
```

- Parameters

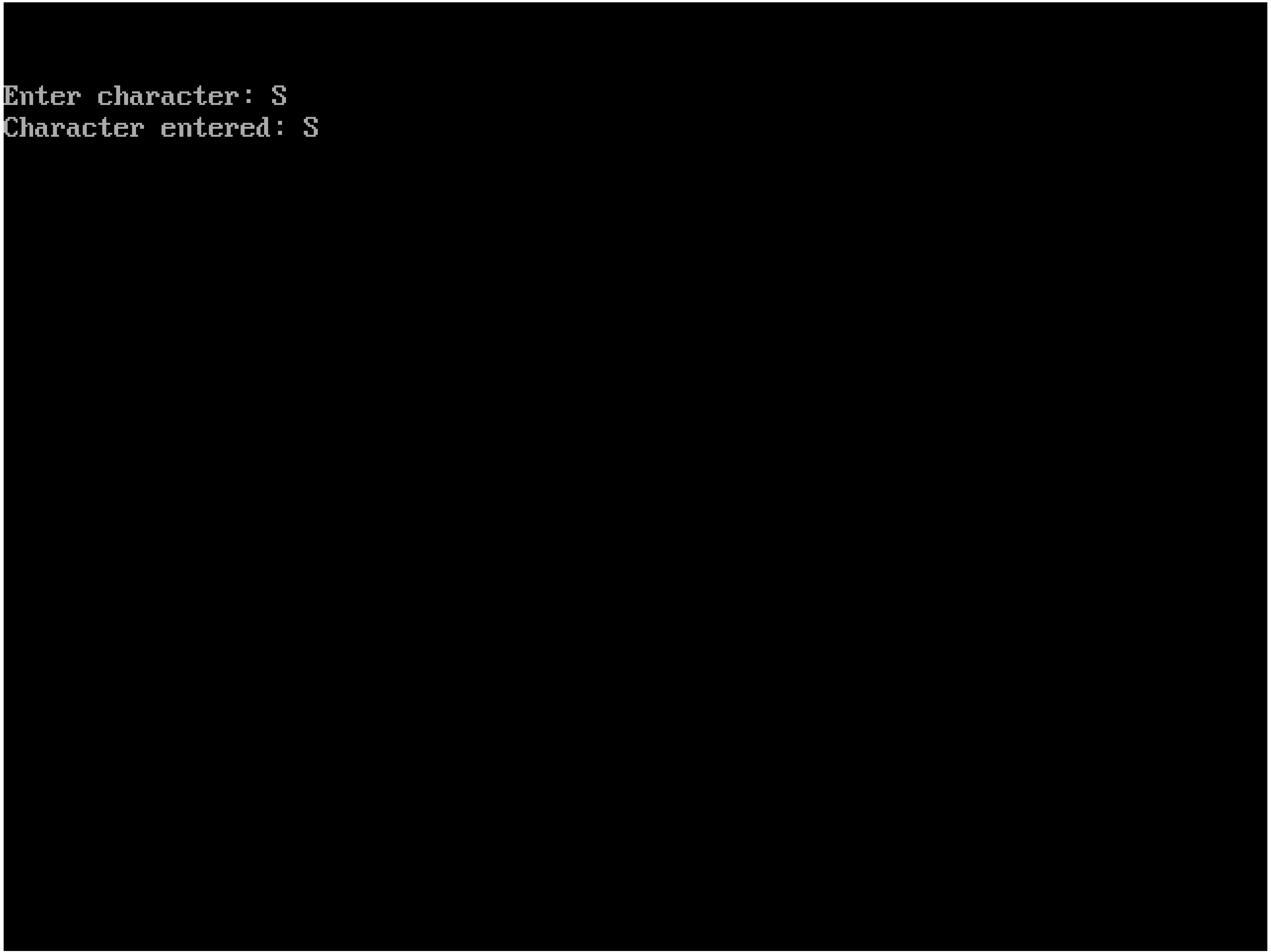
  stream − This is the pointer to a FILE object that identifies the stream on which the operation is to be performed.

- Return Value

  This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] GETC.C 2=[↕]
6:10
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>

int main ()
{
   char c;
clrscr();_
   printf("Enter character: ");
   c = getc(stdin);
   printf("Character entered: ");
   putc(c, stdout);

   return(0);
}
```



```
Enter character: S
Character entered: S
```

gets() and puts()

- The C library function char *gets(char *str) reads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.
- Syntax of gets() function.

```
char *gets(char *str)
```

- Parameters

  str − This is the pointer to an array of chars where the C string is stored.

- Return Value

  This function returns str on success, and NULL on error or when end of file occurs, while no characters have been read.

- The C library function int puts(const char *str) writes a string to stdout up to but not including the null character. A newline character is appended to the output.
- Syntax of puts() function.

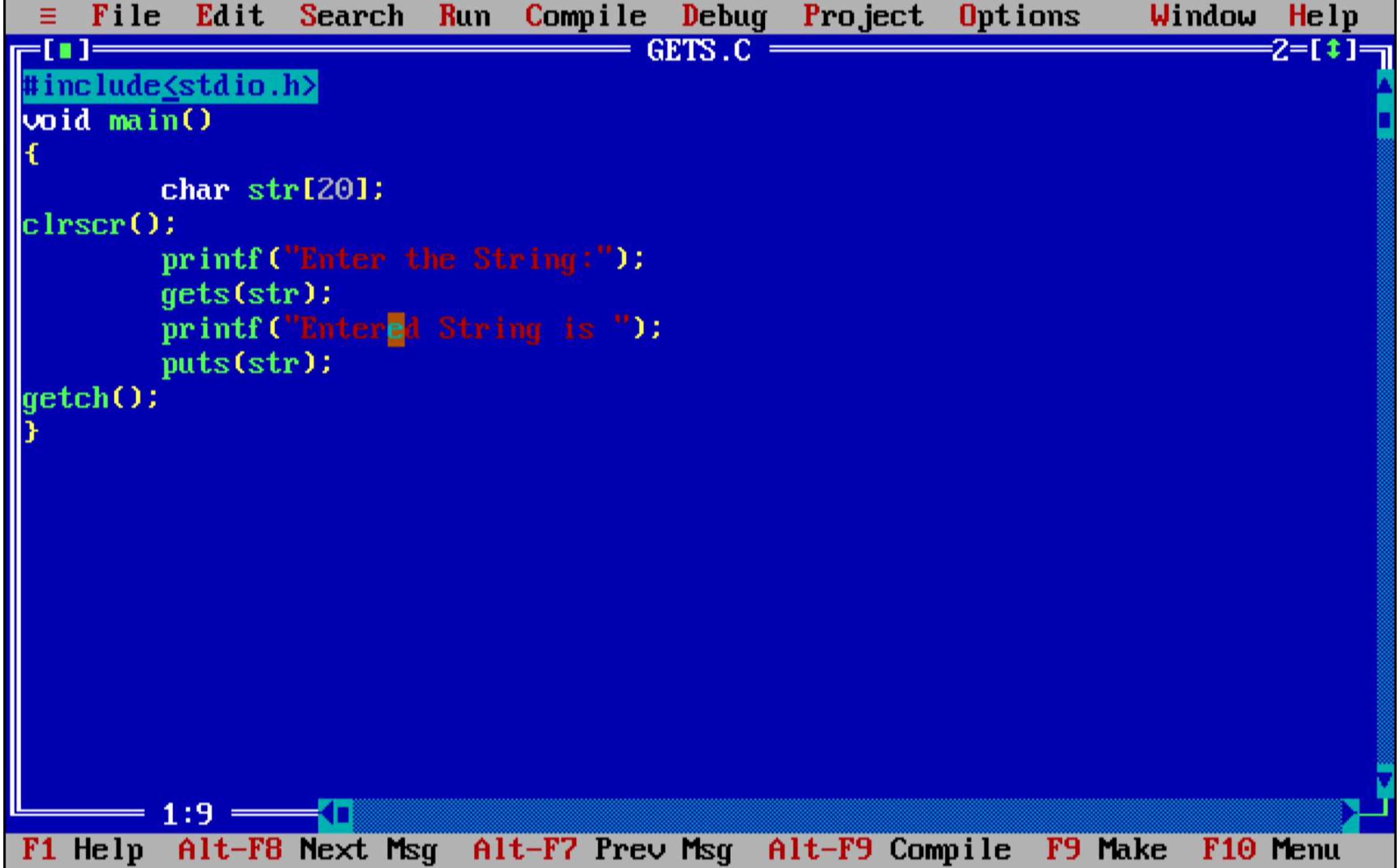
```
int puts(const char *str)
```

Parameters

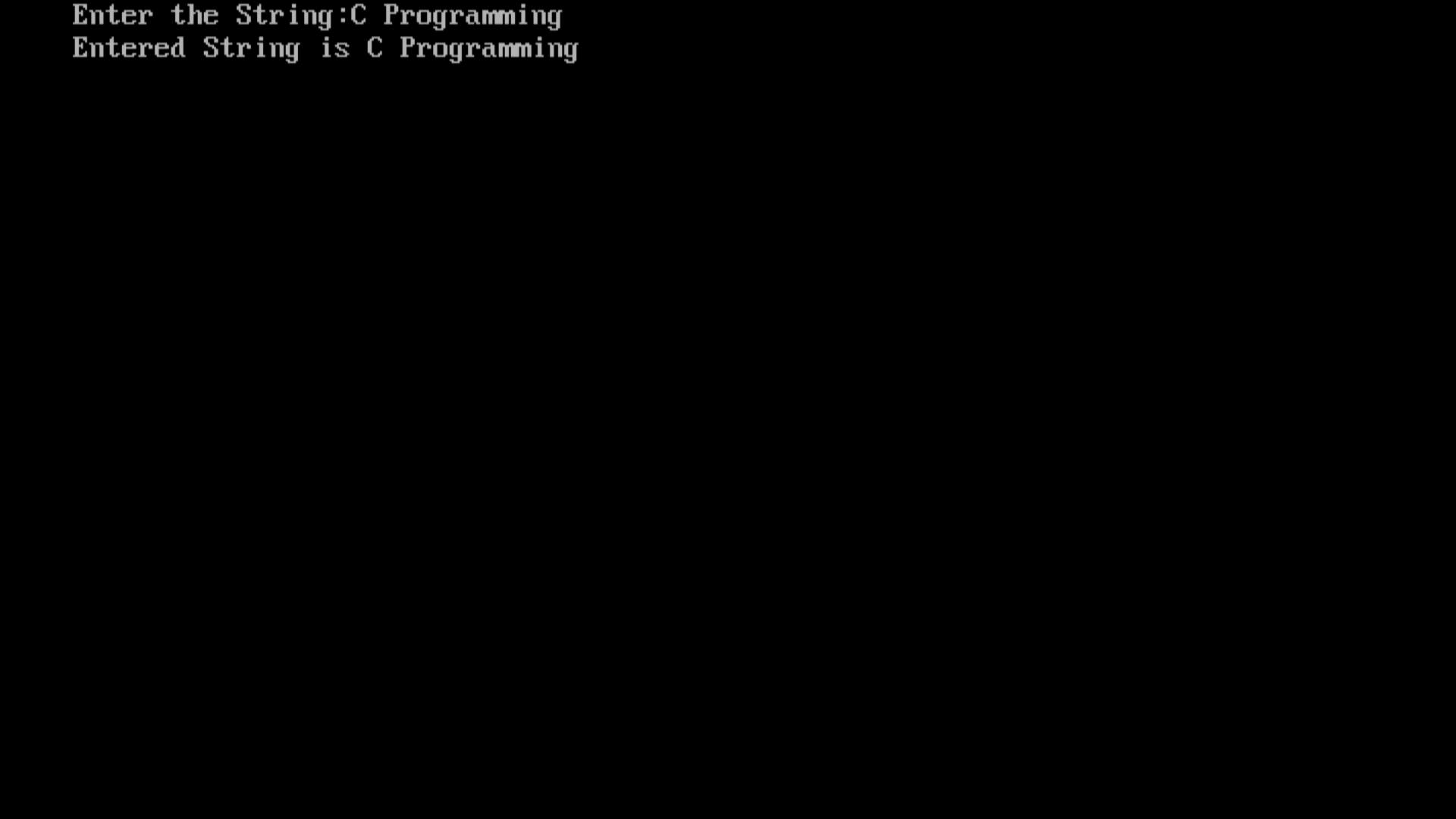
str − This is the C string to be written.

Return Value

If successful, non-negative value is returned. On error, the function returns EOF.


≡ File Edit Search Run Compile Debug Project Options Window Help
GETS.C
2
1:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
void main()
{
        char str[20];
clrscr();
        printf("Enter the String:");
        gets(str);
        printf("Entered String is ");
        puts(str);
getch();
}
```



```
Enter the String:C Programming
Entered String is C Programming
```

scanf()

- The scanf() stands for Scan formatting and is used to read formatted data from keyboard.
- The scanf() function is used to read multiple data values of different data types from the keyboard.
- The scanf() function is built-in function defined in a header file called "stdio.h".
- When we want to use scanf() function in our program, we need to include the respective header file (stdio.h) using #include statement.
- The scanf() function has the following syntax...

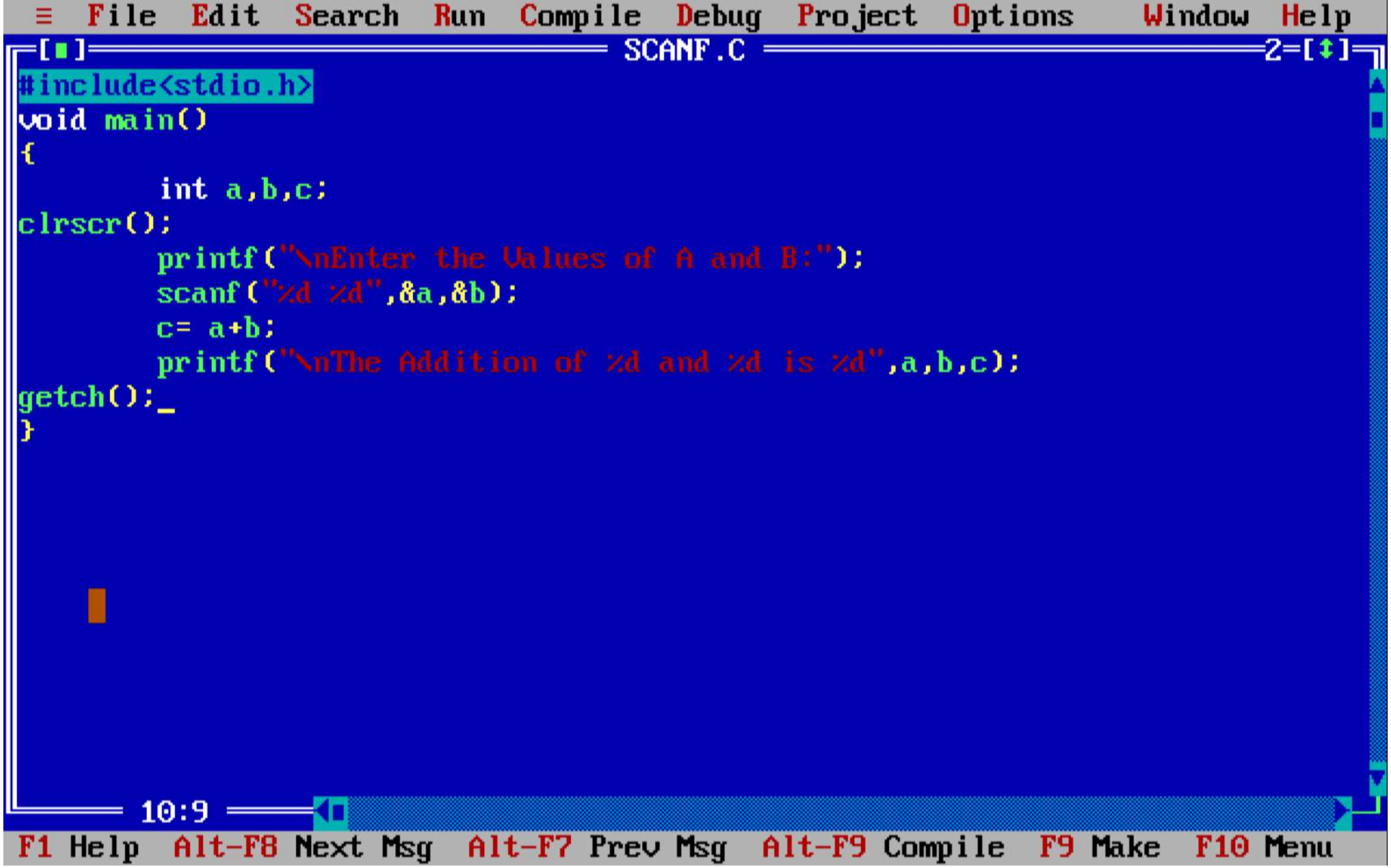
Syntax:

```
scanf("format strings",&variableNames);
```

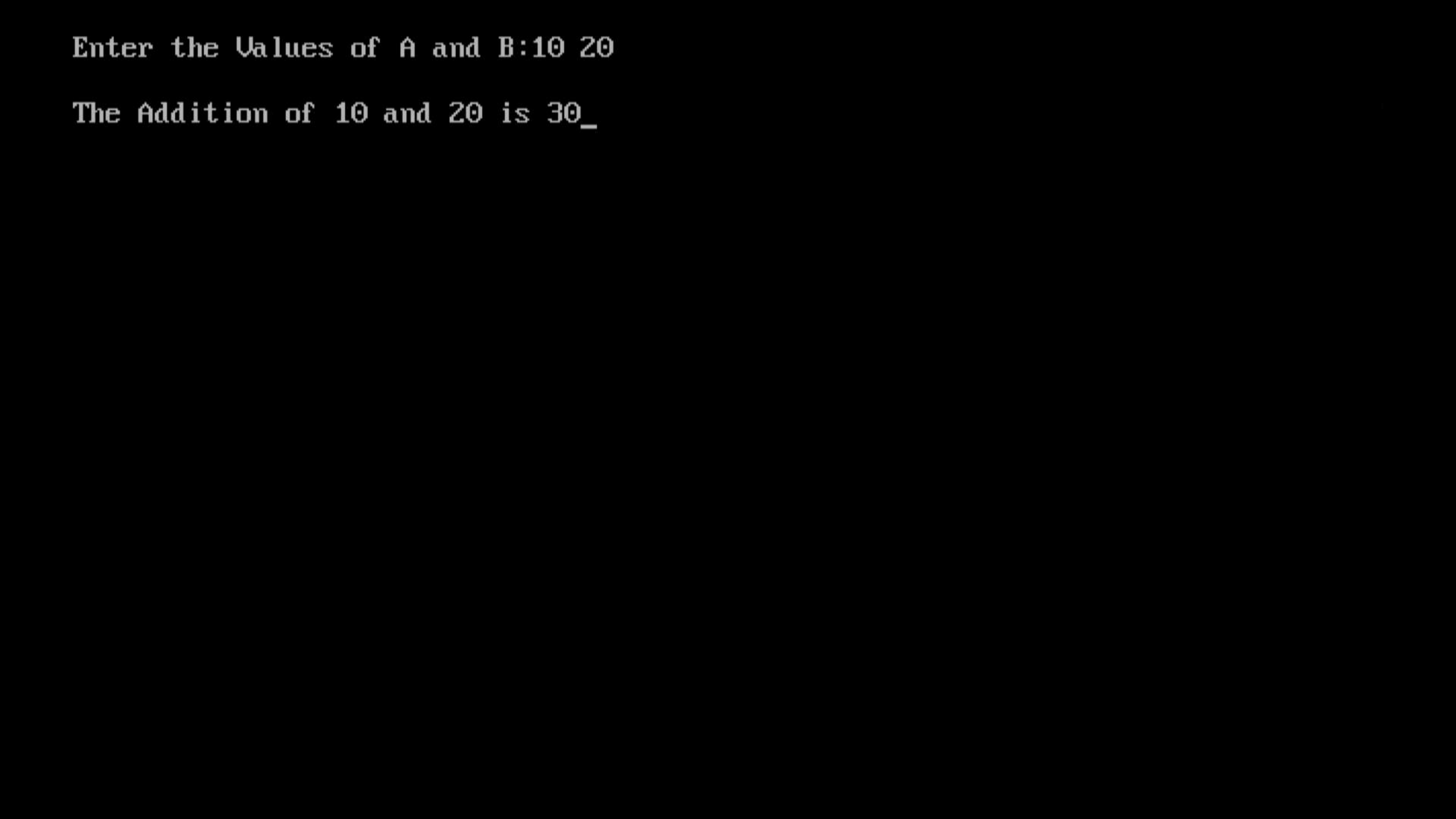
- The format specifiers are used in C for input and output purposes. Using this concept the compiler can understand that what type of data is in a variable during taking input using the scanf() function and printing using printf() function. Here is a list of format specifiers.

Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)

Format Specifier	Type
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character


≡ File Edit Search Run Compile Debug Project Options Window Help
SCANF.C
10:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
void main()
{
        int a,b,c;
clrscr();
        printf("\nEnter the Values of A and B:");
        scanf("%d %d",&a,&b);
        c= a+b;
        printf("\nThe Addition of %d and %d is %d",a,b,c);
getch();_
}
```



```
Enter the Values of A and B:10 20

The Addition of 10 and 20 is 30_
```

printf()

- printf() stands for print formatting and is used to display information required by the user and also prints the values of the variables.
- Syntax:

  printf(“format_string”, var1, var2, var3, …, varN);
- Where format_string may contain :
  - Characters that are simply printed as they are.
  - Format specifier that begin with a % sign.
  - Escape sequences that begin with \ sign.
- The format string indicates how many arguments follow and what their types are. The arguments var1, var2, …, varN are the variables whose values are formatted and printed according to format specifications of the format string. The arguments must match in number, order and type with the format specifications.

CONTROL STATEMENTS IN C

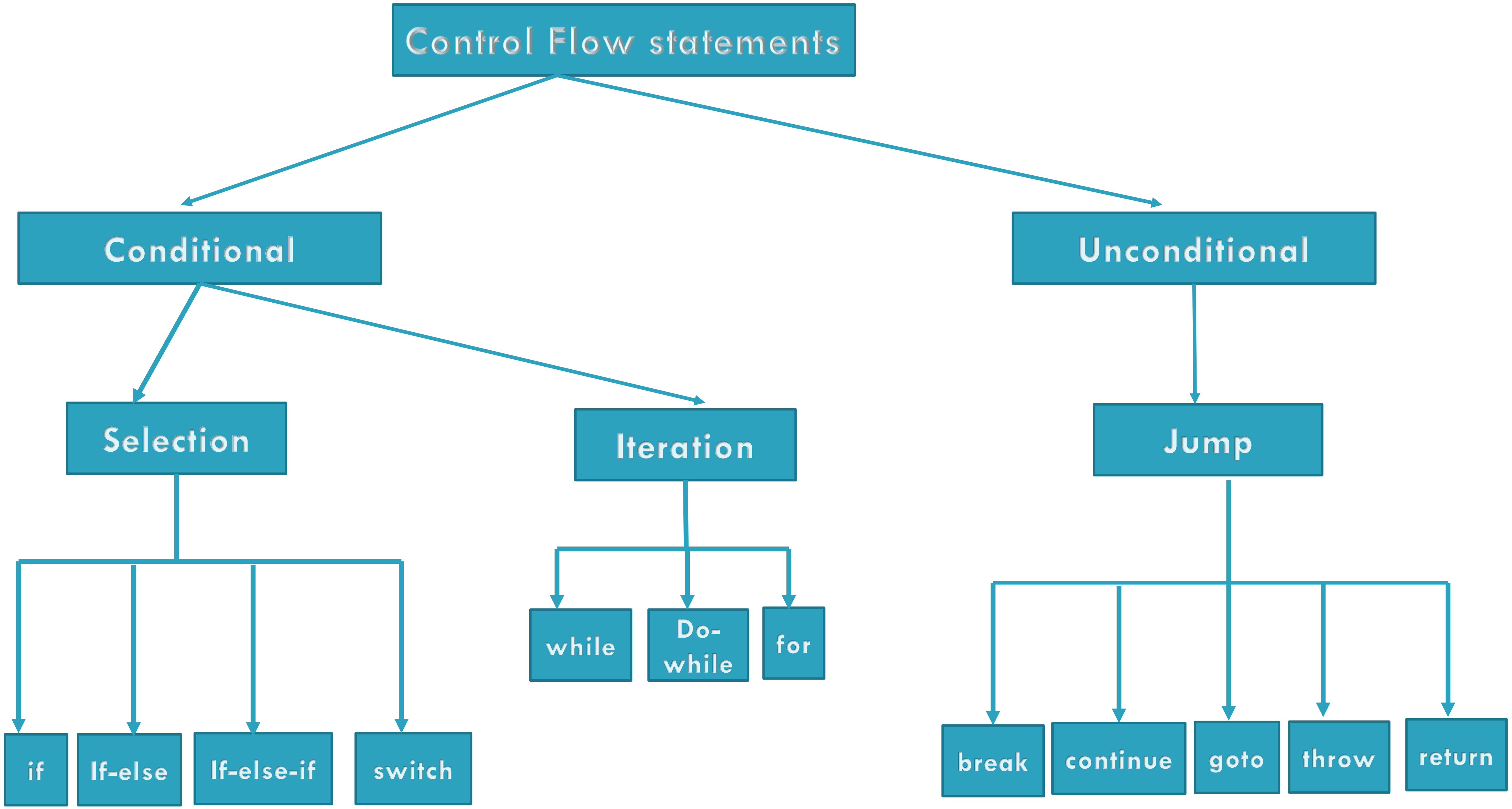
Dr P.V. Praveen Sundar
Assistant Professor,
Department of Computer Science
Adhiparasakthi College of Arts & Science,
Kalavai.

10-01-2022

III BSc Mathematics

Control Statements

- The control statements are used to control the flow of execution of the program.
- If we want to execute a specific block of instructions only when a certain condition is true, then control statements are useful.
- If we want to execute a block repeatedly, then loops are useful.
- C classifies these control statements into two categories
  - Conditional execution
  - Unconditional execution


Control Flow statements
Conditional
Unconditional
Selection
Iteration
Jump
if
If-else
If-else-if
switch
while
Do-while
for
break
continue
goto
throw
return

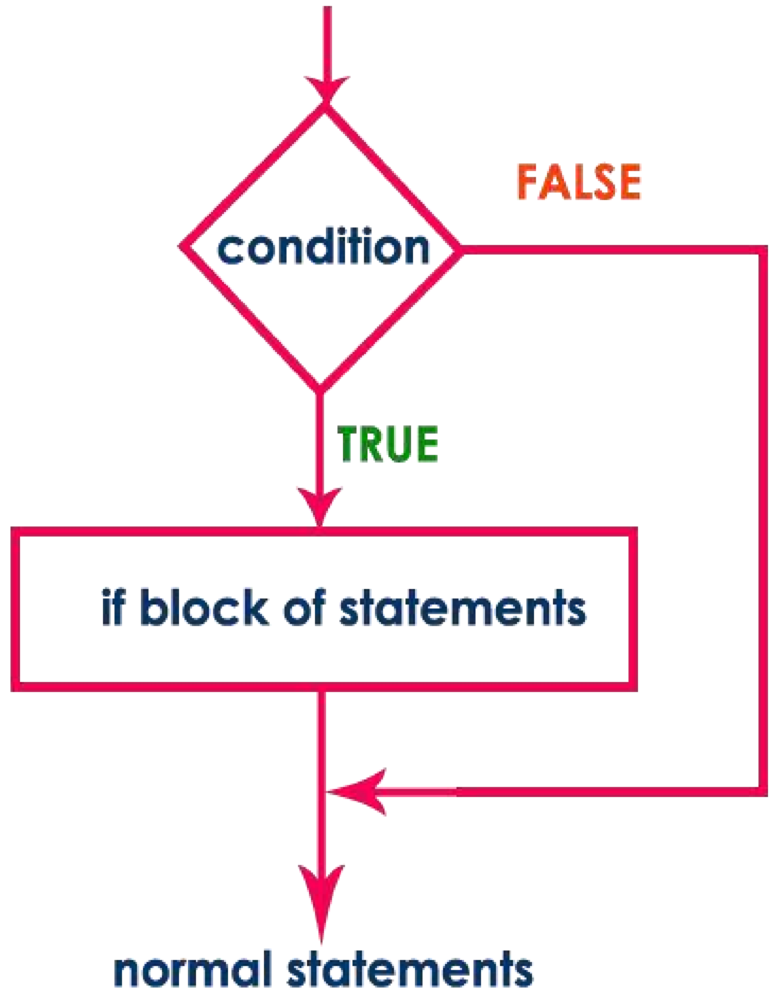
Simple if

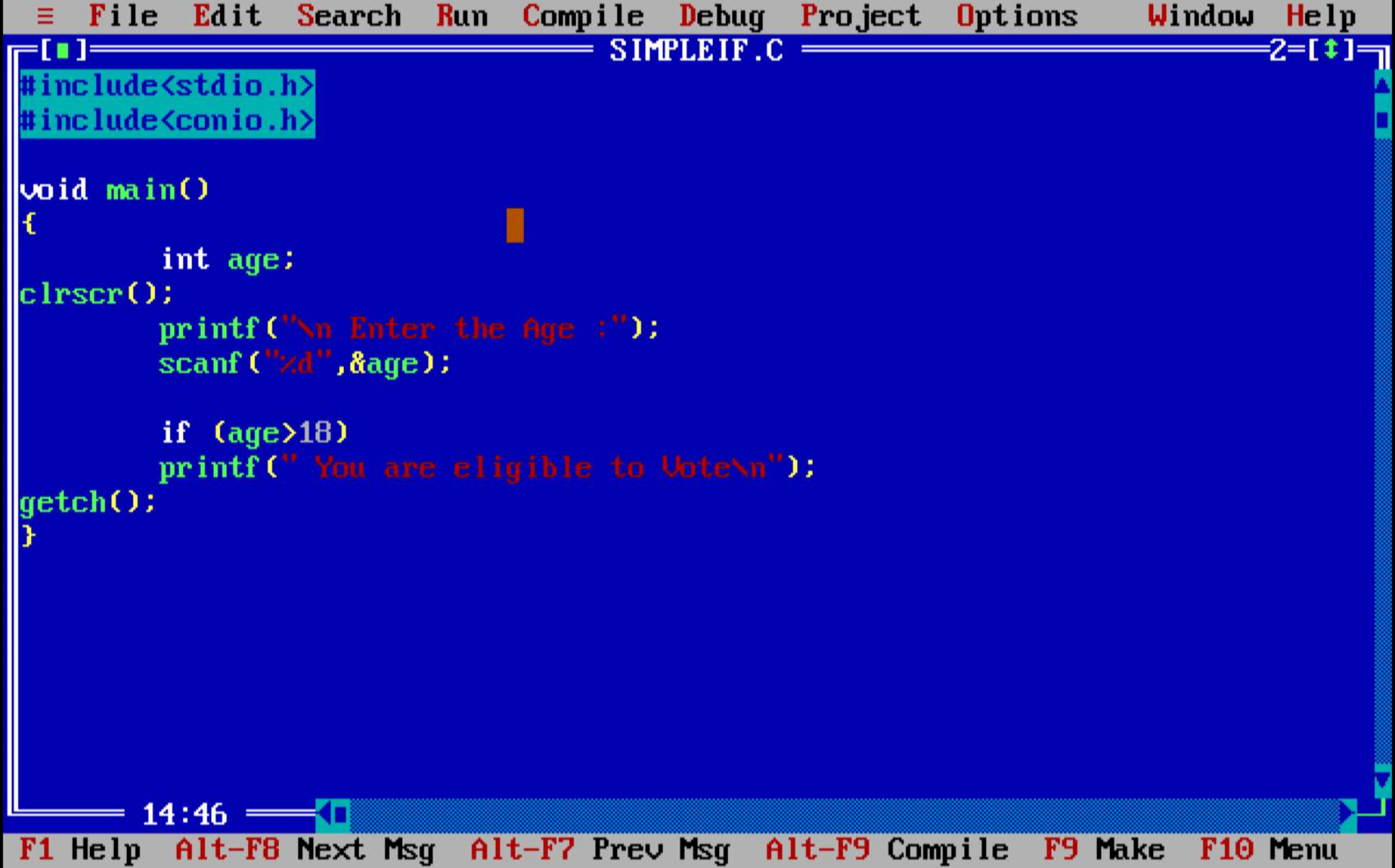
- Simple if statement is used to verify the given condition and executes the block of statements based on the condition result.
- The simple if statement evaluates specified condition.
- If it is greater than 1, it executes the next statement or block of statements.
- If the condition is 0, it skips the execution of the next statement or block of statements.
- Simple if statement is used when we have only one option that is executed or skipped based on a condition.
- The general syntax and execution flow of the simple if statement is as follows.

Syntax

```
if ( condition )
{
    ....
    block of statements;
    ....
}
```

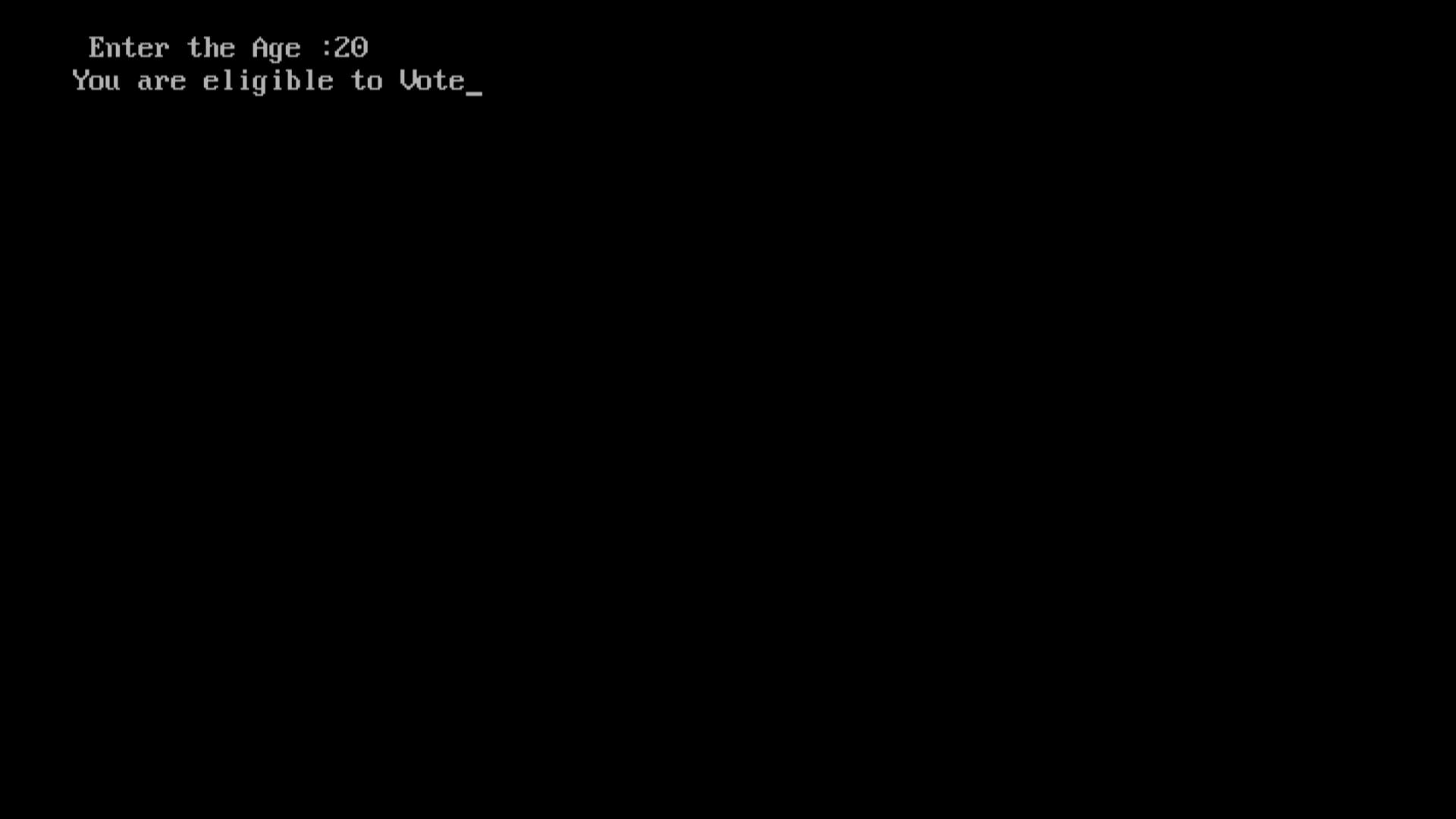
Execution flow diagram



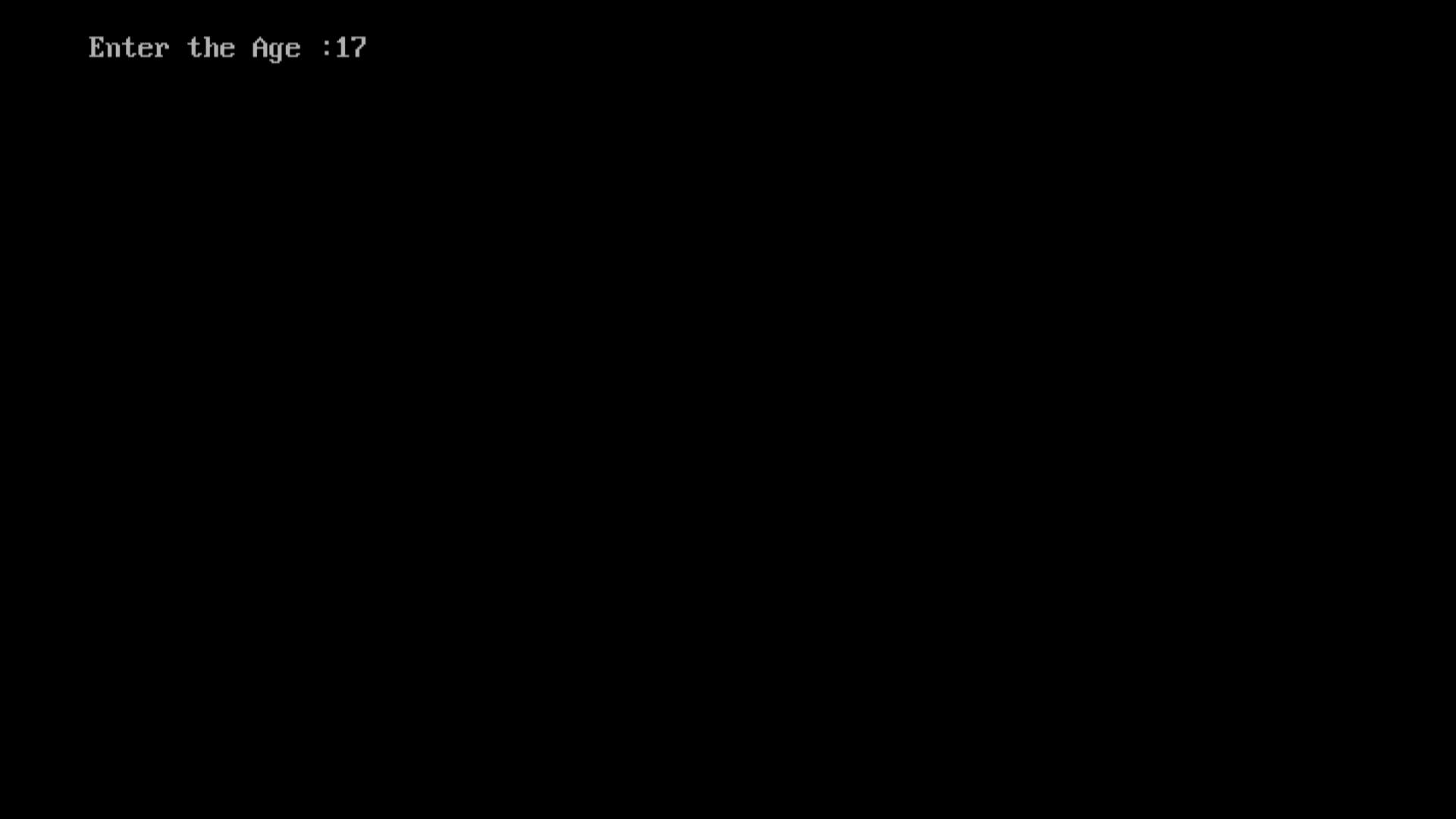

≡ File Edit Search Run Compile Debug Project Options Window Help

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 age;
clrscr();
        printf("\n Enter the Age :");
        scanf("%d",&age);

        if (age>18)
        printf(" You are eligible to Vote\n");
getch();
}
```



```
 Enter the Age :20
You are eligible to Vote_
```



```
Enter the Age :17
```

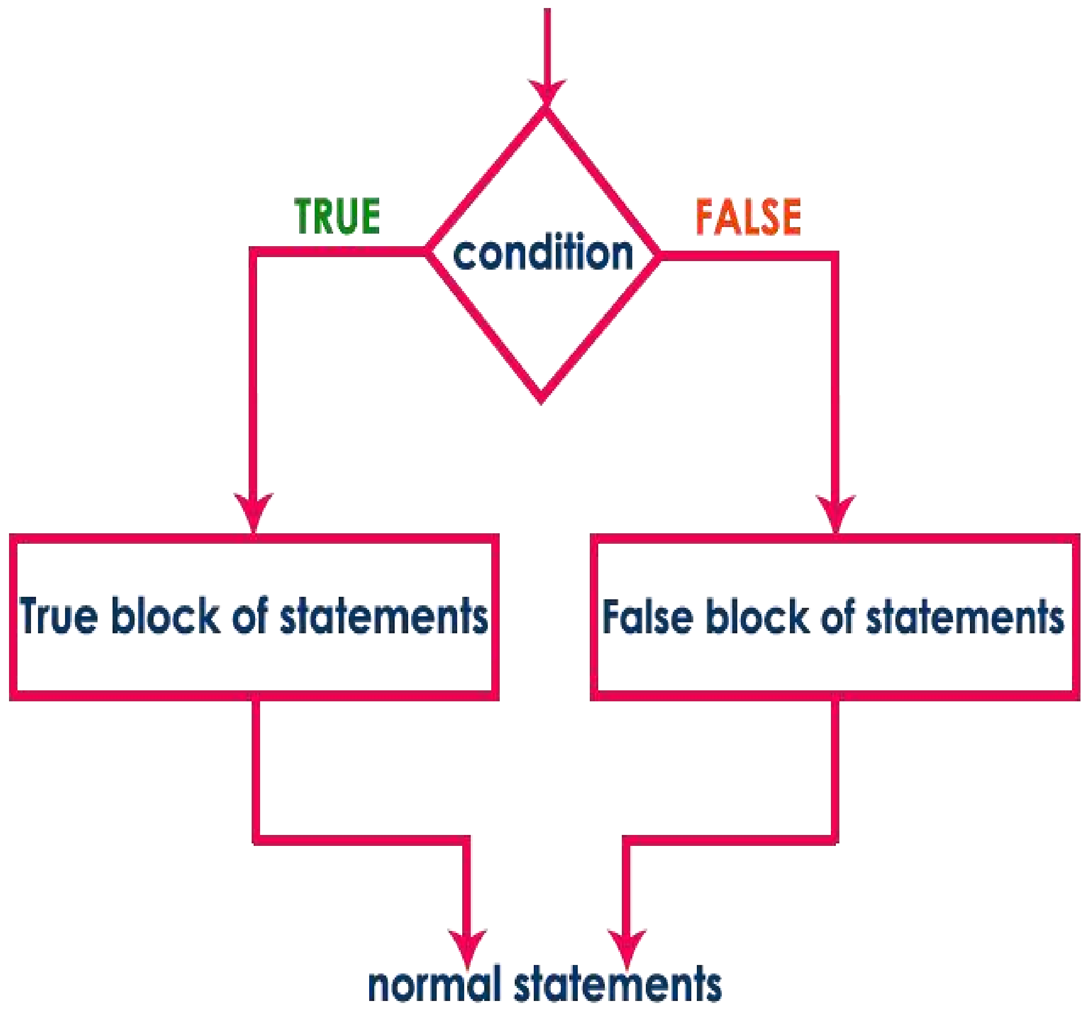
If-else statement

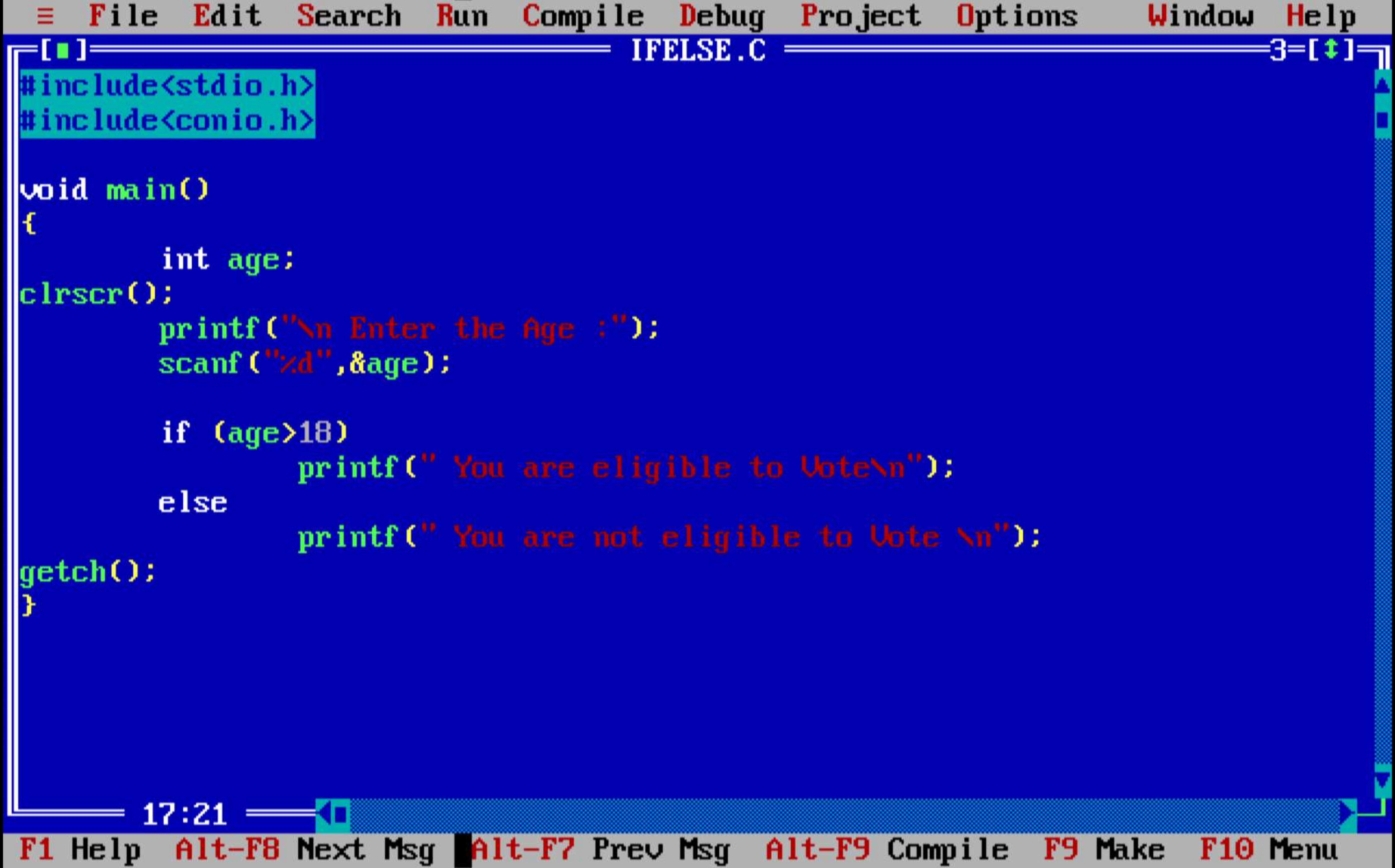
- The if-else statement is used to verify the given condition and executes only one out of the two blocks of statements based on the condition result.
- The if-else statement evaluates the specified condition.
- If it is greater than 1, it executes a block of statements (True block).
- If the condition is 0, it executes another block of statements (False block).
- The if-else statement is used when we have two options and only one option has to be executed based on a condition result (TRUE or FALSE).
- The general syntax and execution flow of the if-else statement is as follows.

Syntax

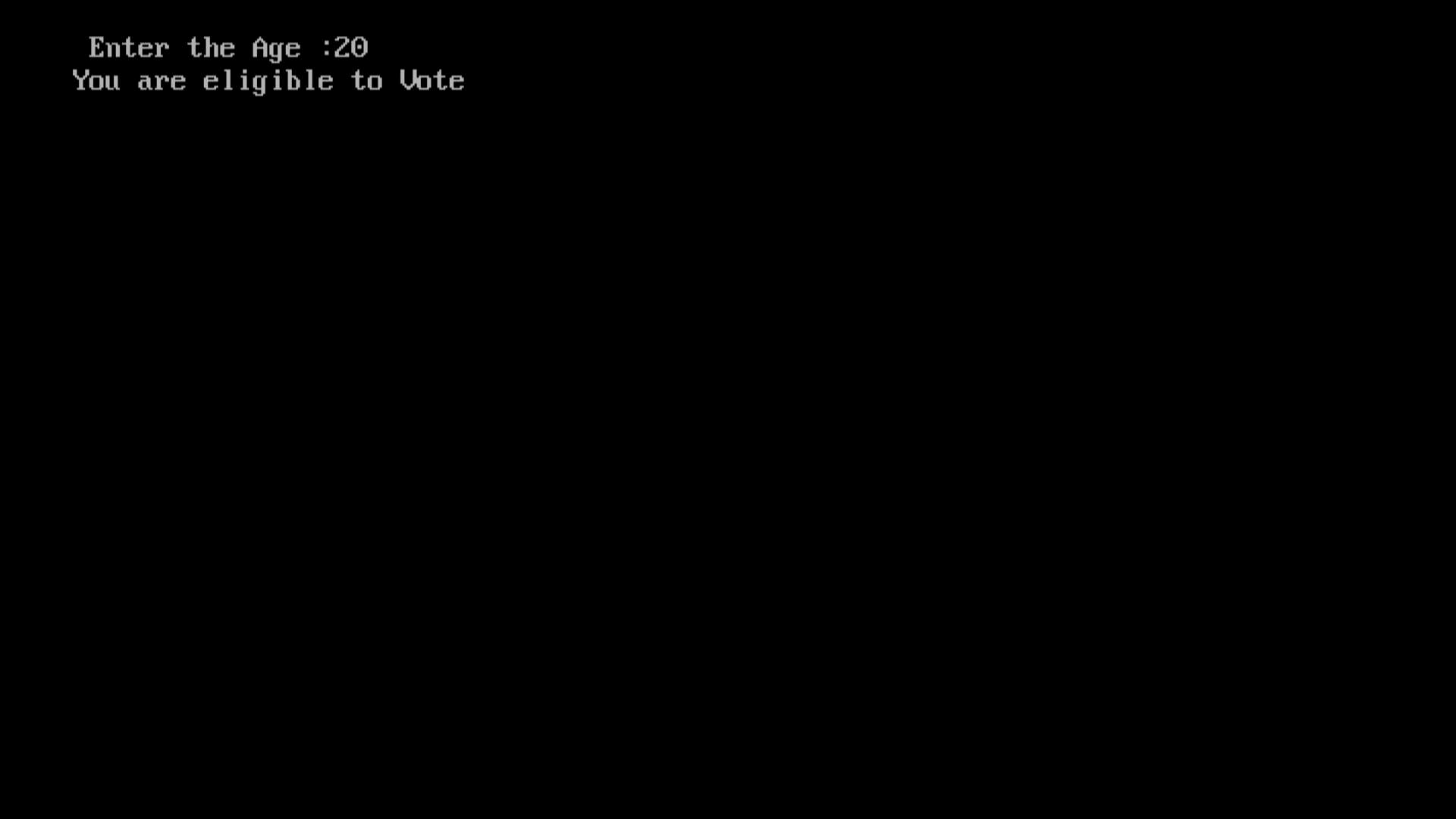
```
if ( condition )
{
    ....
    True block of statements;
    ....
}
else
{
    ....
    False block of statements;
    ....
}
```

Execution flow diagram

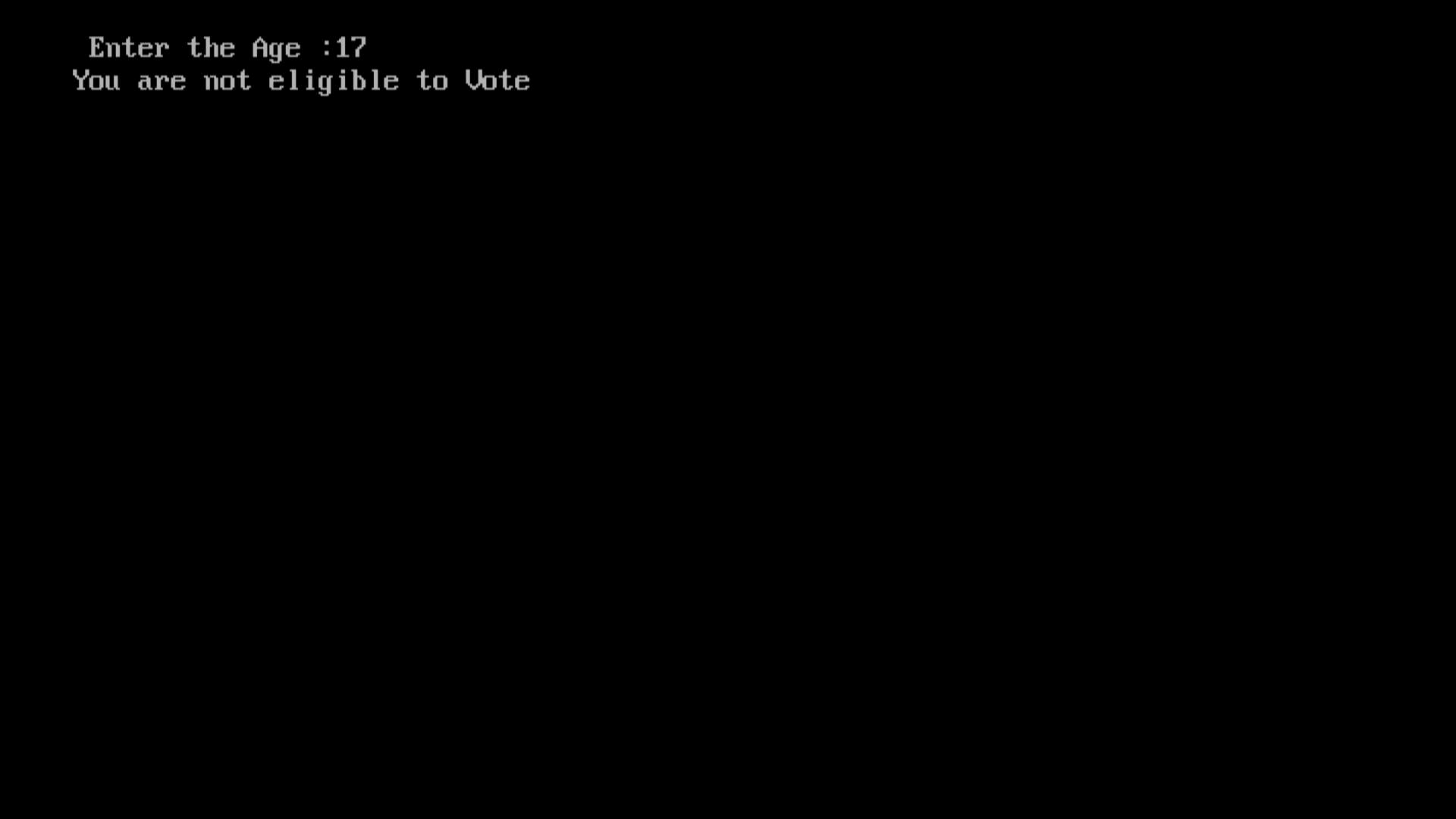



≡ File Edit Search Run Compile Debug Project Options Window Help
[■] IFELSE.C 3=[↕]
17:21
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

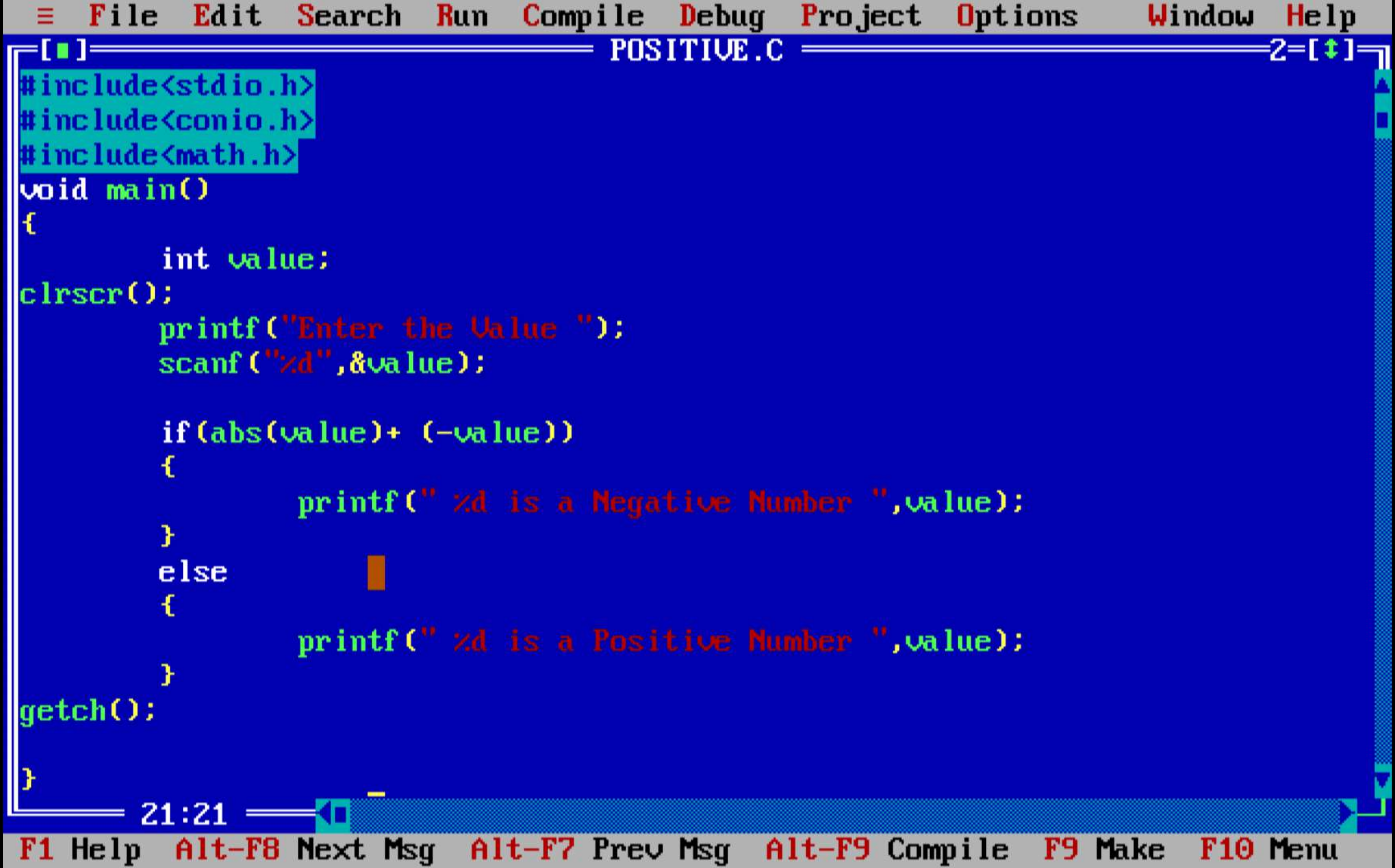
```
#include<stdio.h>
#include<conio.h>

void main()
{
        int age;
clrscr();
        printf("\n Enter the Age :");
        scanf("%d",&age);

        if (age>18)
                printf(" You are eligible to Vote\n");
        else
                printf(" You are not eligible to Vote \n");
getch();
}
```



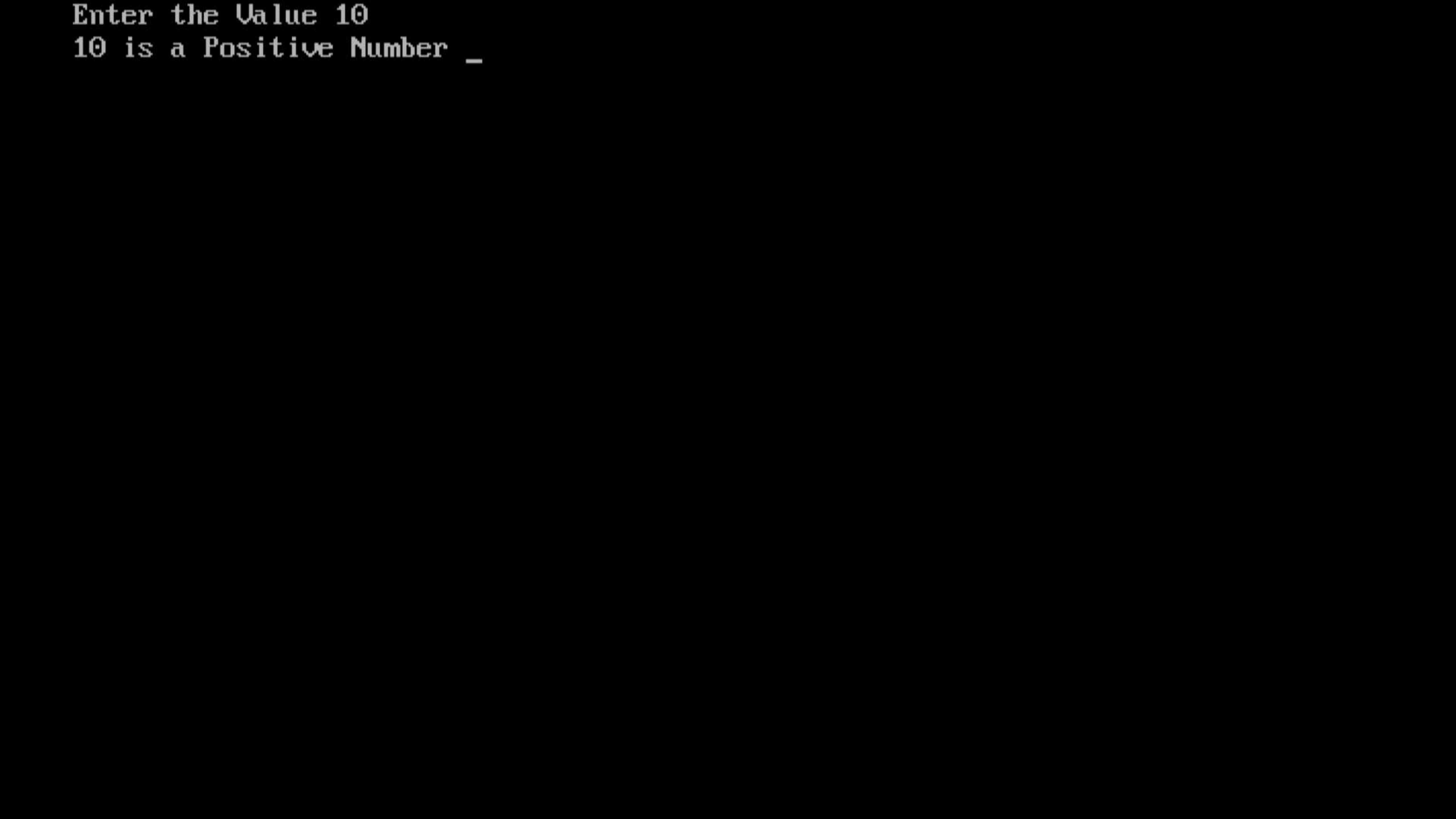
```
 Enter the Age :20
You are eligible to Vote
```



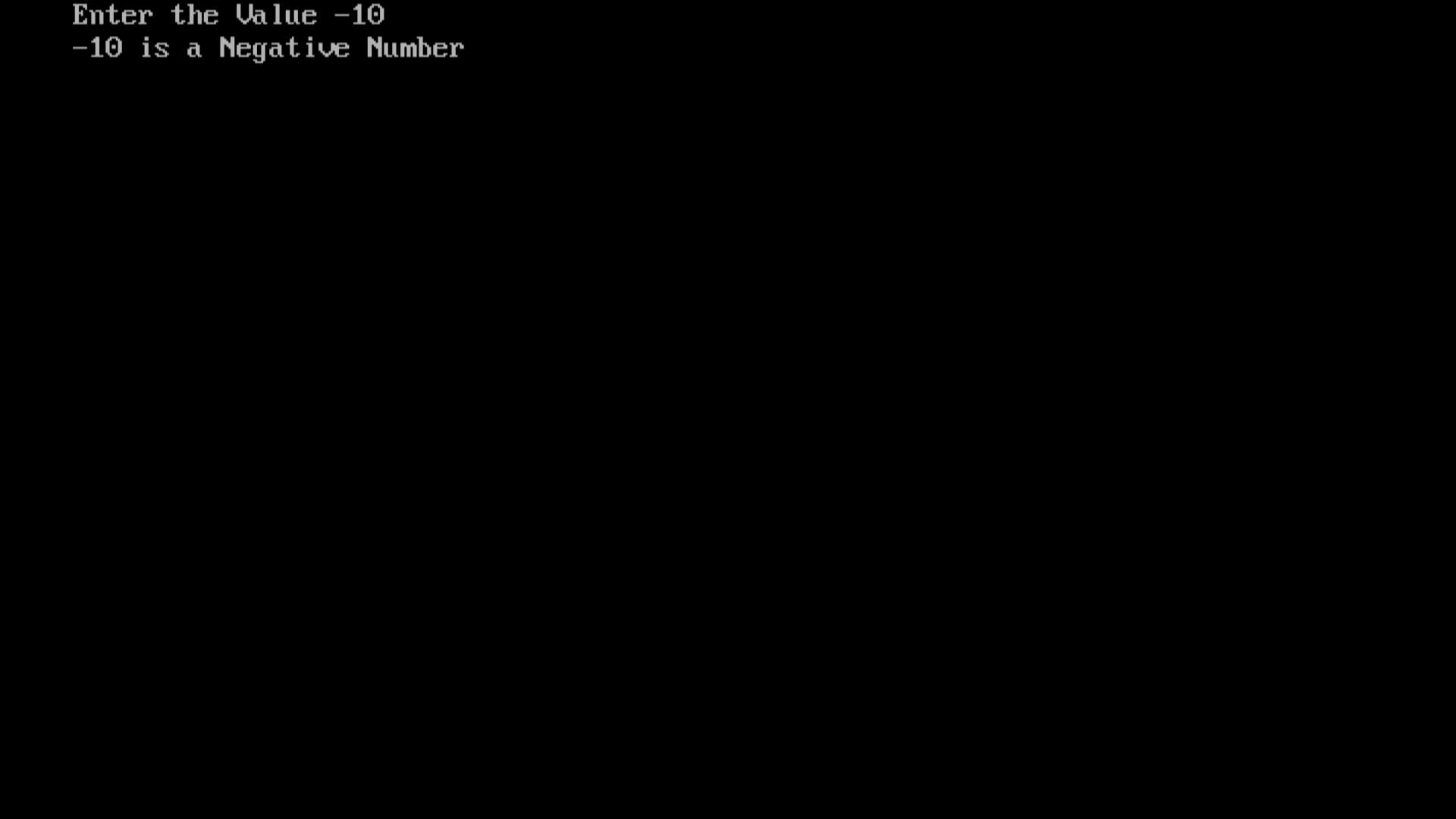
```
 Enter the Age :17
You are not eligible to Vote
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
POSITIVE.C
2=[↕]
21:21
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>
void main()
{
        int value;
clrscr();
        printf("Enter the Value ");
        scanf("%d",&value);

        if(abs(value)+ (-value))
        {
                printf(" %d is a Negative Number ",value);
        }
        else
        {
                printf(" %d is a Positive Number ",value);
        }
getch();

}
```



```
Enter the Value 10
10 is a Positive Number _
```



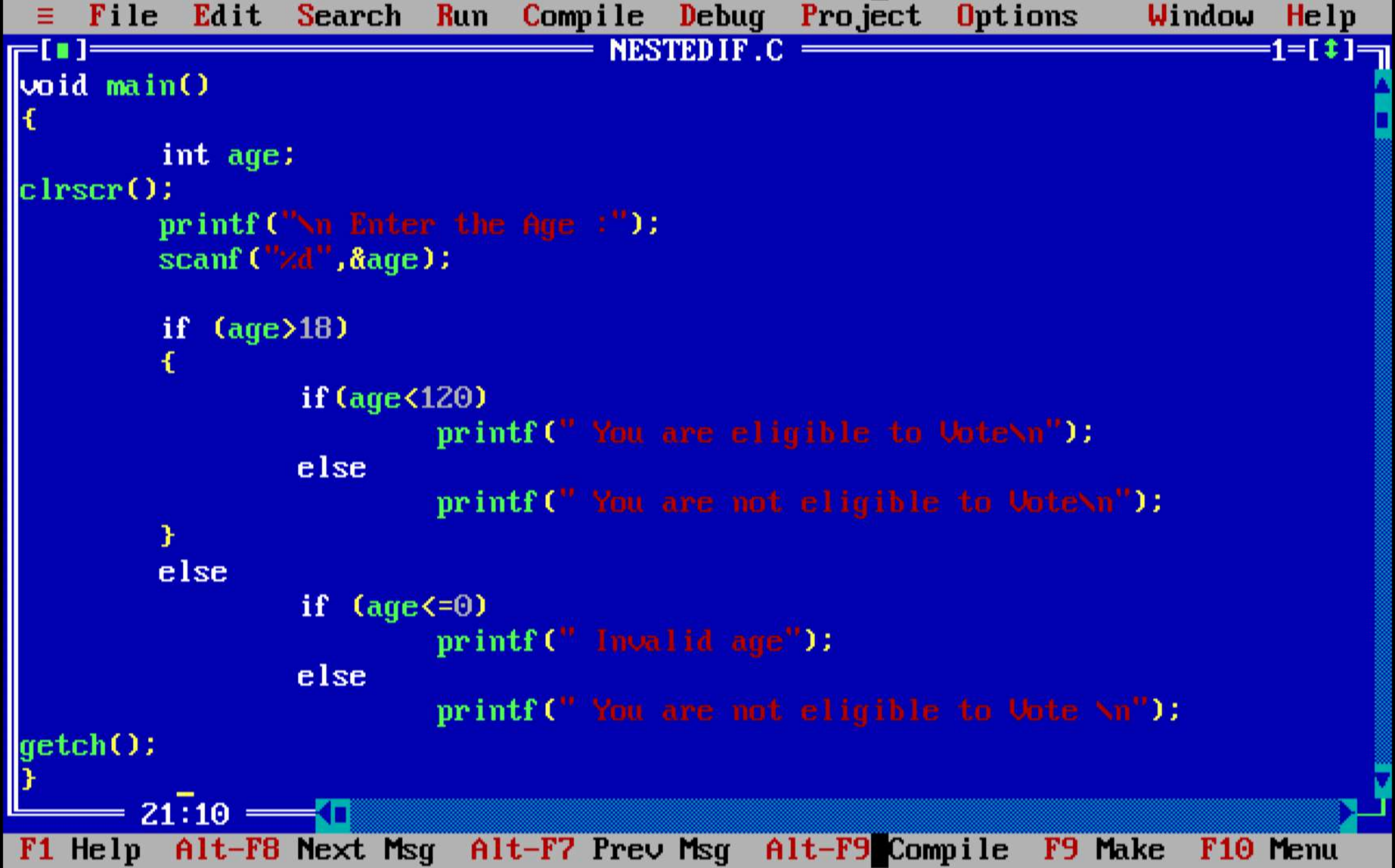
```
Enter the Value -10
-10 is a Negative Number
```

Nested if statement

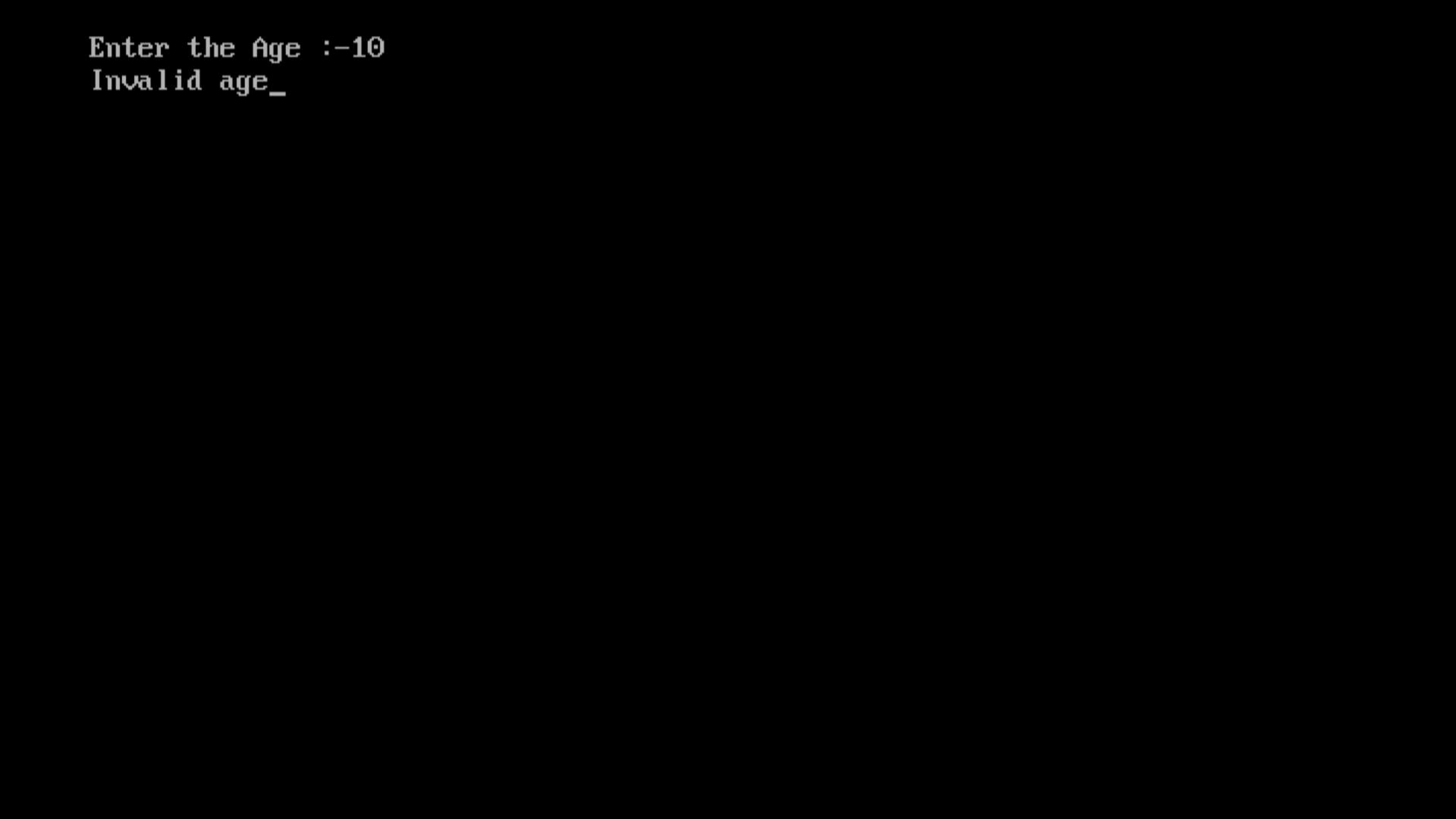
- Writing a if statement inside another if statement is called nested if statement.
- The nested if statement can be defined using any combination of simple if & if-else statements.
- The general syntax of the nested if statement is as follows...

Syntax

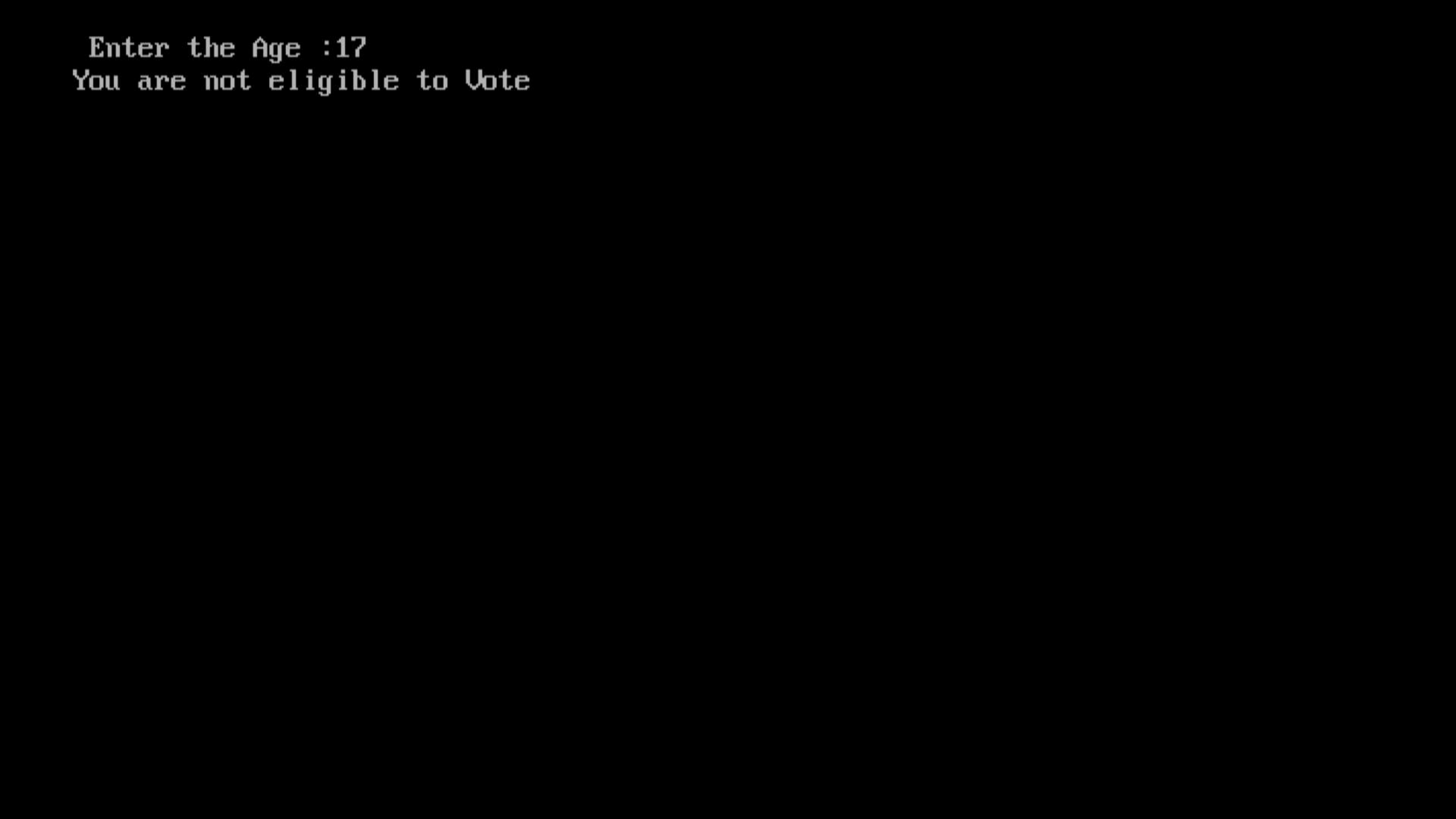
```
if ( condition1 )
{
    if ( condition2 )
    {
        ....
        True block of statements 1;
    }
    ....
}
else
{
     False block of condition1;
}
```


≡ File Edit Search Run Compile Debug Project Options Window Help
NESTEDIF.C
1
21:10
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

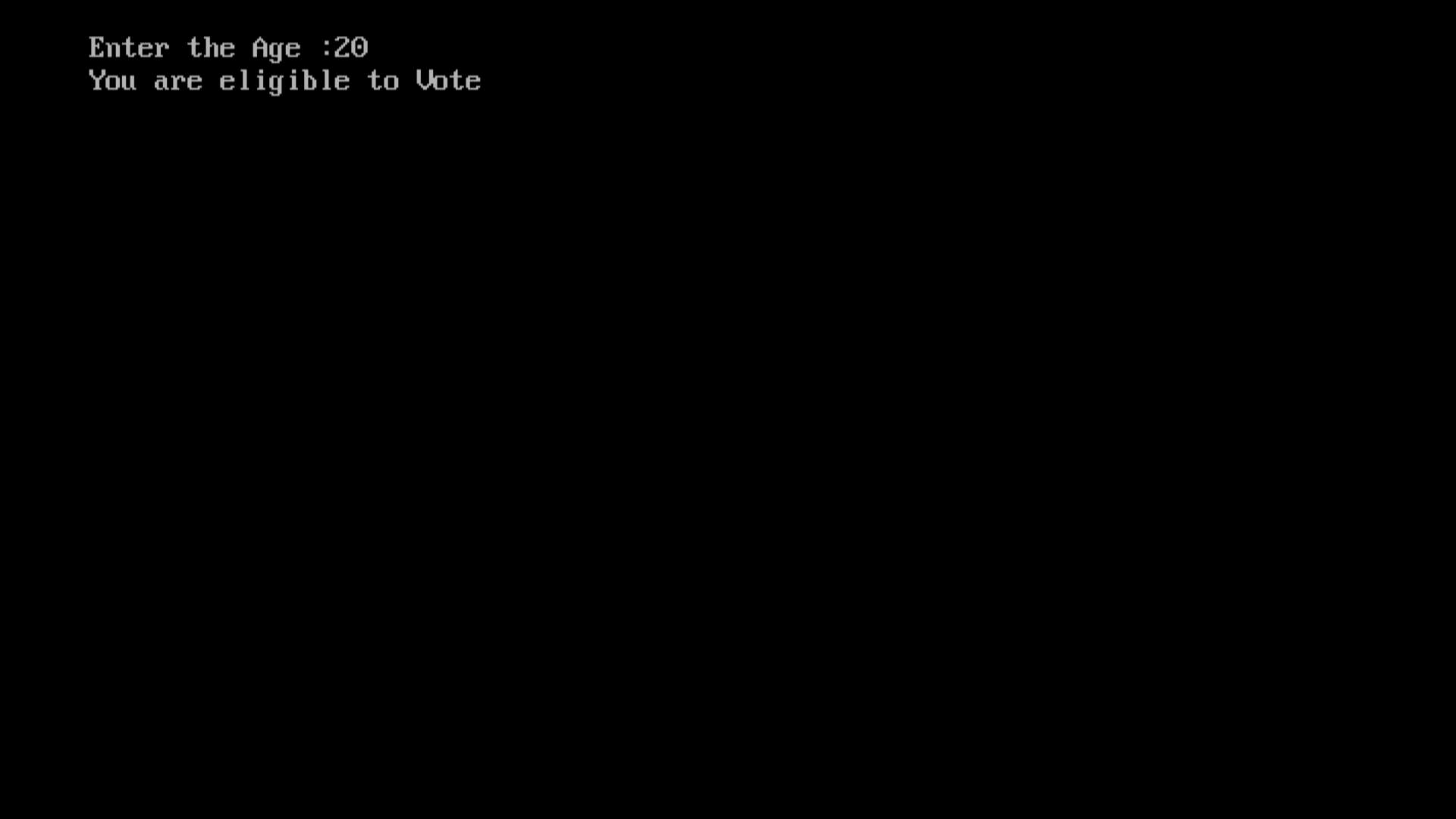
```
void main()
{
        int age;
clrscr();
        printf("\n Enter the Age :");
        scanf("%d",&age);

        if (age>18)
        {
                if(age<120)
                        printf(" You are eligible to Vote\n");
                else
                        printf(" You are not eligible to Vote\n");
        }
        else
                if (age<=0)
                        printf(" Invalid age");
                else
                        printf(" You are not eligible to Vote \n");
getch();
}
```



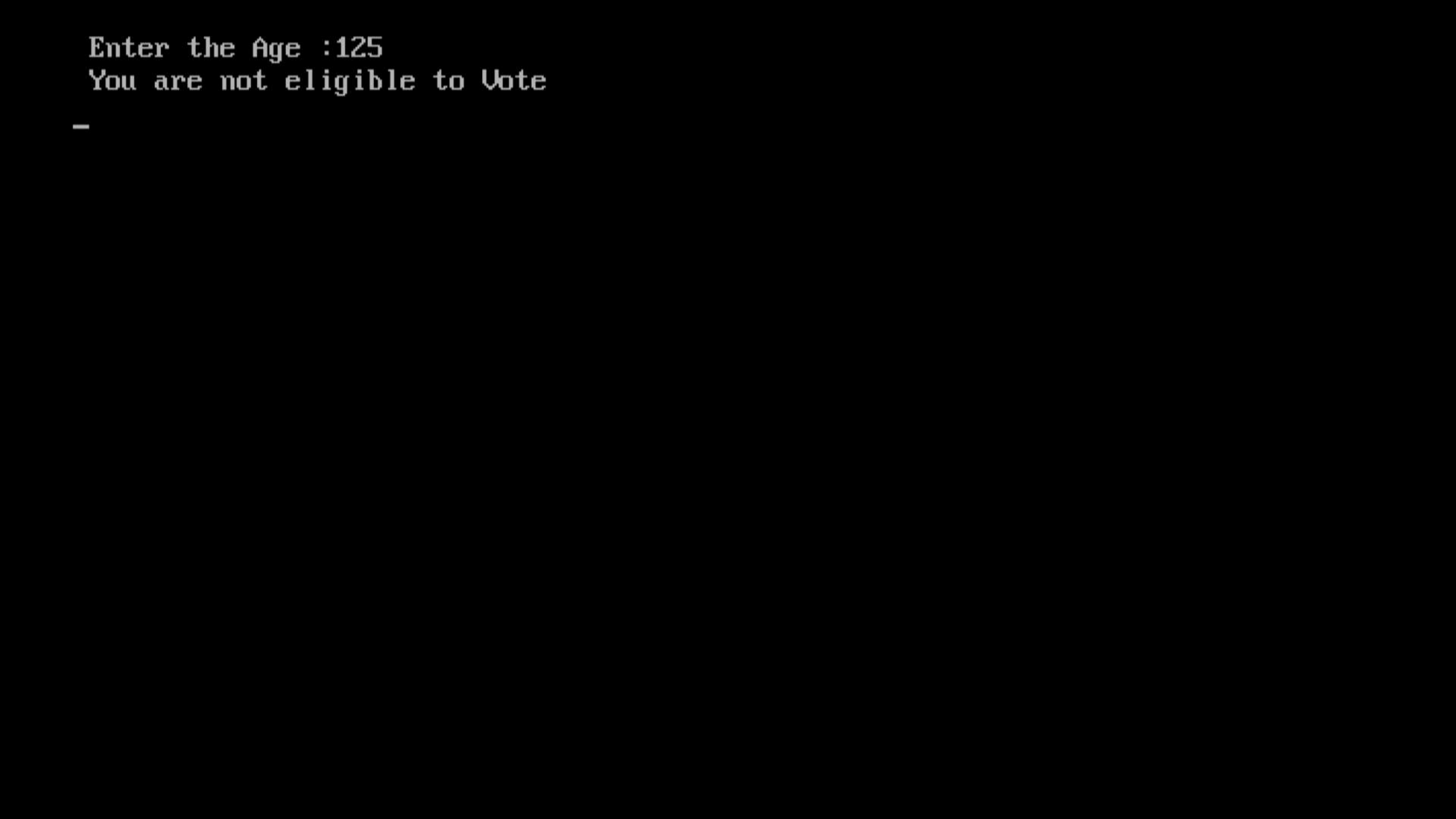
```
Enter the Age :-10
Invalid age_
```



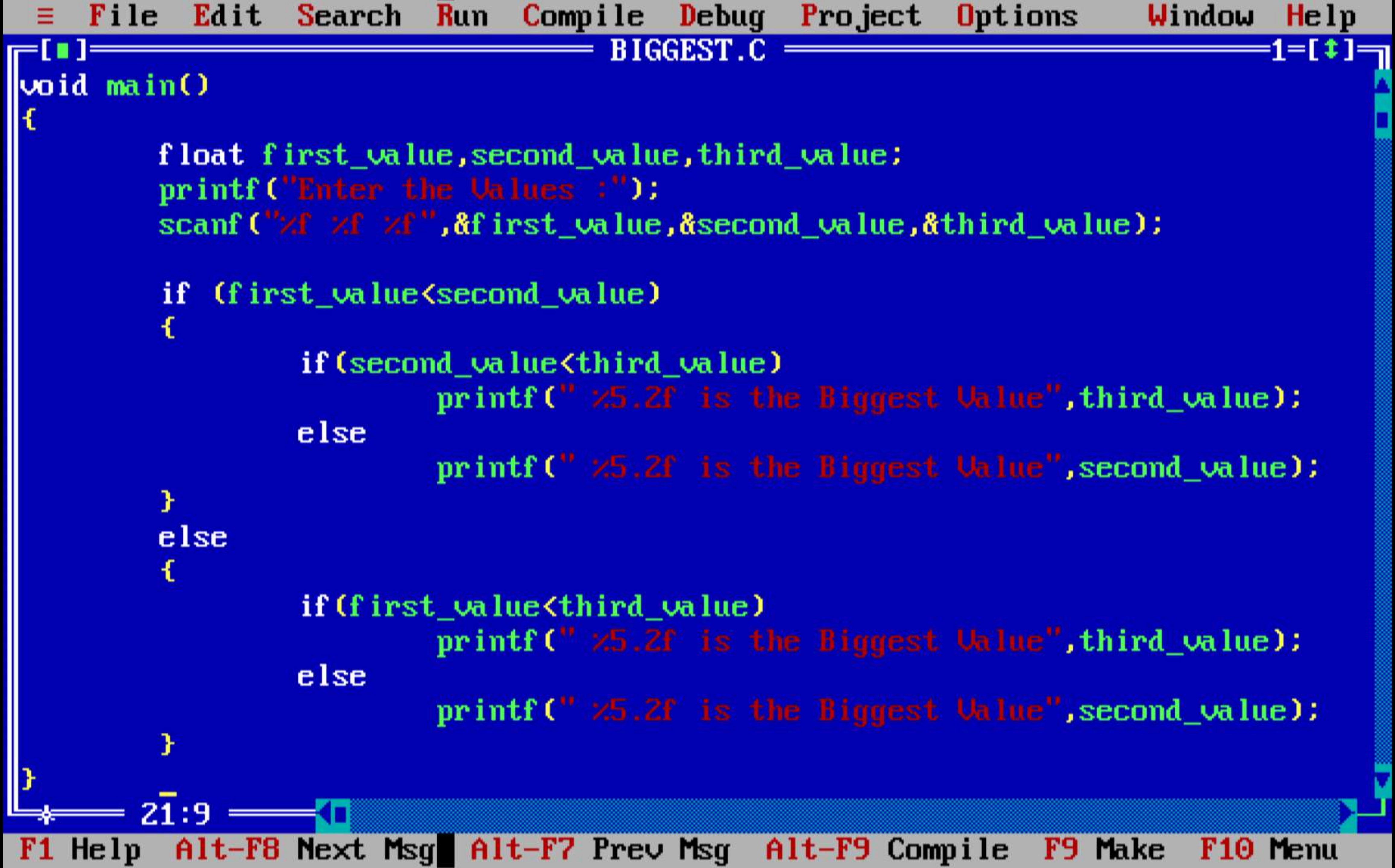
```
 Enter the Age :17
You are not eligible to Vote
```



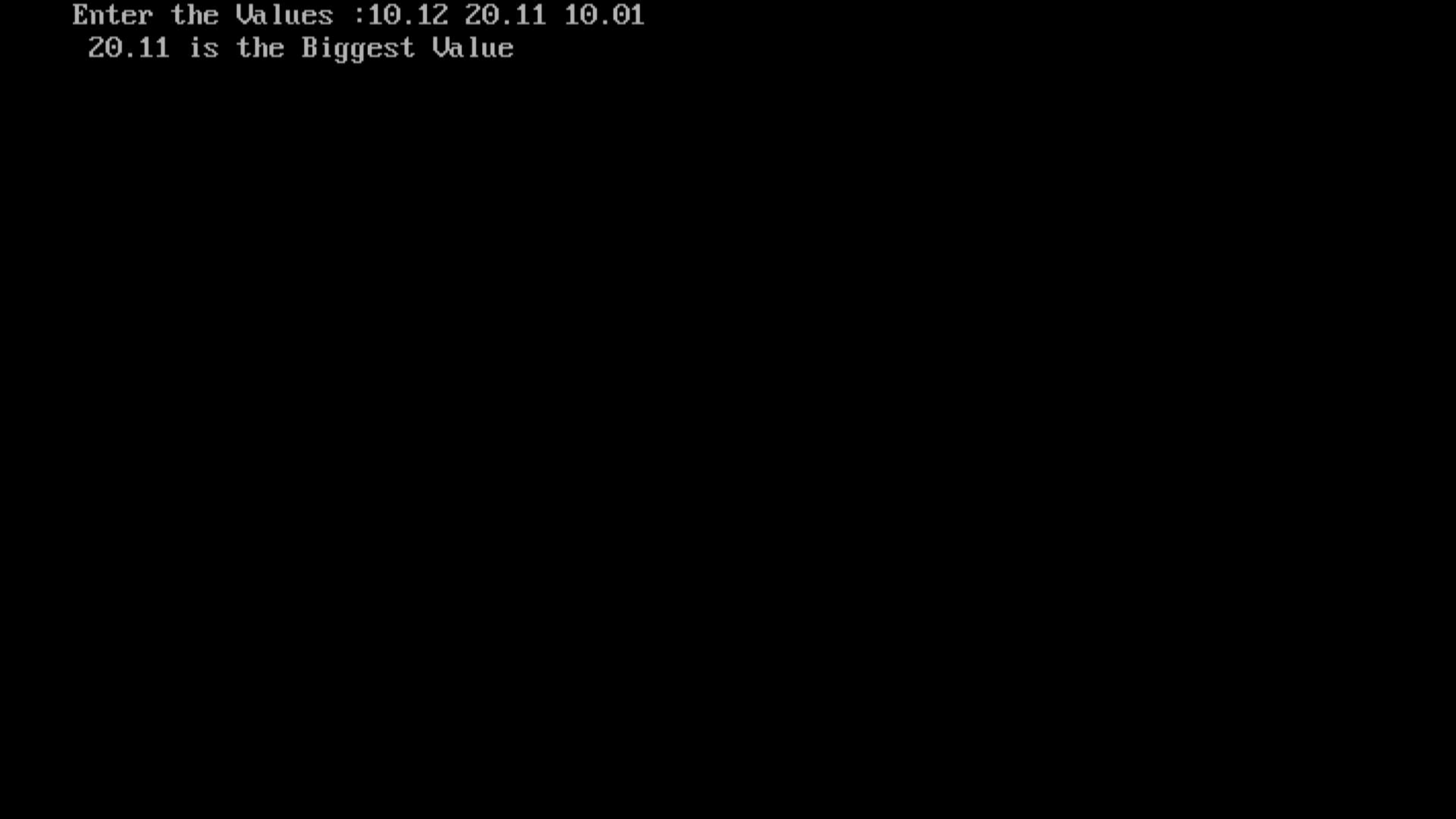
```
Enter the Age :20
You are eligible to Vote
```



```
Enter the Age :125
You are not eligible to Vote

_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
BIGGEST.C
1
21:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

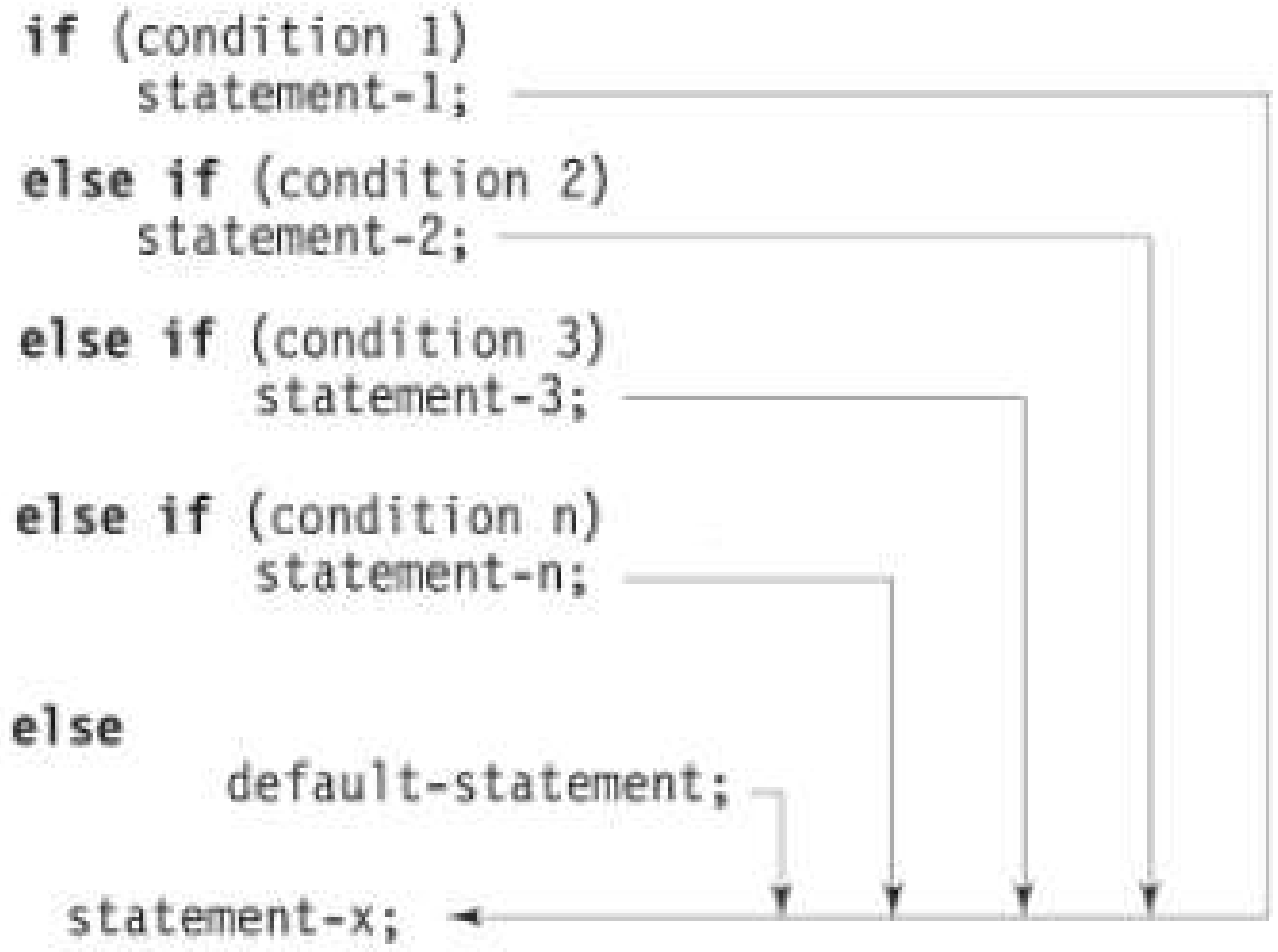
```
void main()
{
        float first_value,second_value,third_value;
        printf("Enter the Values :");
        scanf("%f %f %f",&first_value,&second_value,&third_value);

        if (first_value<second_value)
        {
                if(second_value<third_value)
                        printf(" %5.2f is the Biggest Value",third_value);
                else
                        printf(" %5.2f is the Biggest Value",second_value);
        }
        else
        {
                if(first_value<third_value)
                        printf(" %5.2f is the Biggest Value",third_value);
                else
                        printf(" %5.2f is the Biggest Value",second_value);
        }
}
```



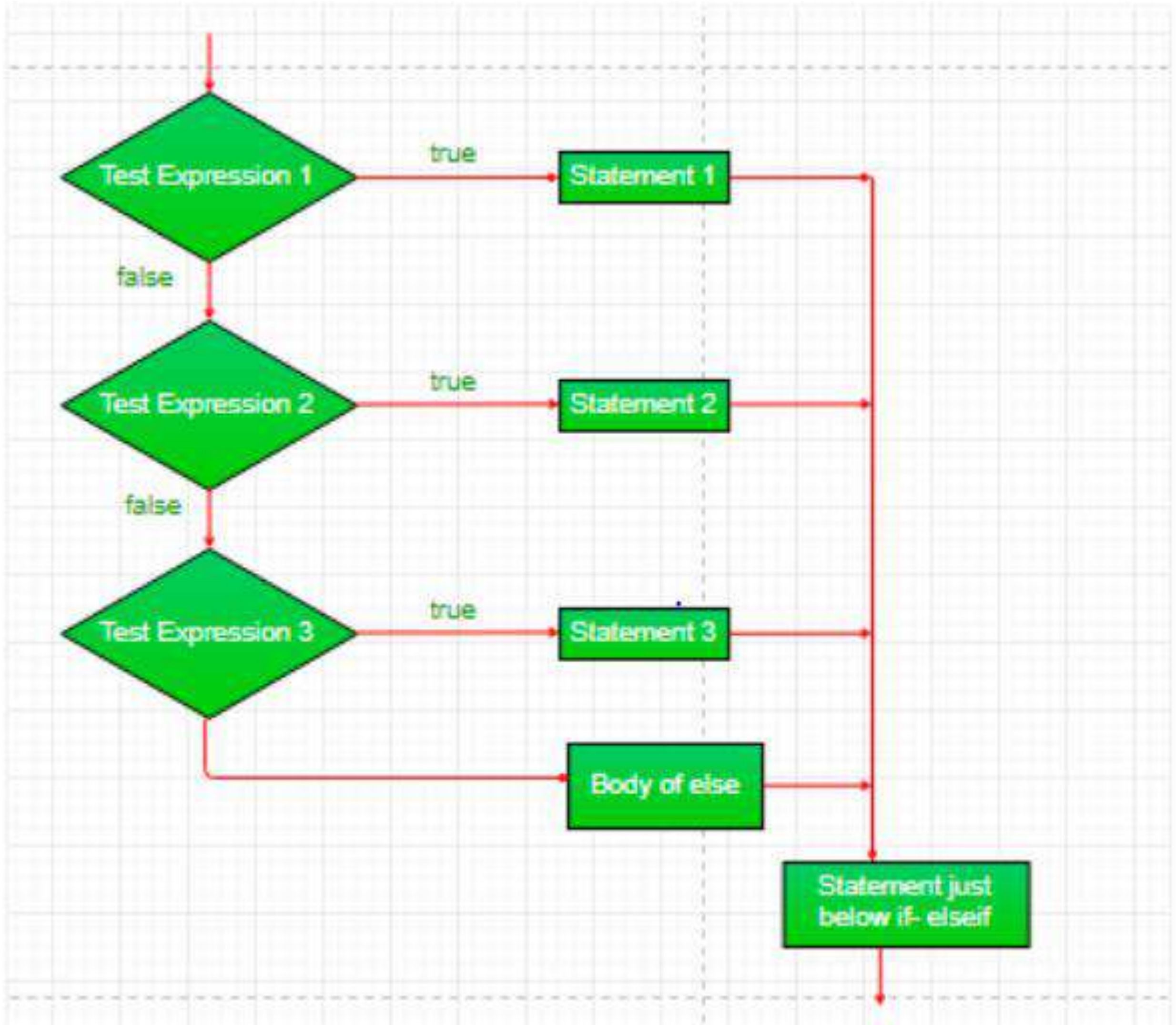
```
Enter the Values :10.12 20.11 10.01
 20.11 is the Biggest Value
```

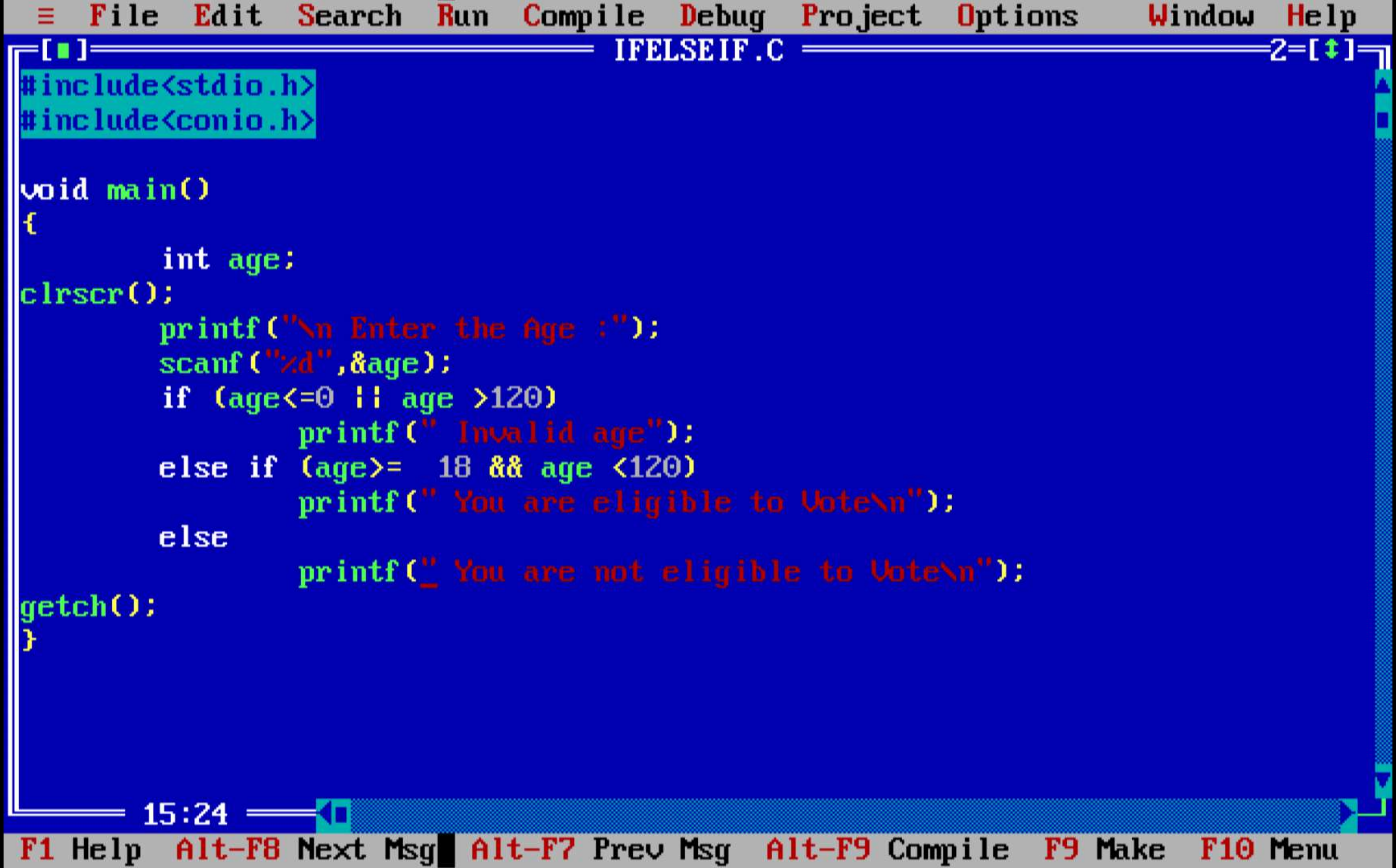
if...else if...else statement

- The if-else-if ladder statement executes one condition from multiple statements. The execution starts from top and checked for each if condition.
- We can use multiple else if blocks to add multiple conditions but it requires at least one if block at the beginning, we can't directly write else and else if statements without having any if block.
- The statement of if block will be executed which evaluates to be true. If none of the if condition evaluates to be true then the last else block is evaluated.
- The general syntax of the if-else-if statement is as follows...

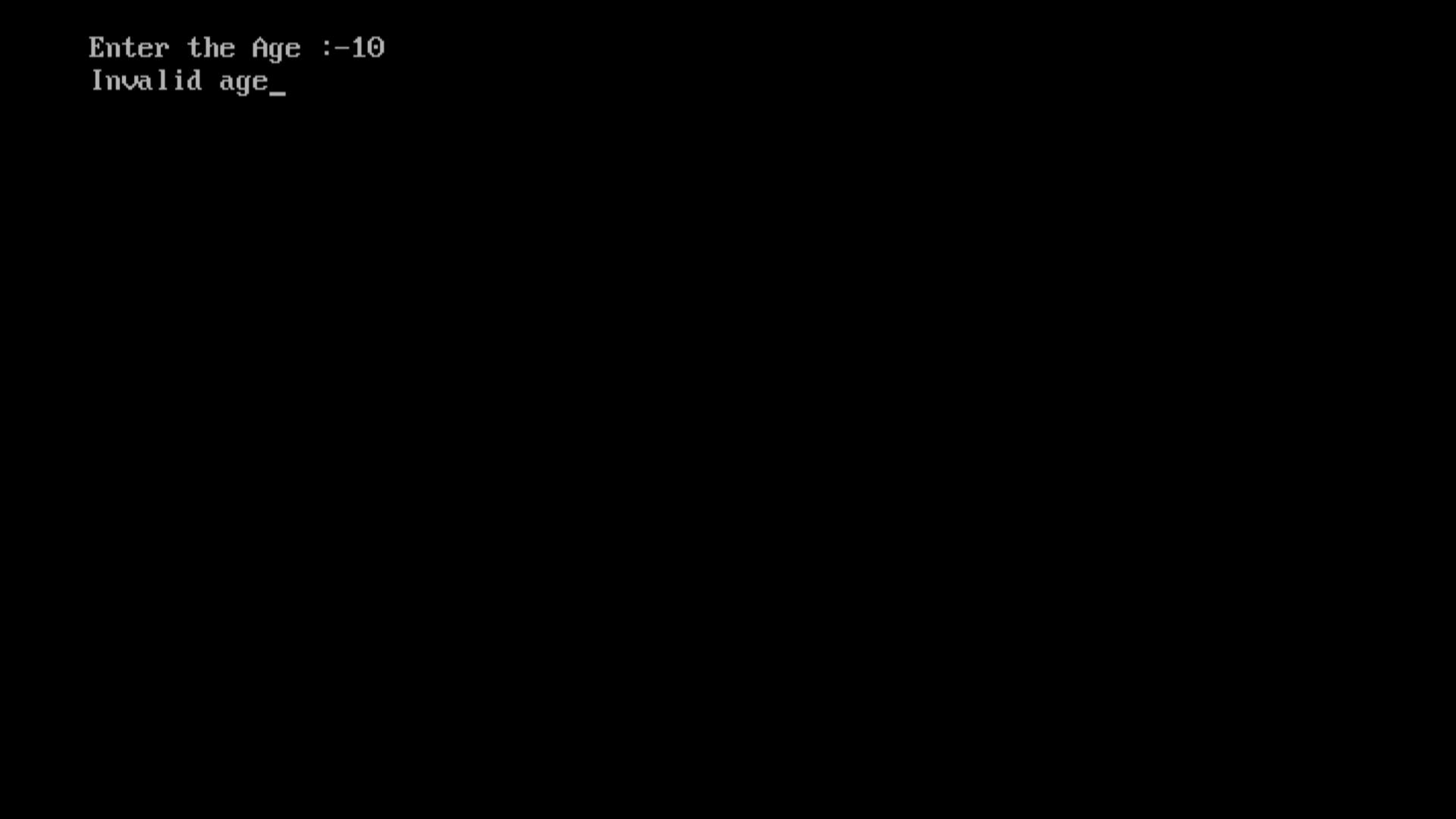


```
if (condition 1)
     statement-1;
else if (condition 2)
     statement-2;
else if (condition 3)
          statement-3;
else if (condition n)
          statement-n;
else
          default-statement;
  statement-x;
```

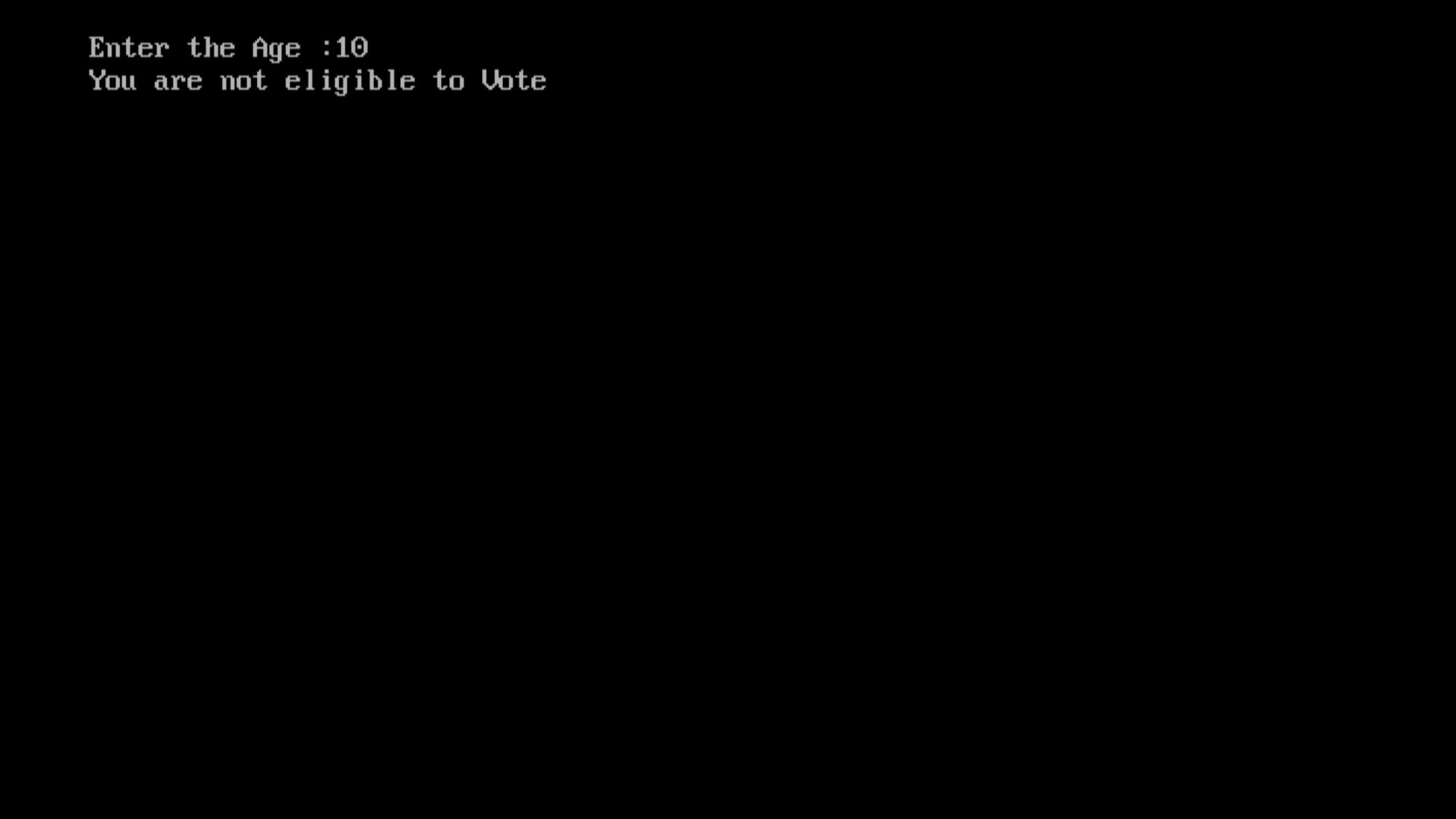

Test Expression 1
true
Statement 1
false
Test Expression 2
true
Statement 2
false
Test Expression 3
true
Statement 3
Body of else
Statement just below if- elseif


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] IFELSEIF.C 2=[↕]
15:24
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

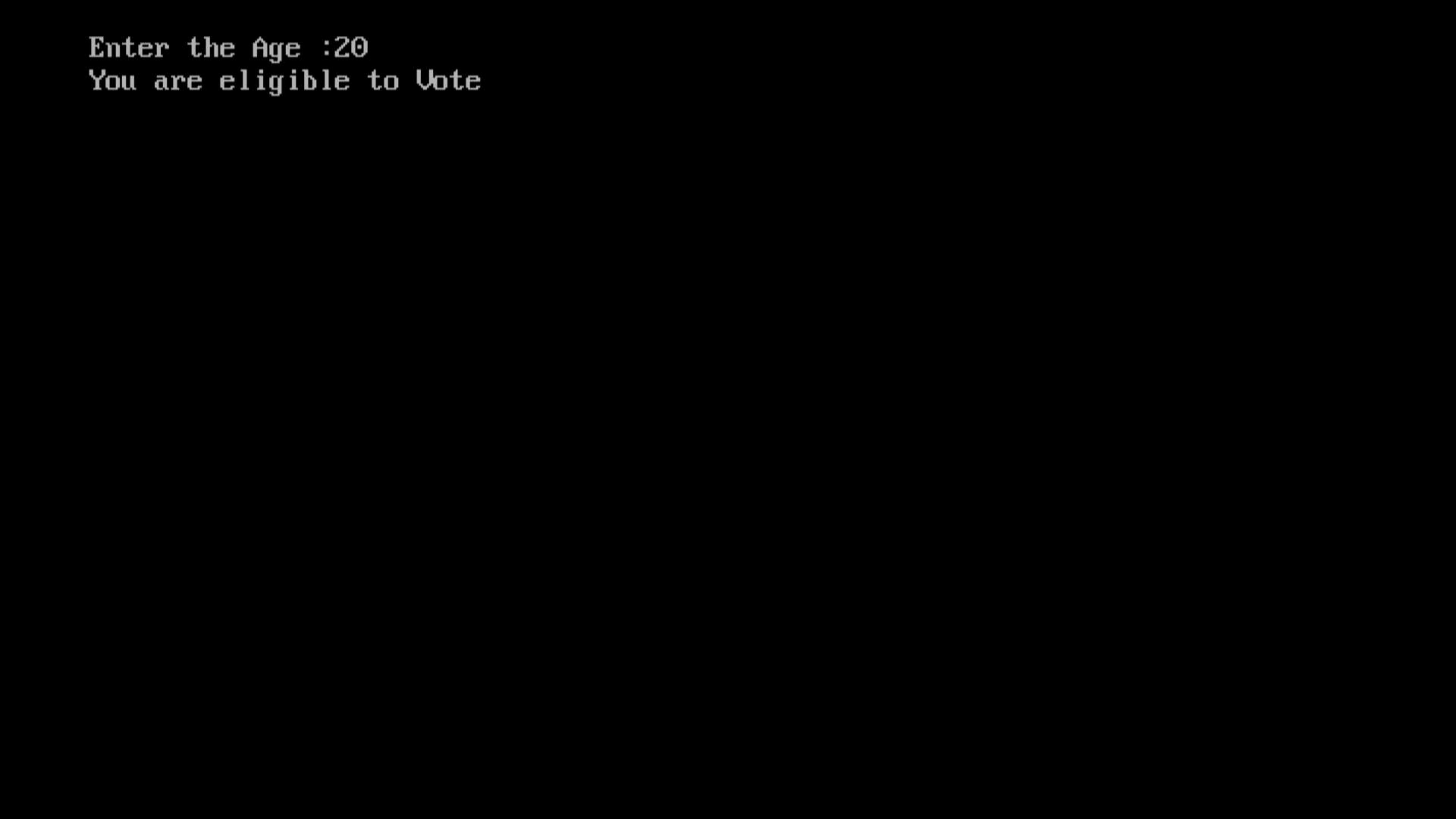
```
#include<stdio.h>
#include<conio.h>

void main()
{
        int age;
clrscr();
        printf("\n Enter the Age :");
        scanf("%d",&age);
        if (age<=0 || age >120)
                printf(" Invalid age");
        else if (age>=  18 && age <120)
                printf(" You are eligible to Vote\n");
        else
                printf(" You are not eligible to Vote\n");
getch();
}
```



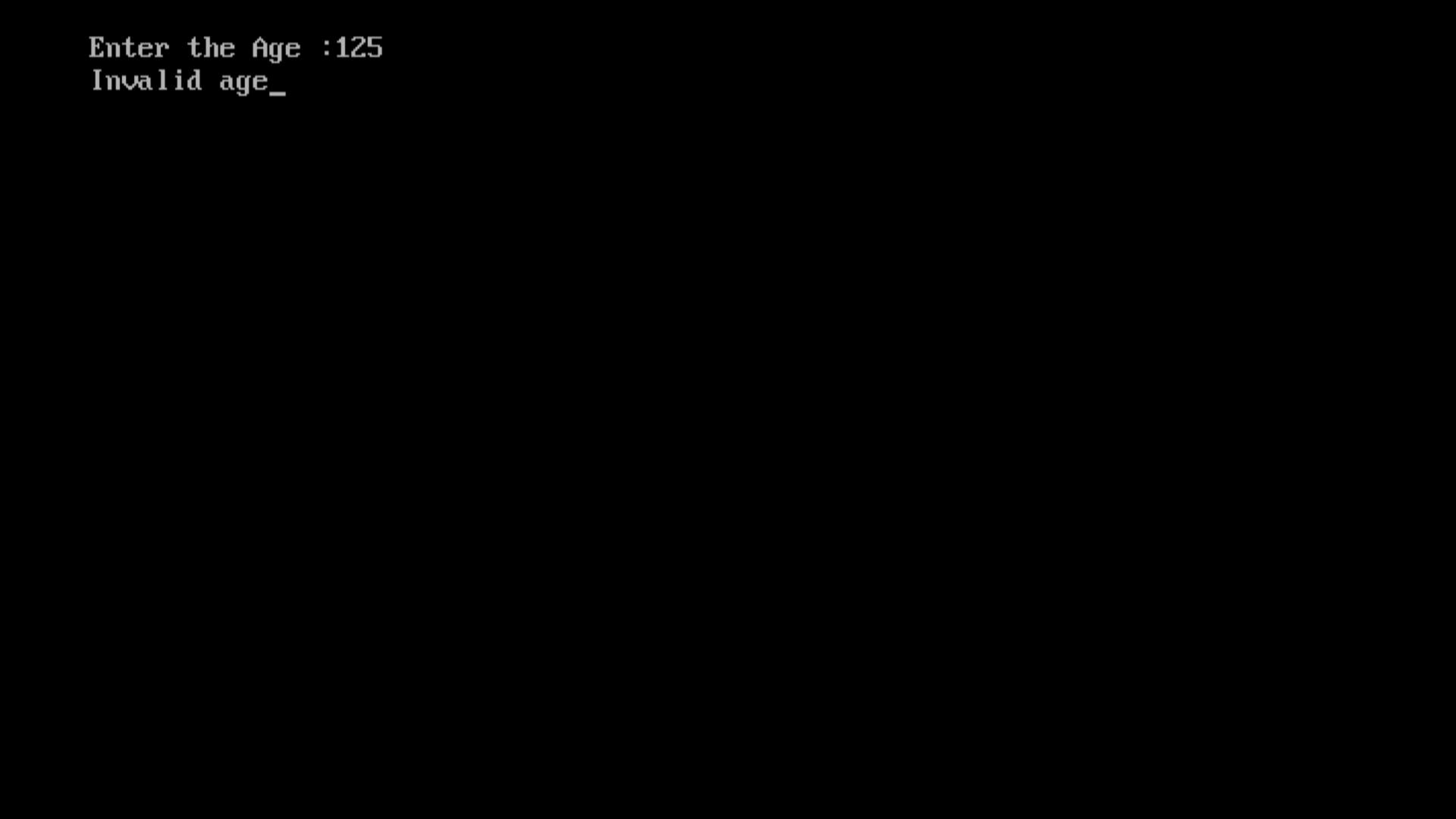
```
Enter the Age :-10
Invalid age_
```



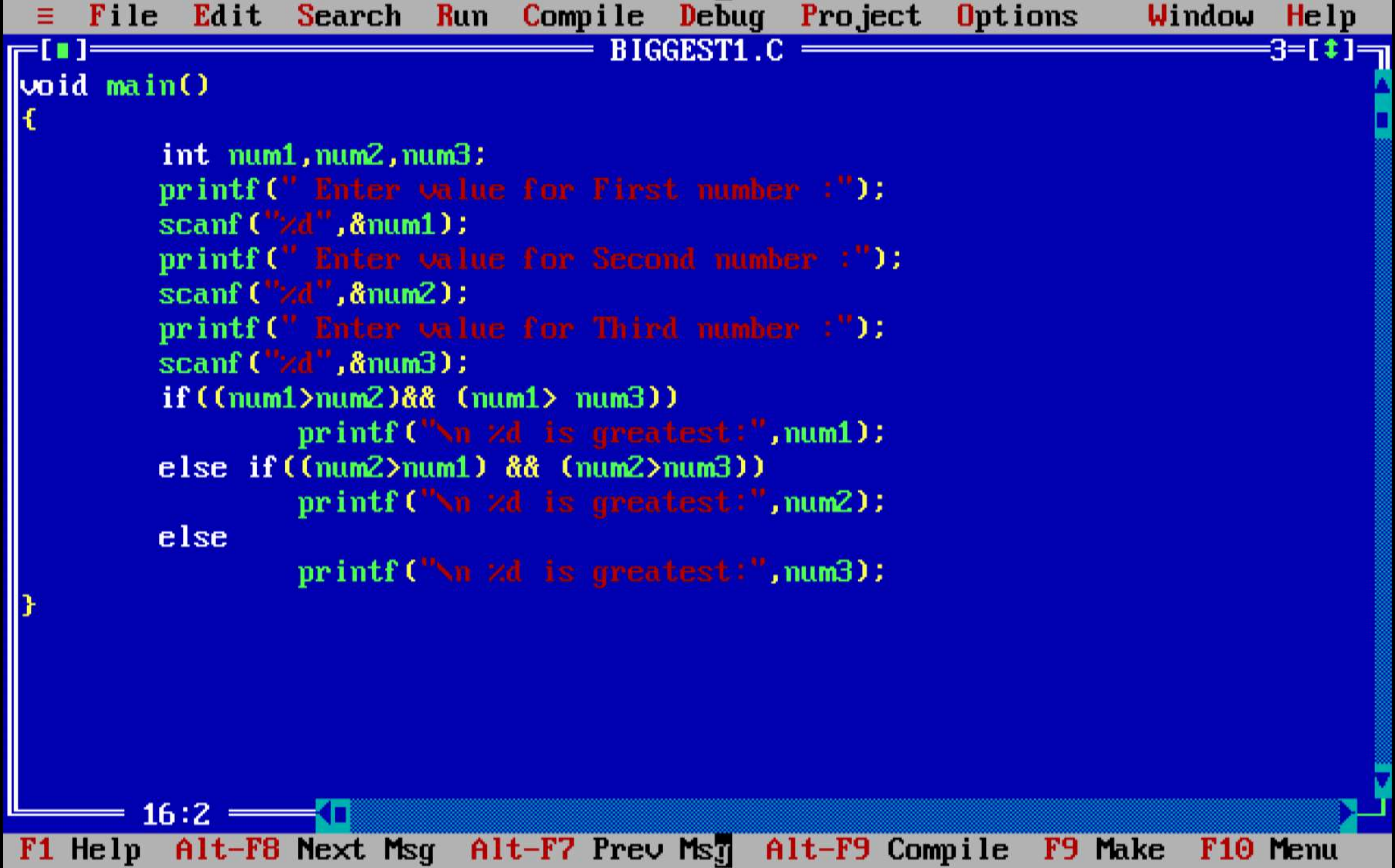
```
Enter the Age :10
You are not eligible to Vote
```



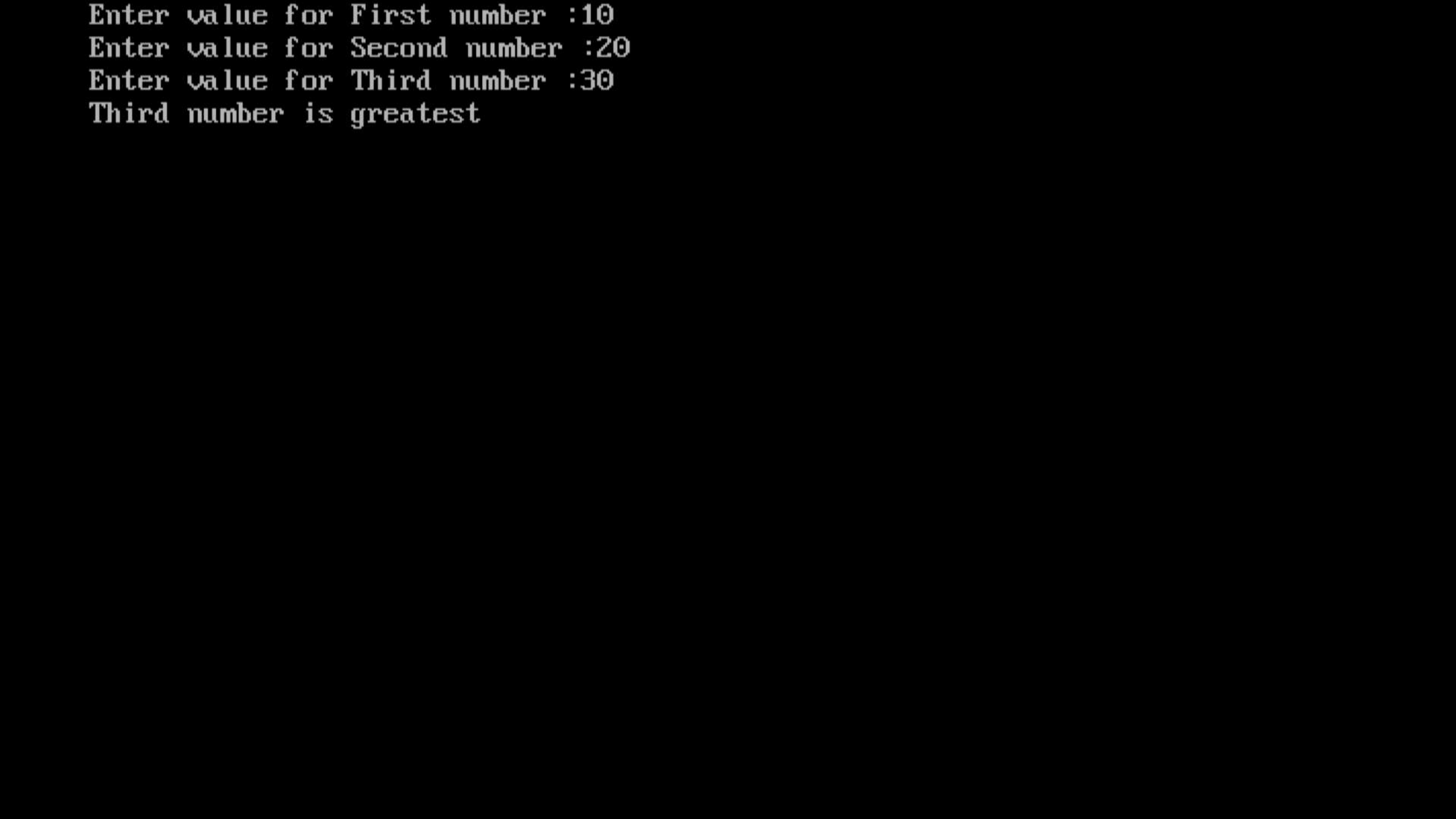
```
Enter the Age :20
You are eligible to Vote
```



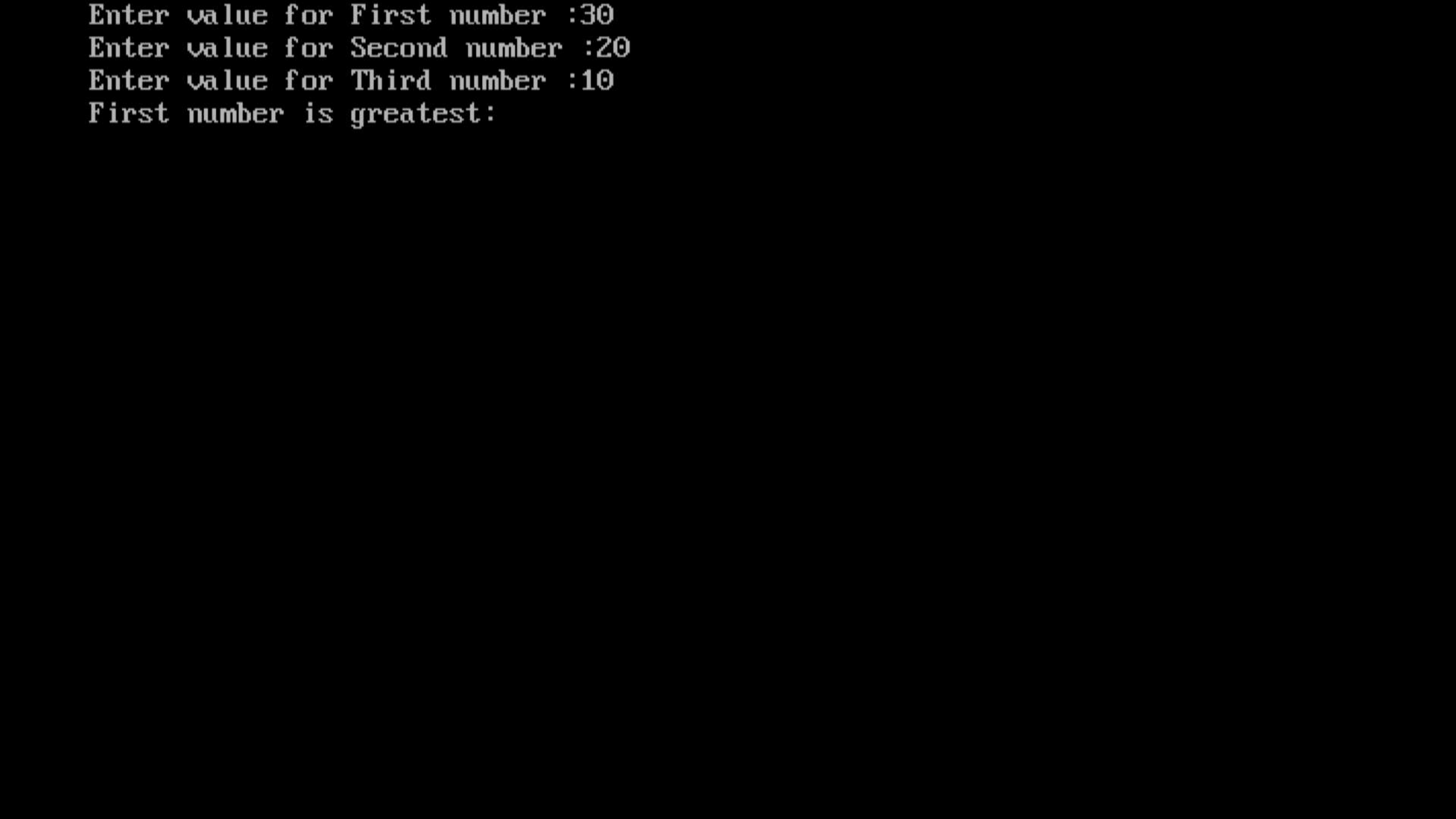
```
Enter the Age :125
Invalid age_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] BIGGEST1.C 3=[↕]
16:2
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

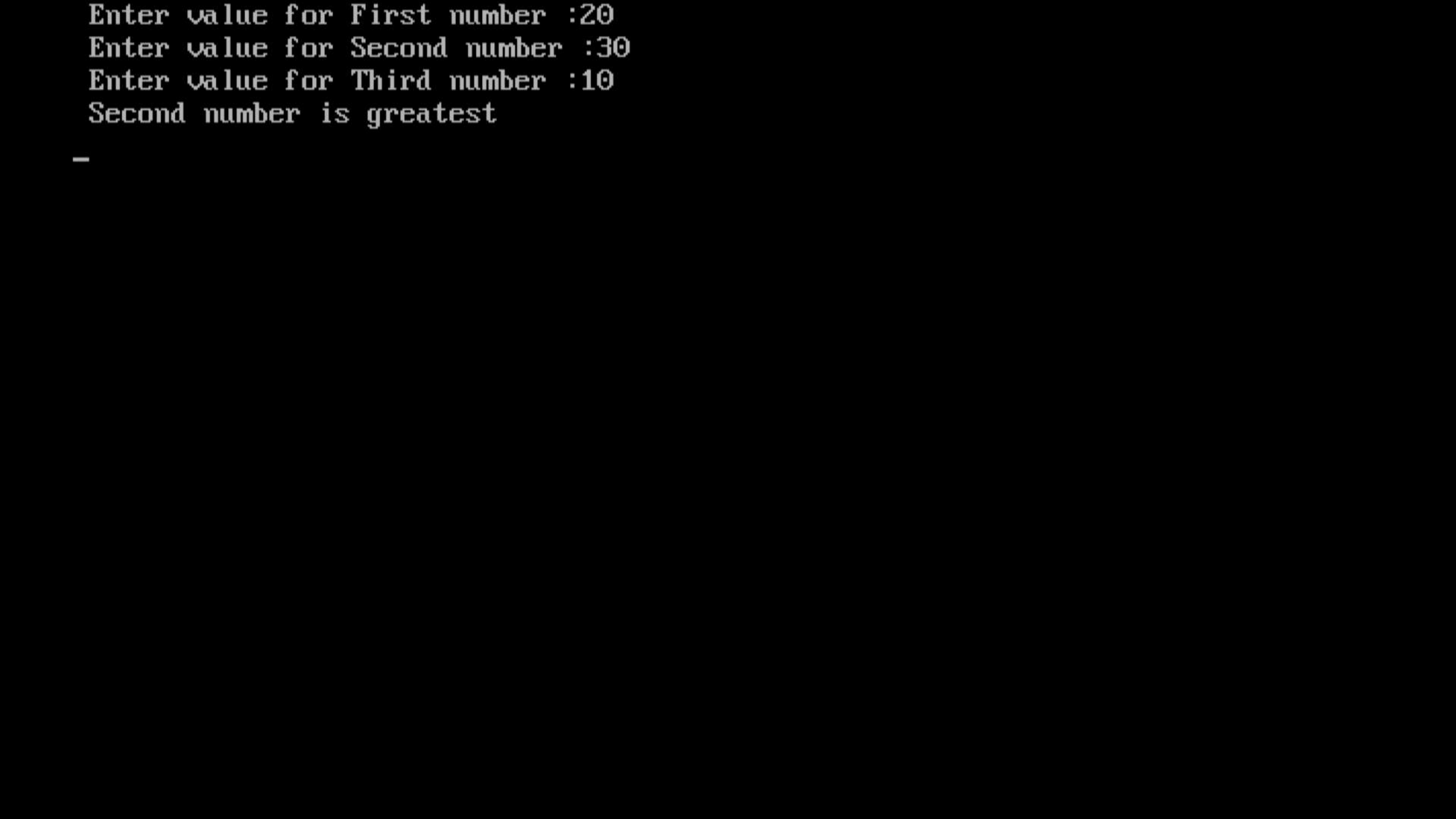
```
void main()
{
        int num1,num2,num3;
        printf(" Enter value for First number :");
        scanf("%d",&num1);
        printf(" Enter value for Second number :");
        scanf("%d",&num2);
        printf(" Enter value for Third number :");
        scanf("%d",&num3);
        if((num1>num2)&& (num1> num3))
                printf("\n %d is greatest:",num1);
        else if((num2>num1) && (num2>num3))
                printf("\n %d is greatest:",num2);
        else
                printf("\n %d is greatest:",num3);
}
```



```
Enter value for First number :10
Enter value for Second number :20
Enter value for Third number :30
Third number is greatest
```



```
Enter value for First number :30
Enter value for Second number :20
Enter value for Third number :10
First number is greatest:
```



```
Enter value for First number :20
Enter value for Second number :30
Enter value for Third number :10
Second number is greatest

_
```

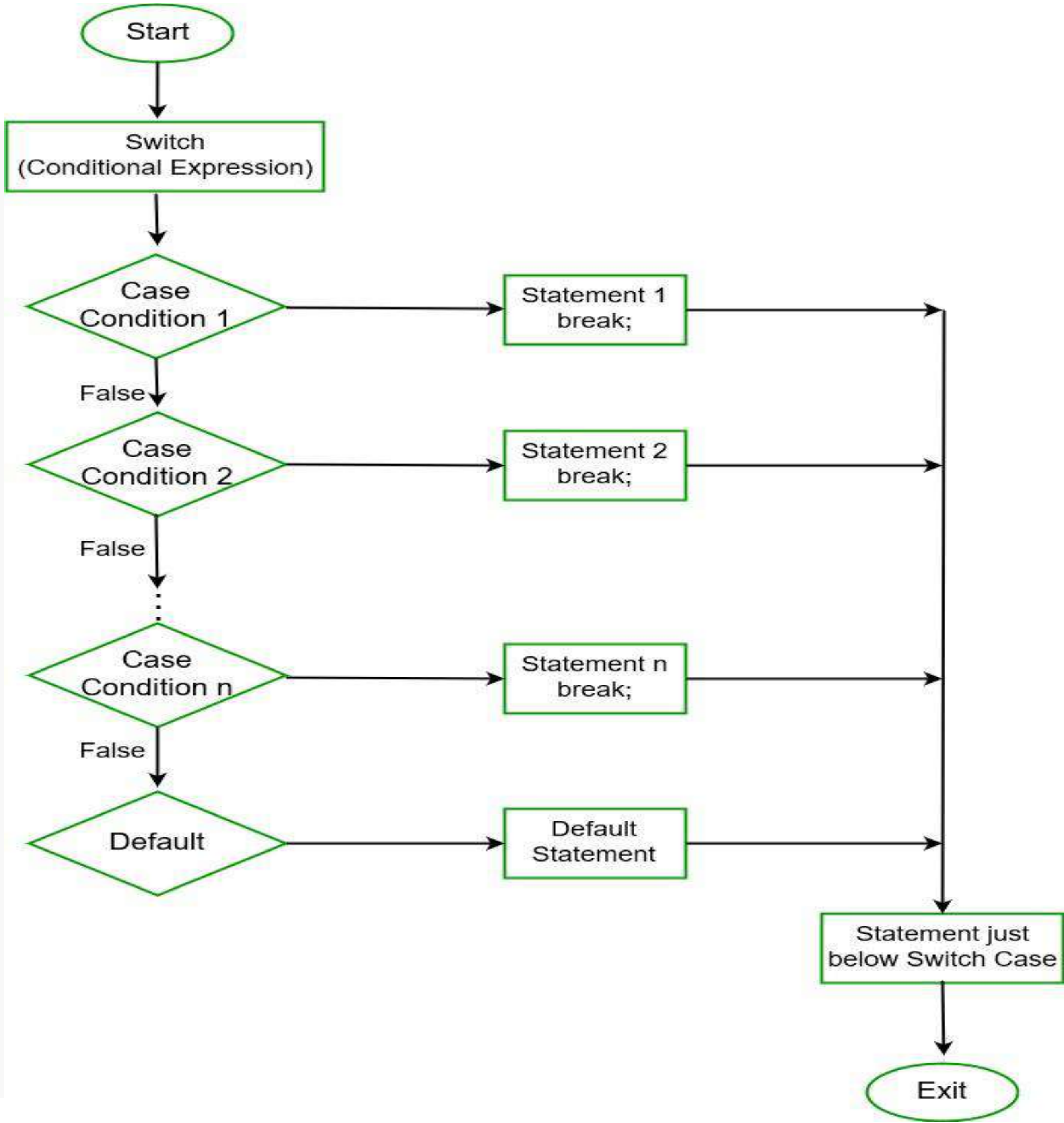
Switch Statement

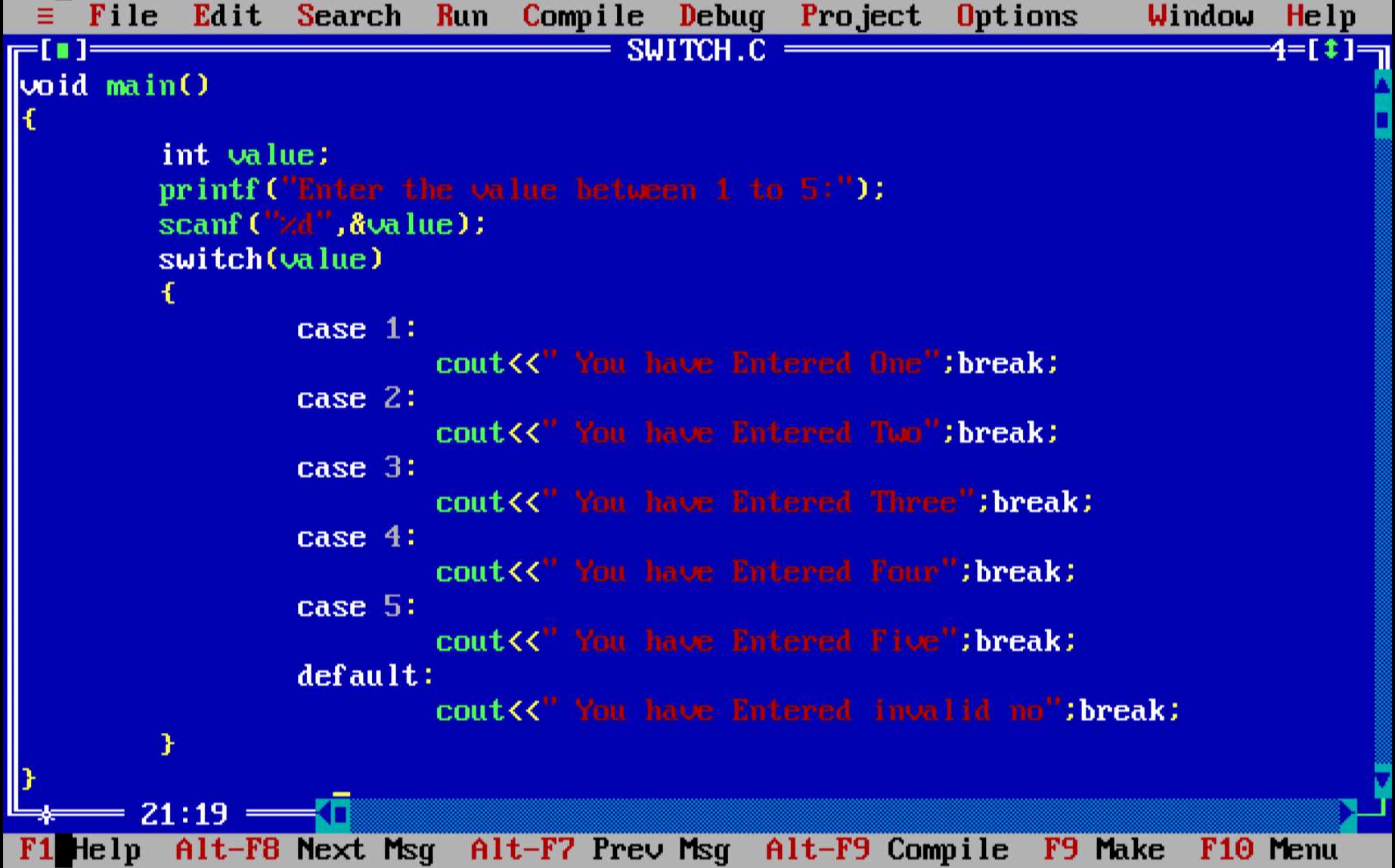
- In C, Switch statement is a multiway branch statement.
- It provides an efficient way to transfer the execution to different parts of a code based on the value of the expression.
- The switch expression is of integer type such as int, char, or short, or of an enumeration type, or of string type.
- The expression is checked for different cases and the one match is executed.
- The switch statement is often used as an alternative to an if-else construct if a single expression is tested against three or more conditions.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, then it will raise a compile time error.

- In C, duplicate case values are not allowed.
- The data type of the variable in the switch and value of a case must be of the same type.
- The value of a case must be a constant or a literal. Variables are not allowed.
- The break in switch statement is used to terminate the current sequence.
- The default statement is optional and it can be used anywhere inside the switch statement.
- Multiple default statements are not allowed.

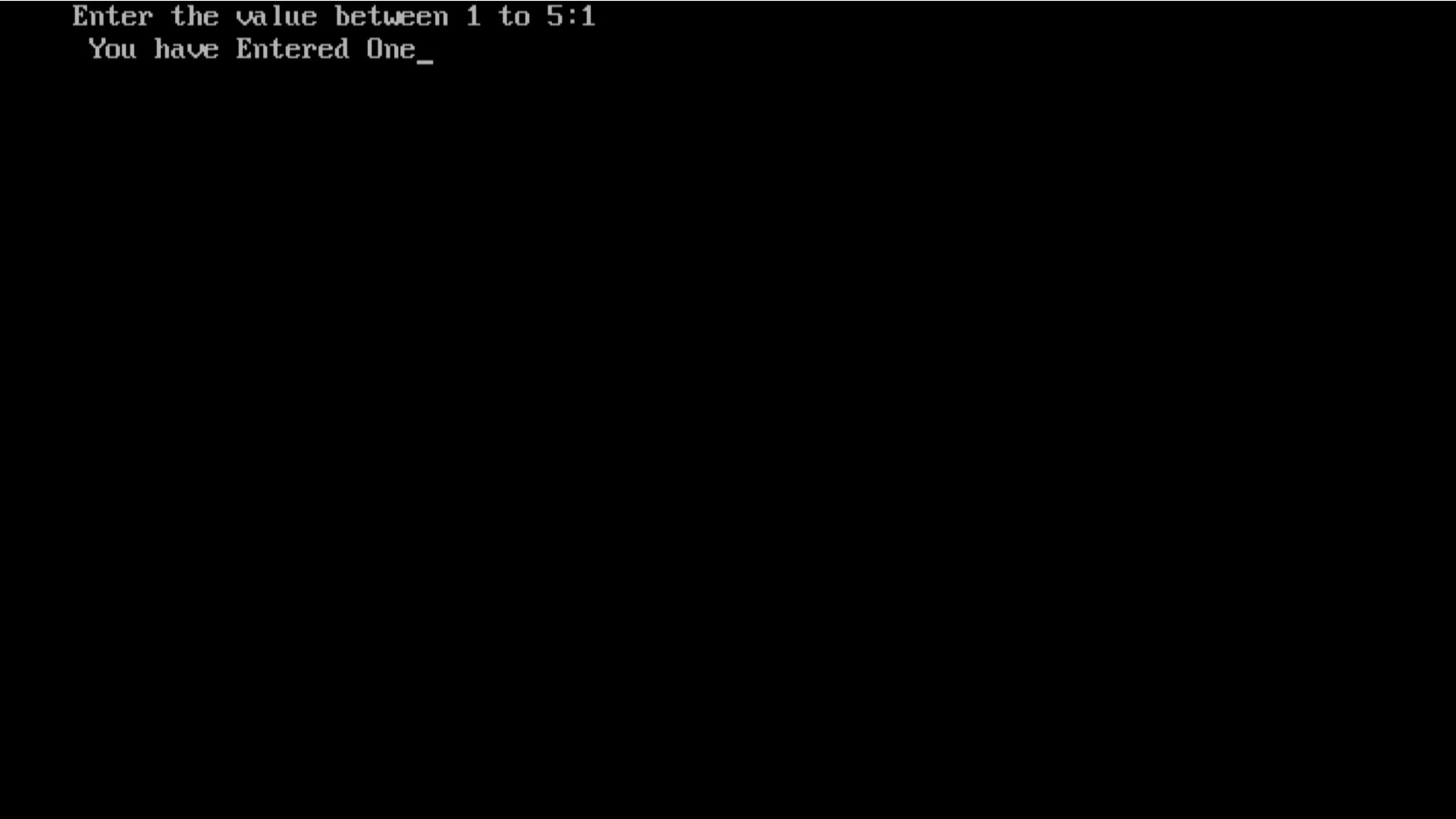
Syntax:

```
switch (expression) {

case value1: // statement sequence
     break;

case value2: // statement sequence
     break;
.
.
.
case valueN: // statement sequence
     break;

default:    // default statement sequence
}
```

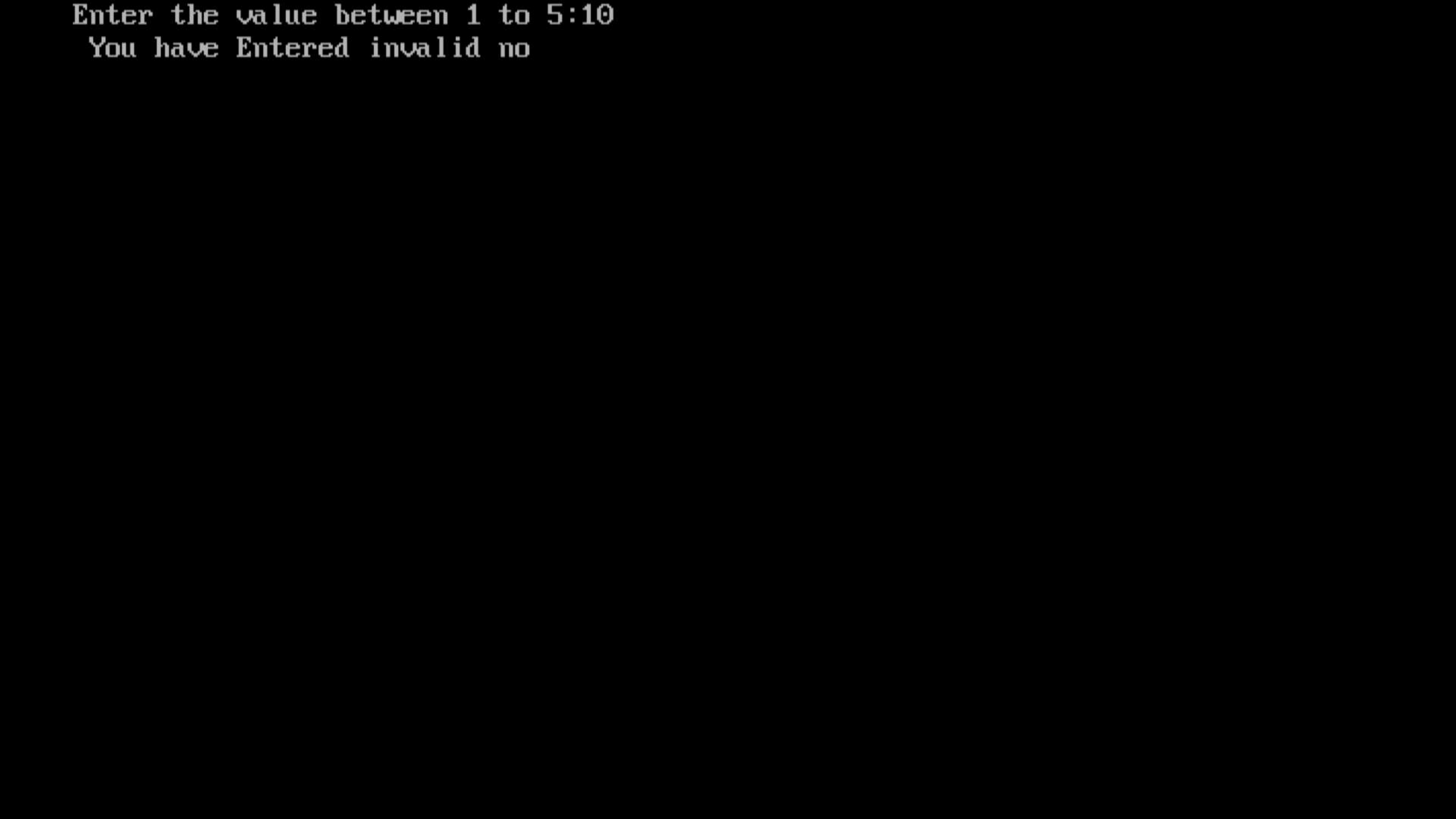



≡ File Edit Search Run Compile Debug Project Options Window Help
[■] SWITCH.C 4=[↕]
21:19
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

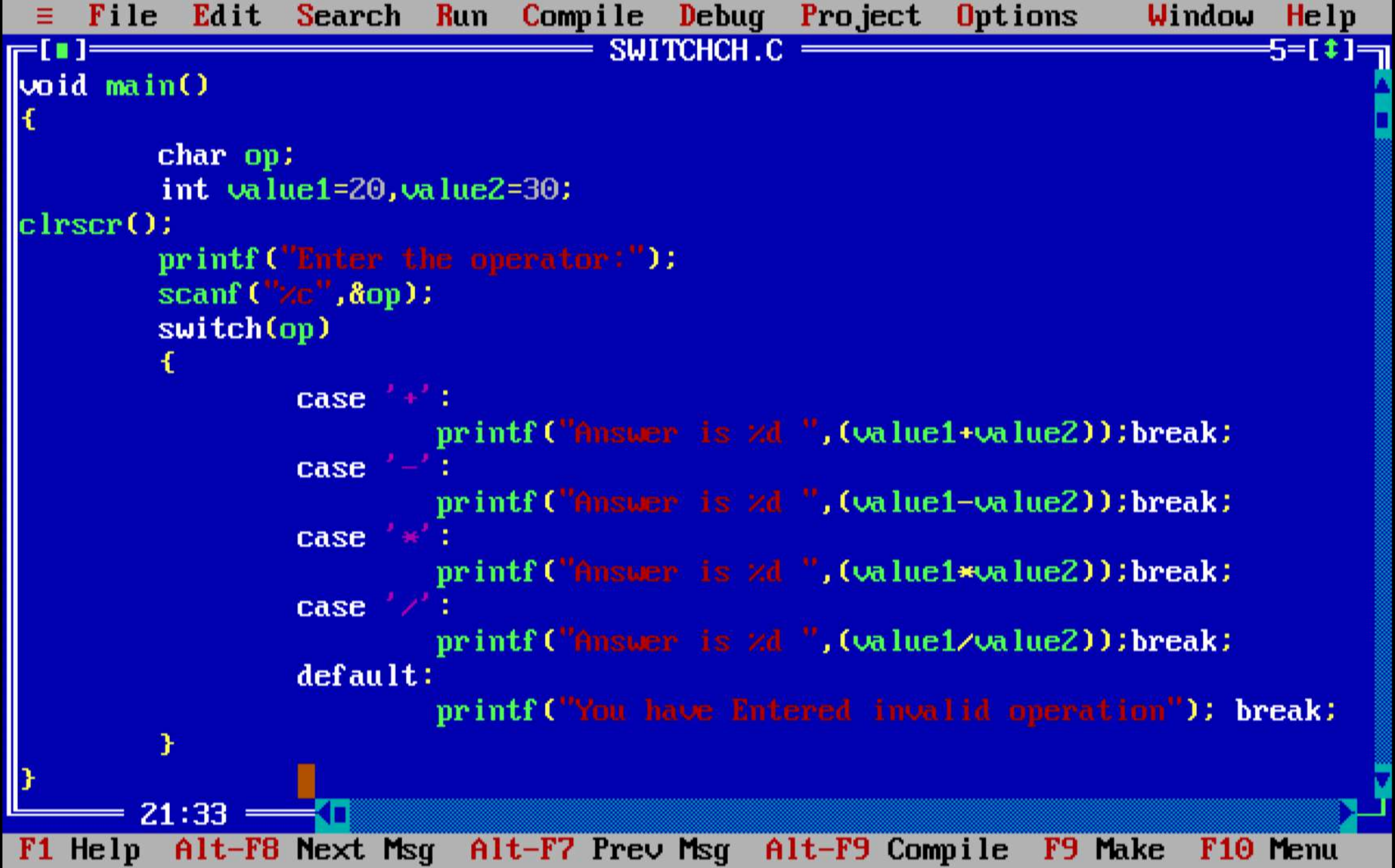
```
void main()
{
        int value;
        printf("Enter the value between 1 to 5:");
        scanf("%d",&value);
        switch(value)
        {
                case 1:
                        cout<<" You have Entered One";break;
                case 2:
                        cout<<" You have Entered Two";break;
                case 3:
                        cout<<" You have Entered Three";break;
                case 4:
                        cout<<" You have Entered Four";break;
                case 5:
                        cout<<" You have Entered Five";break;
                default:
                        cout<<" You have Entered invalid no";break;
        }
}
```



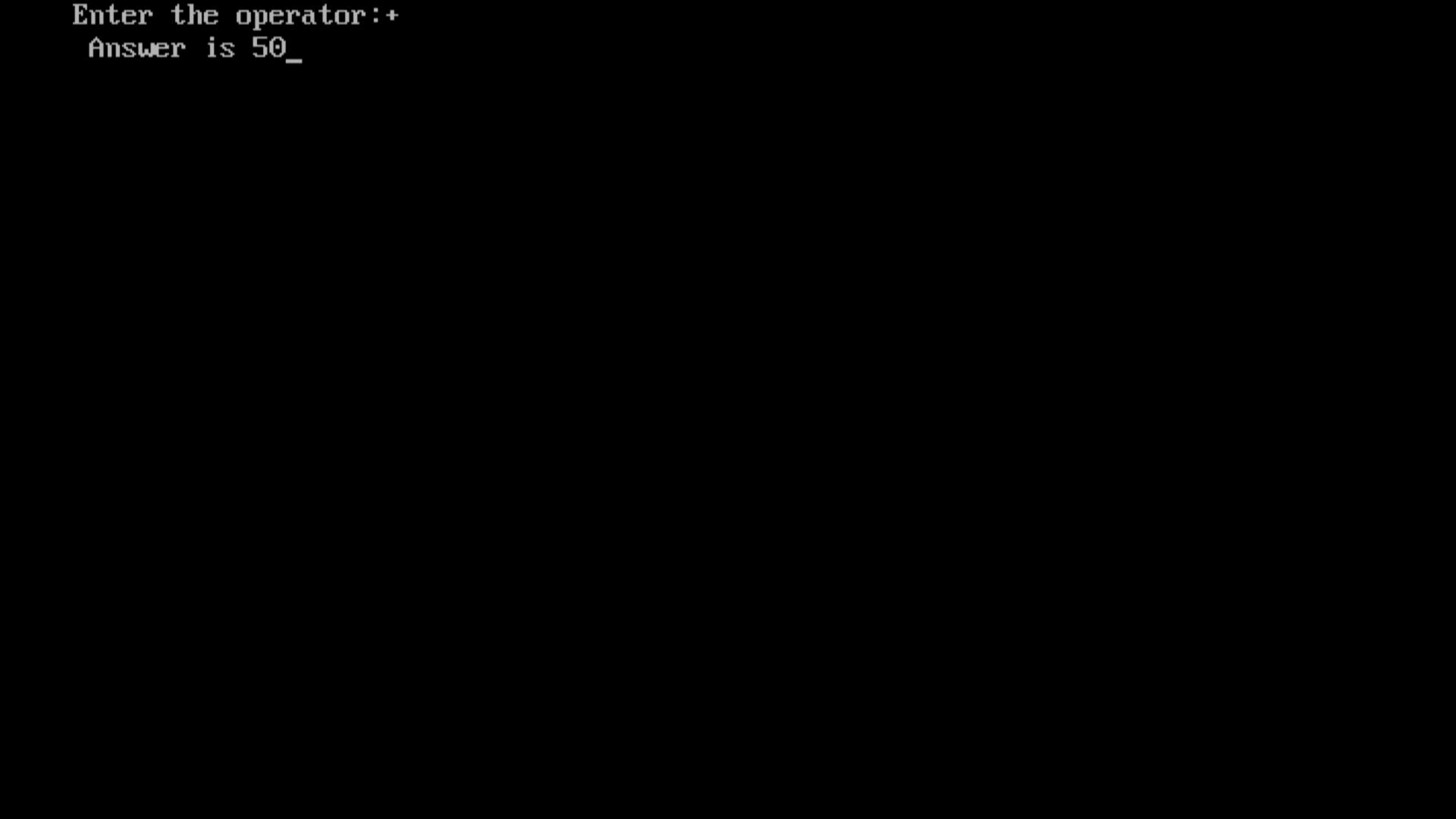
```
Enter the value between 1 to 5:1
 You have Entered One_
```



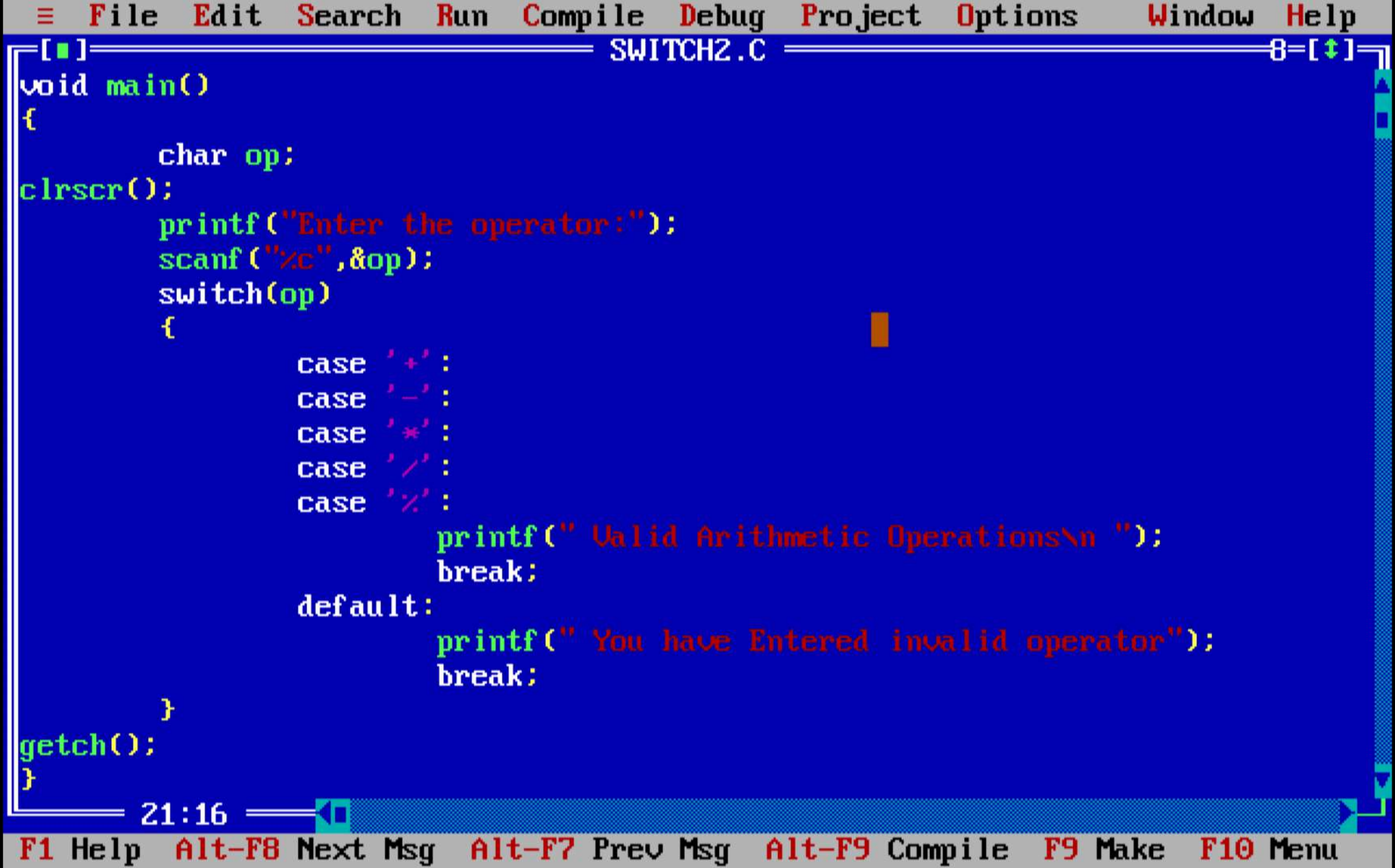
```
Enter the value between 1 to 5:10
 You have Entered invalid no
```


≡ File Edit Search Run Compile Debug Project Options Window Help
SWITCHCH.C
5
21:33
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

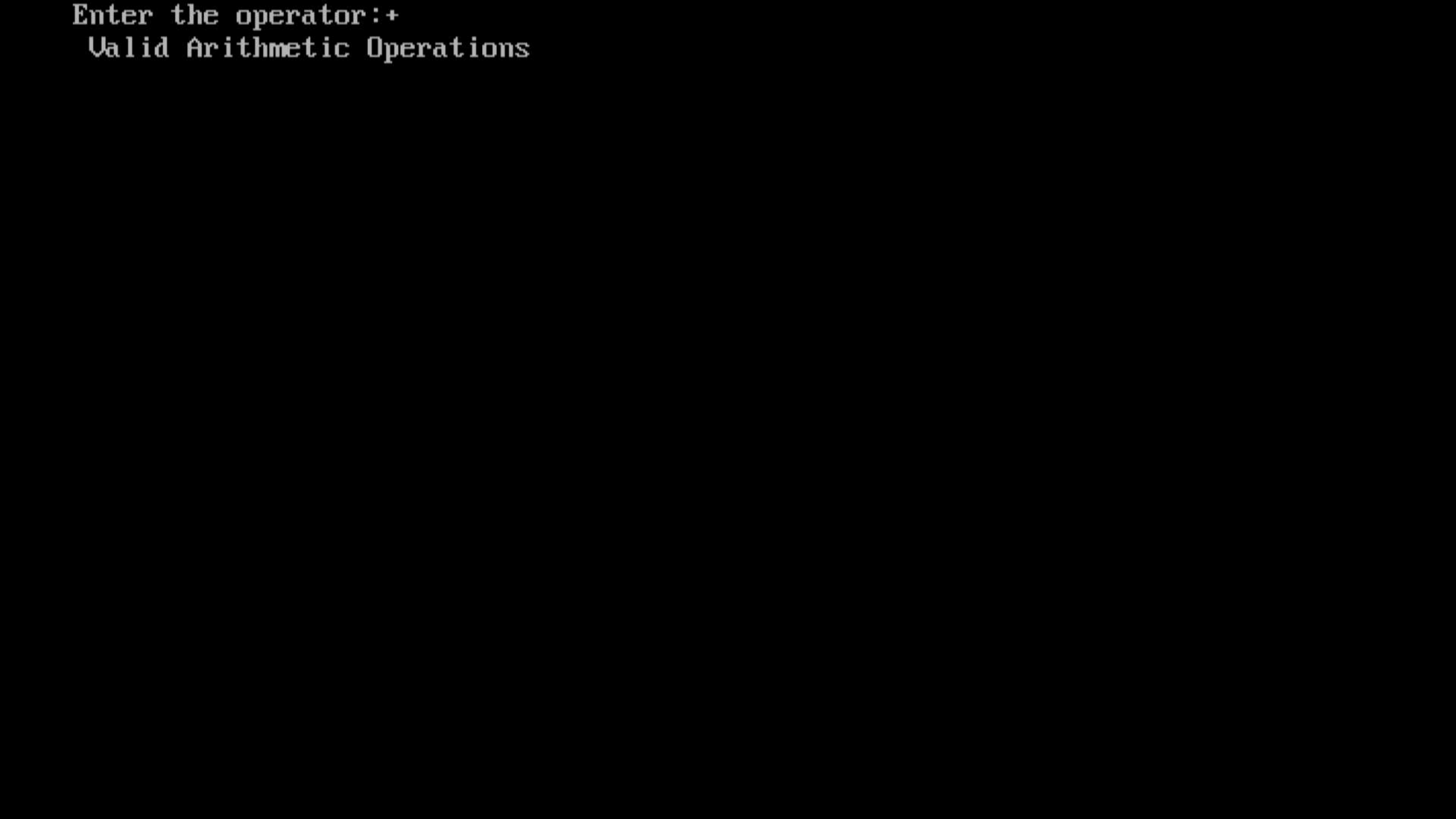
```
void main()
{
        char op;
        int value1=20,value2=30;
clrscr();
        printf("Enter the operator:");
        scanf("%c",&op);
        switch(op)
        {
                case '+':
                        printf("Answer is %d ",(value1+value2));break;
                case '-':
                        printf("Answer is %d ",(value1-value2));break;
                case '*':
                        printf("Answer is %d ",(value1*value2));break;
                case '/':
                        printf("Answer is %d ",(value1/value2));break;
                default:
                        printf("You have Entered invalid operation"); break;
        }
}
```



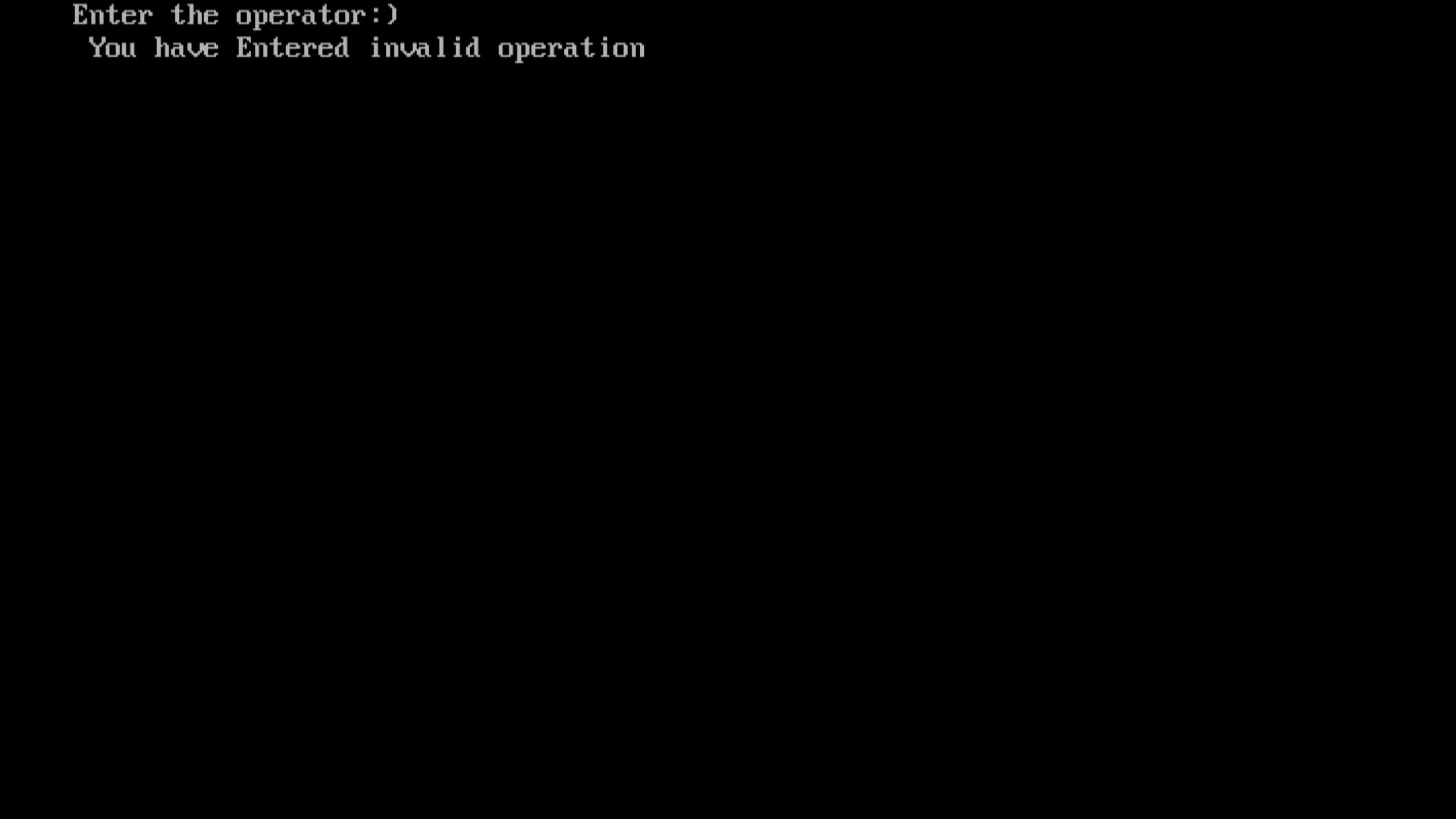
```
Enter the operator:+
 Answer is 50_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
SWITCH2.C
8
21:16
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        char op;
clrscr();
        printf("Enter the operator:");
        scanf("%c",&op);
        switch(op)
        {
                case '+':
                case '-':
                case '*':
                case '/':
                case '%':
                        printf(" Valid Arithmetic Operations\n ");
                        break;
                default:
                        printf(" You have Entered invalid operator");
                        break;
        }
getch();
}
```



```
Enter the operator:+
 Valid Arithmetic Operations
```



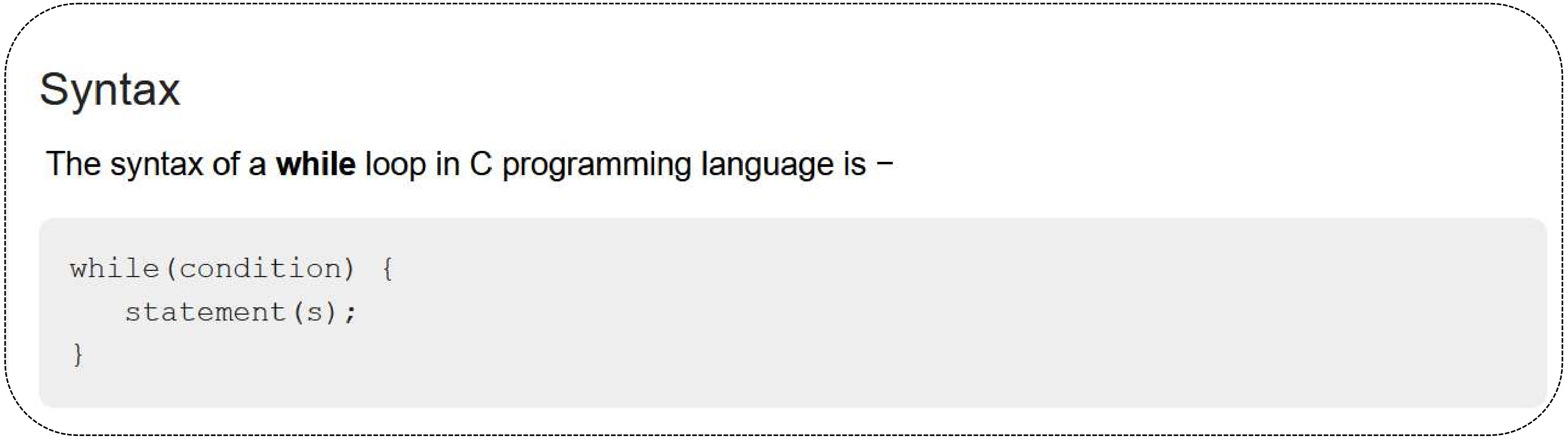
```
Enter the operator:)
 You have Entered invalid operation
```

Iteration Statements

- Iteration statements or Loops are used in programming to repeatedly execute a certain block of statements until some condition is met.
- The following statements repeatedly execute a statement or a block of statements:
  - The for statement: executes its body while a specified Boolean expression evaluates to true.
  - The do statement: conditionally executes its body one or more times.
  - The while statement: conditionally executes its body zero or more times.
- At any point within the body of an iteration statement, you can break out of the loop by using the break statement, or step to the next iteration in the loop by using the continue statement.

while loop

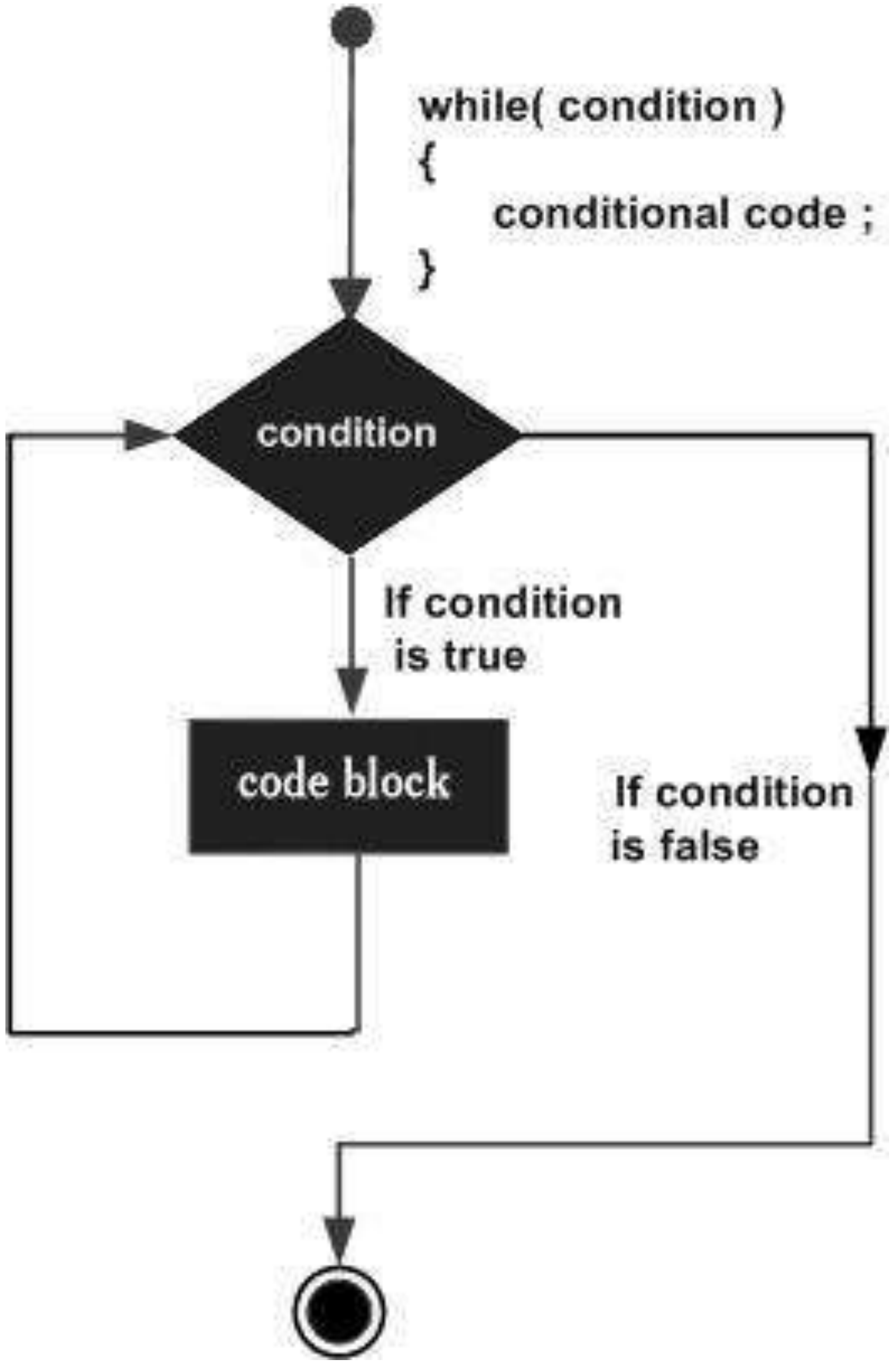
- C provides the while loop to repeatedly execute a block of code as long as the specified condition returns false.
- The while loop starts with the while keyword, and it must include a conditional expression inside brackets that returns either true or false.
- It executes the code block until the specified conditional expression returns 0.
- In a while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the loop.
- The statement(s) inside the while loop may be a single statement or a block of statements.
- The key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

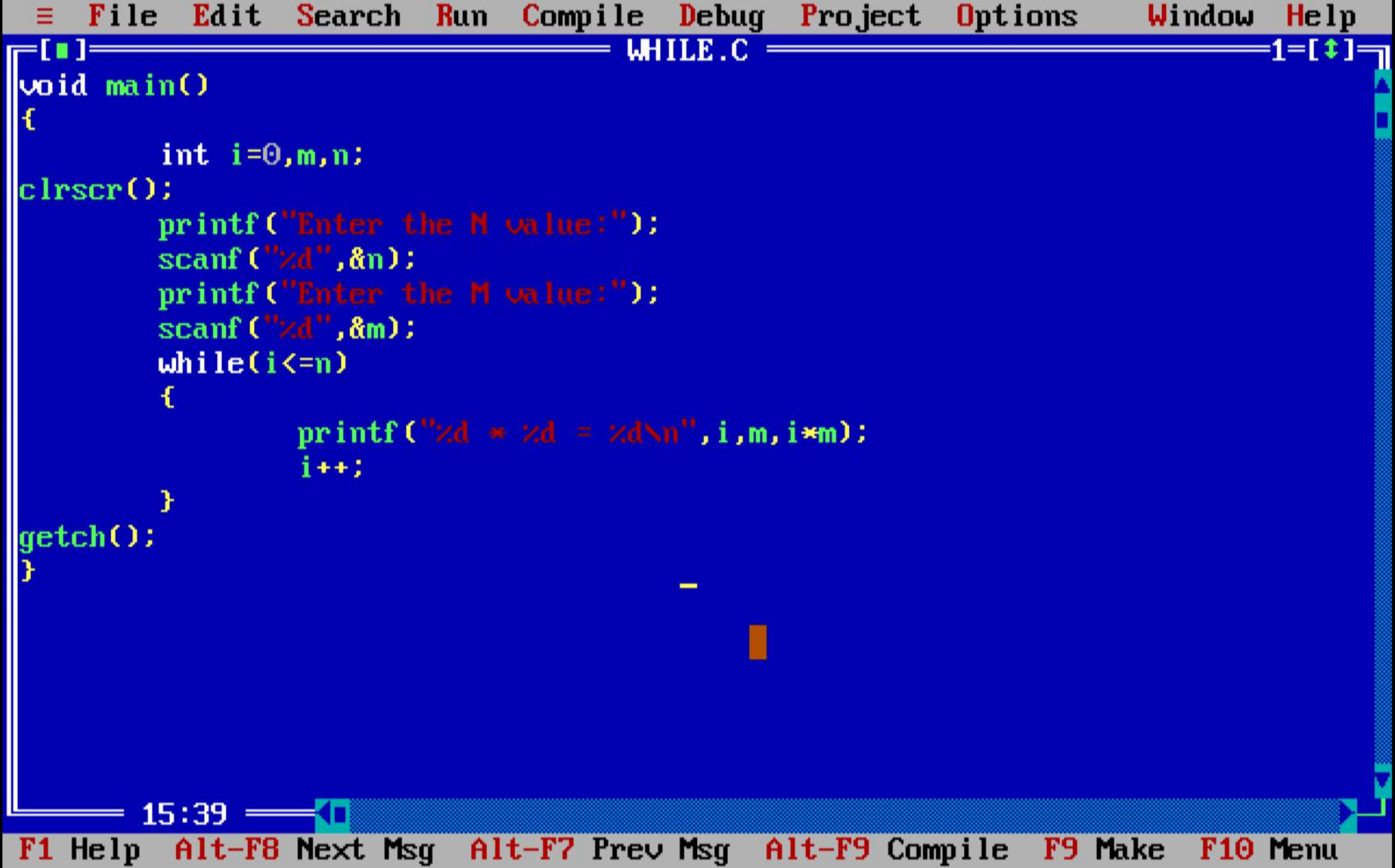


Syntax

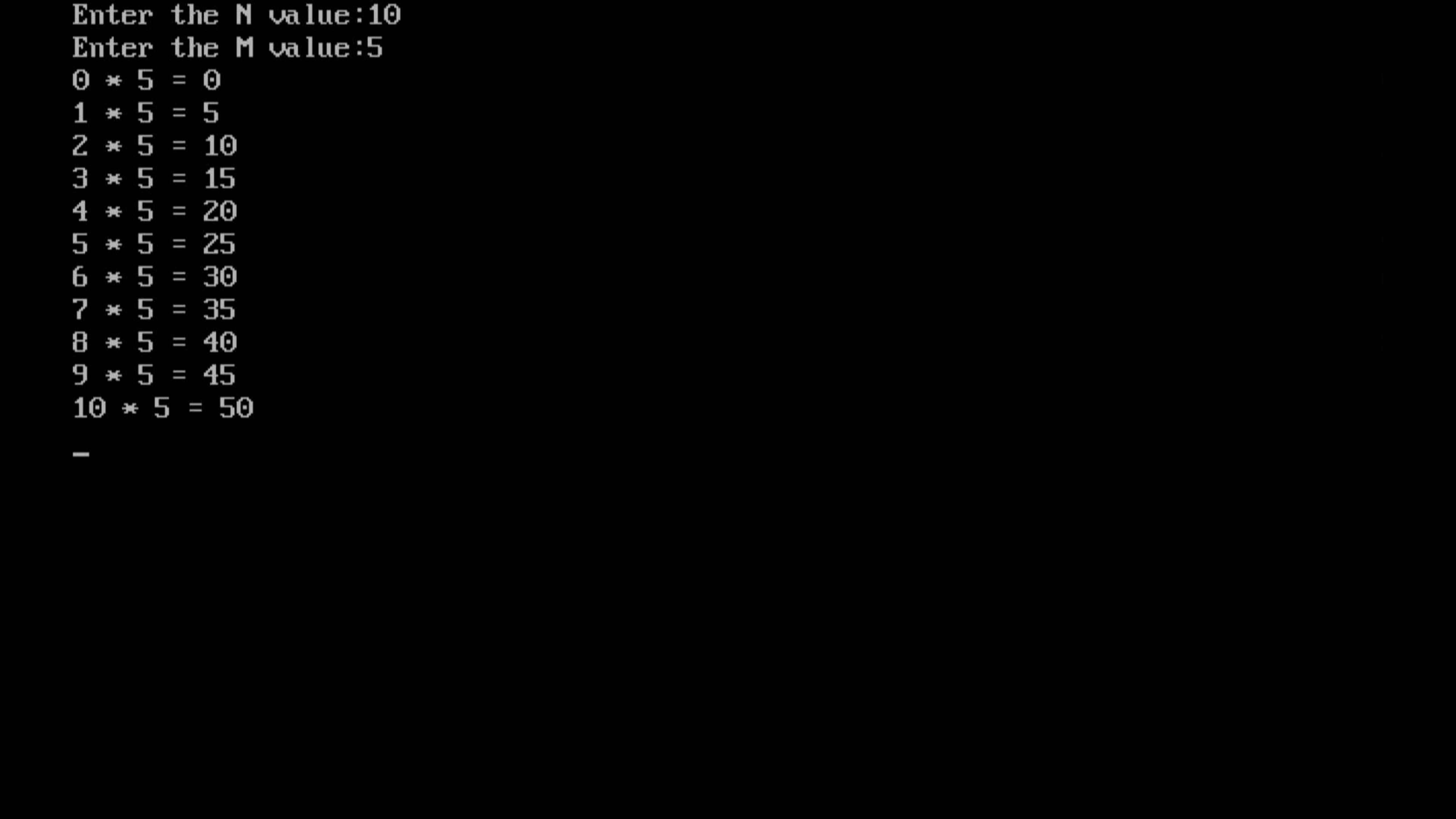
The syntax of a **while** loop in C programming language is −

```
while(condition) {
   statement(s);
}
```


while(condition)
{
conditional code ;
}
condition
If condition is true
code block
If condition is false


≡ File Edit Search Run Compile Debug Project Options Window Help
WHILE.C
15:39
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

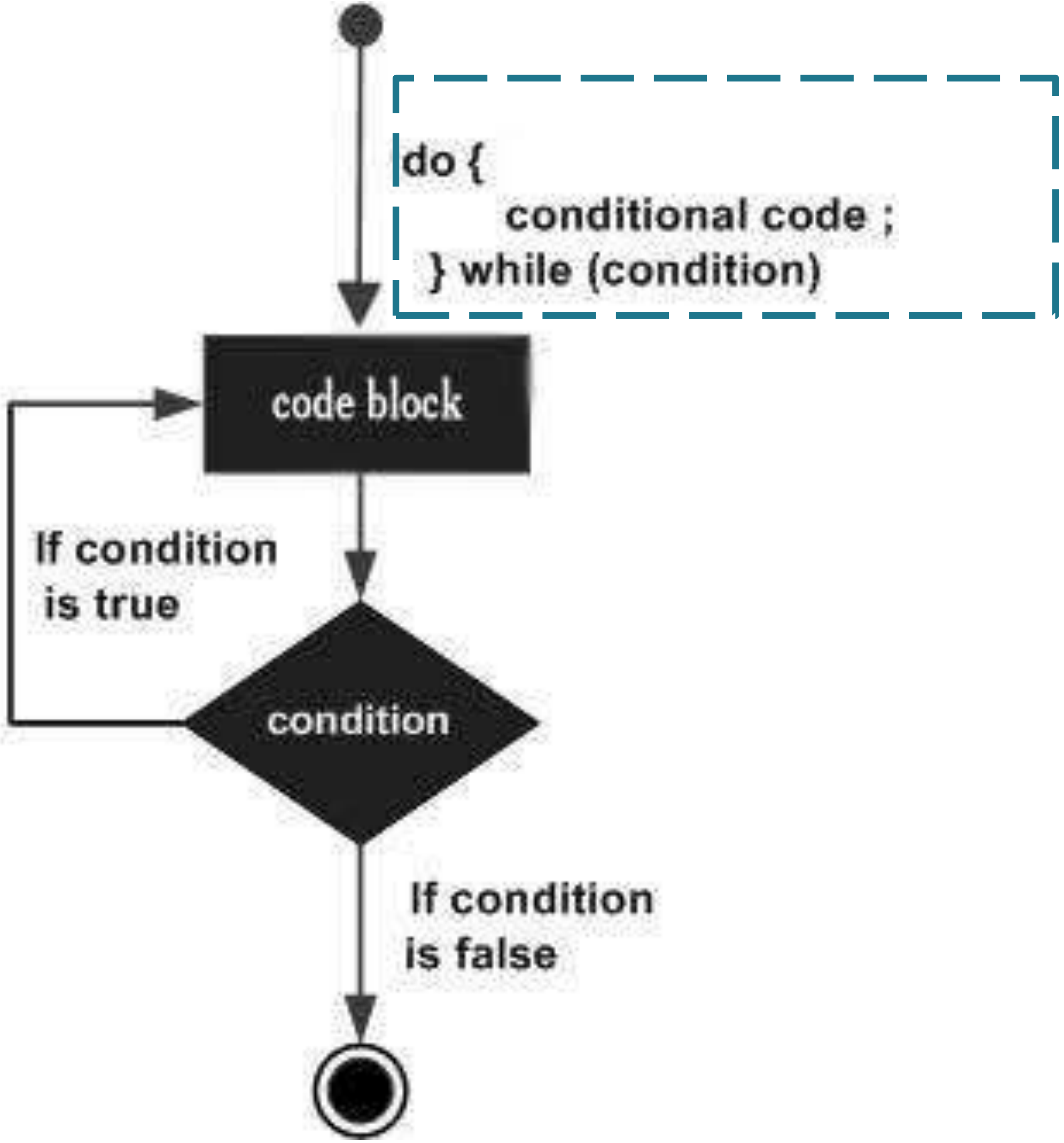
```
void main()
{
        int i=0,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        while(i<=n)
        {
                printf("%d * %d = %d\n",i,m,i*m);
                i++;
        }
getch();
}
```



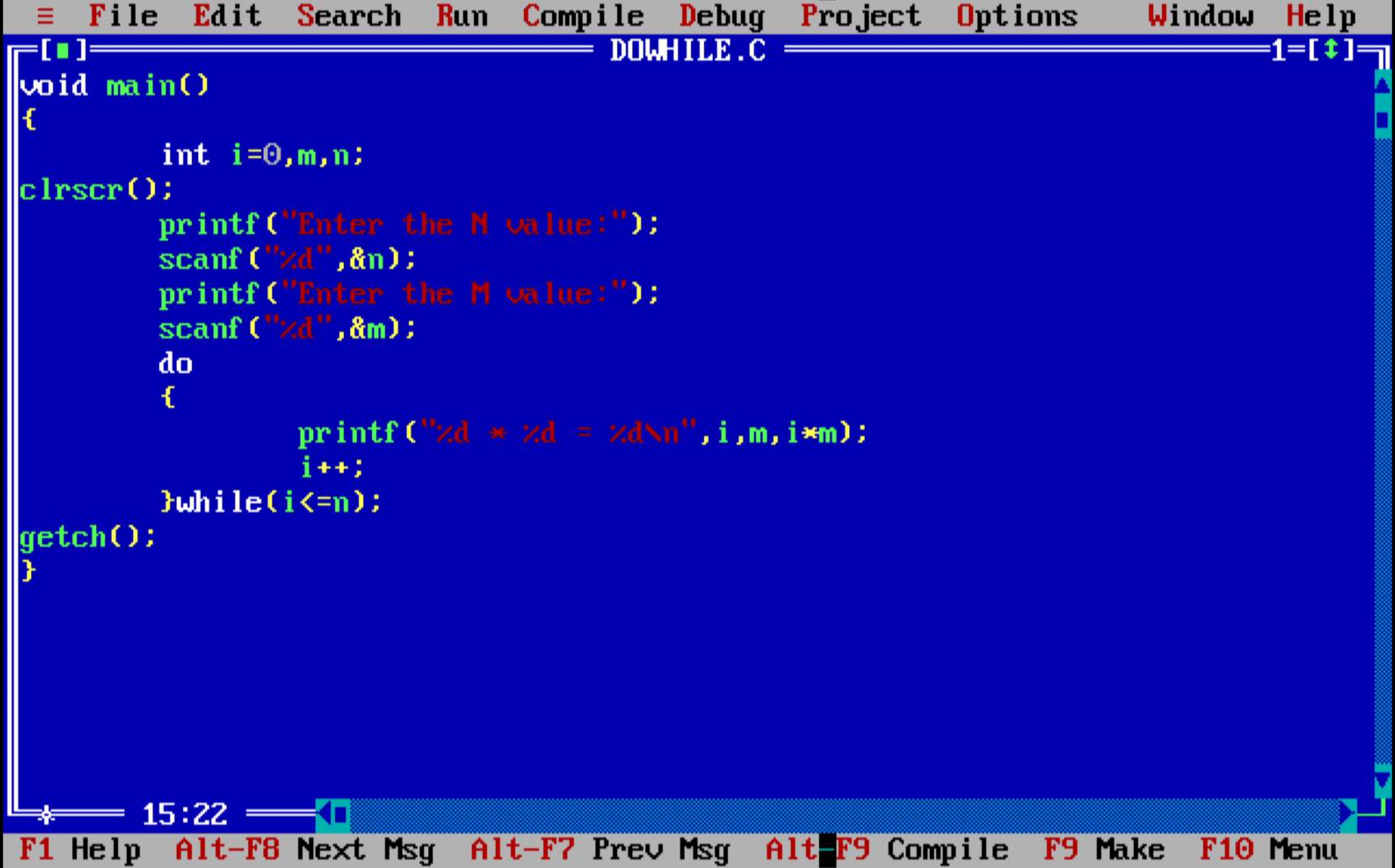
```
Enter the N value:10
Enter the M value:5
0 * 5 = 0
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50

_
```

do while Statement

- The do while loop is the same as while loop except that it executes the code block at least once.
- The do-while loop starts with the do keyword followed by a code block and a conditional expression with the while keyword.
- The do while loop stops execution exits when a condition evaluates to false. Because the while(condition) specified at the end of the block, it certainly executes the code block at least once.
- In a while loop, initialization should be done before the loop starts, and increment or decrement steps should be inside the loop.
- The statement(s) inside the while loop may be a single statement or a block of statements.
- The key point of the do while loop is that the loop will executed at least once, even on the first time When the condition is tested and the result is false.
- This is the reason, do while is called as exit controlled loop.

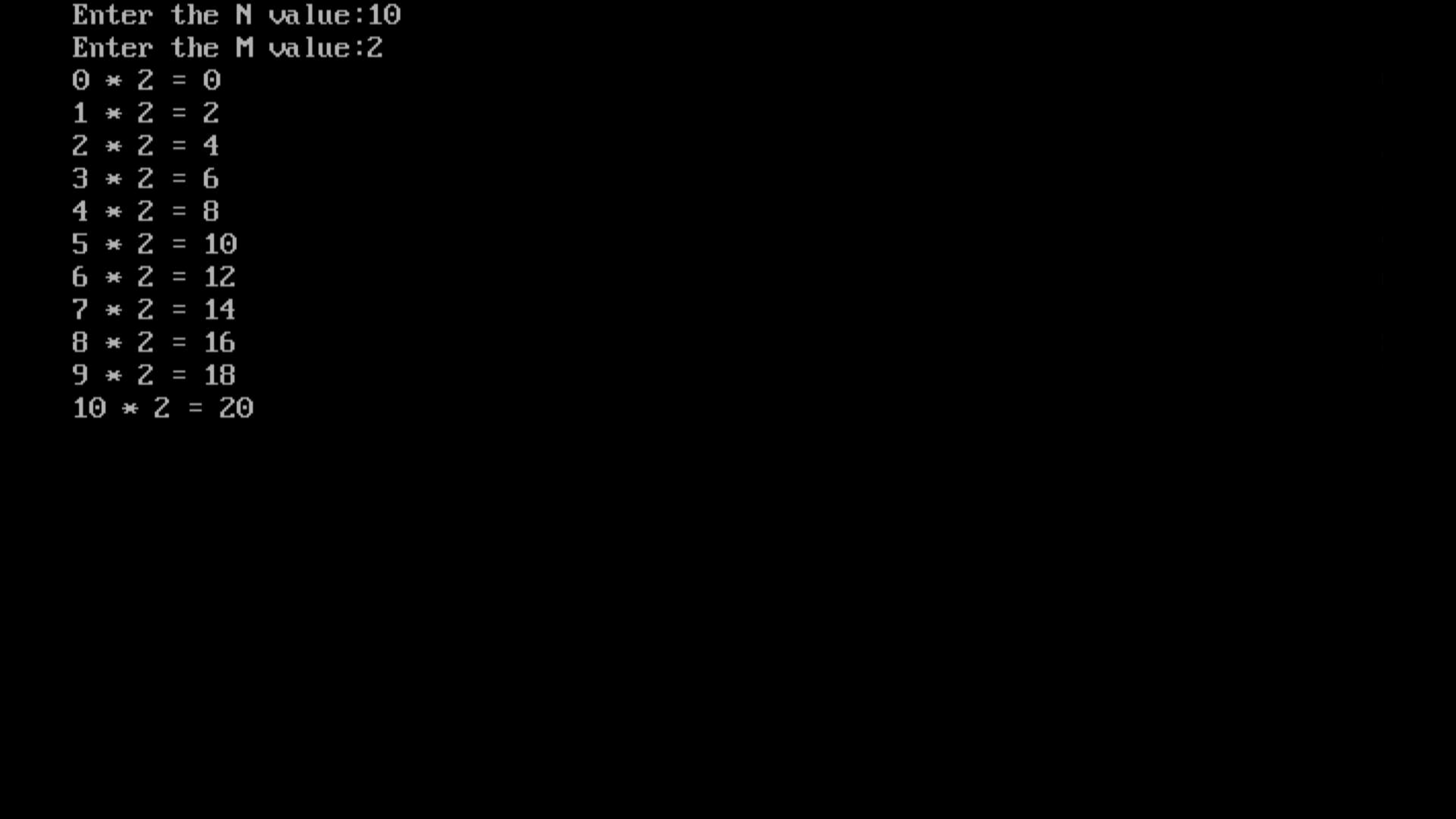


```
do {
        conditional code ;
 } while (condition)
```


≡ File Edit Search Run Compile Debug Project Options Window Help
DOWHILE.C
1
15:22
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int i=0,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        do
        {
                printf("%d * %d = %d\n",i,m,i*m);
                i++;
        }while(i<=n);
getch();
}
```

```
Enter the N value:0
Enter the M value:5
0 * 5 = 0
```

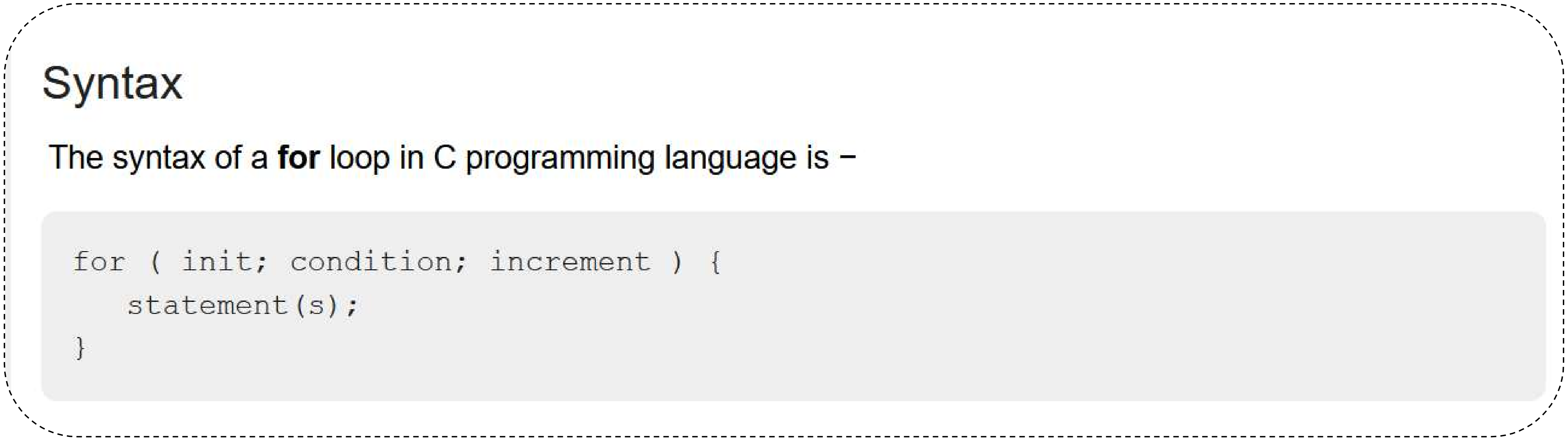


```
Enter the N value:10
Enter the M value:2
0 * 2 = 0
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14
8 * 2 = 16
9 * 2 = 18
10 * 2 = 20
```

for loop

- A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
  - for loop has three statements: initialization, condition and iterator.
  - The initialization statement is executed at first and only once. Here, the variable is usually declared and initialized.
  - Then, the condition is evaluated. The condition is a Boolean expression, i.e. it returns either true or false.
  - If the condition is evaluated to true:
    - The body of the loop, which must be a statement or a block of statements.
    - Then, the iterator statement is executed which usually changes the value of the initialized variable.
    - Again the condition is evaluated.
    - The process continues until the condition is evaluated to false.
  - If the condition is evaluated to false, the for loop terminates.

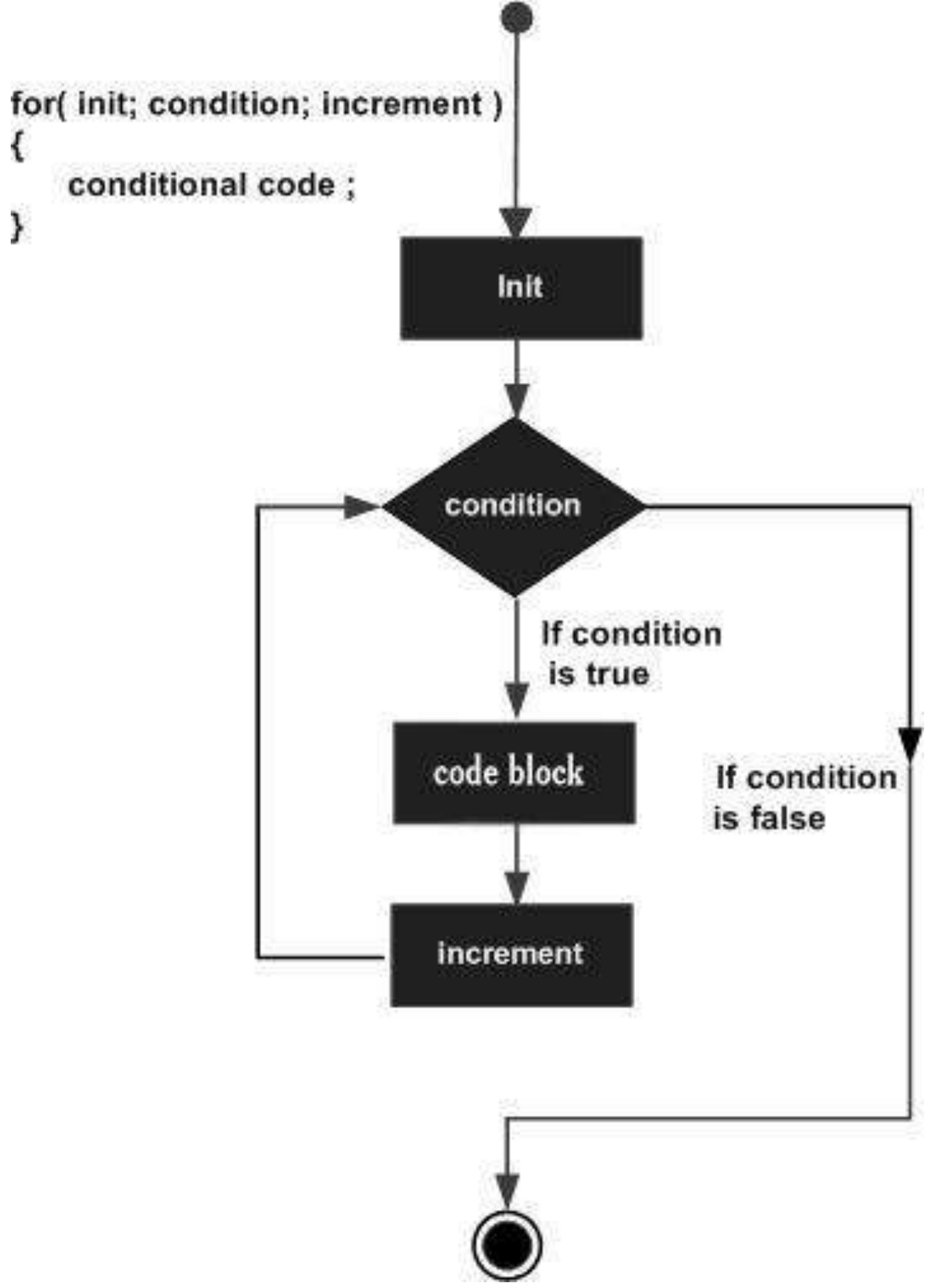
- The iterator section can contain zero or more of the following statement expressions, separated by commas:
  - prefix or postfix increment expression, such as ++i or i++
  - prefix or postfix decrement expression, such as --i or i--
  - assignment
  - invocation of a method
- All the sections of the for statement are optional.



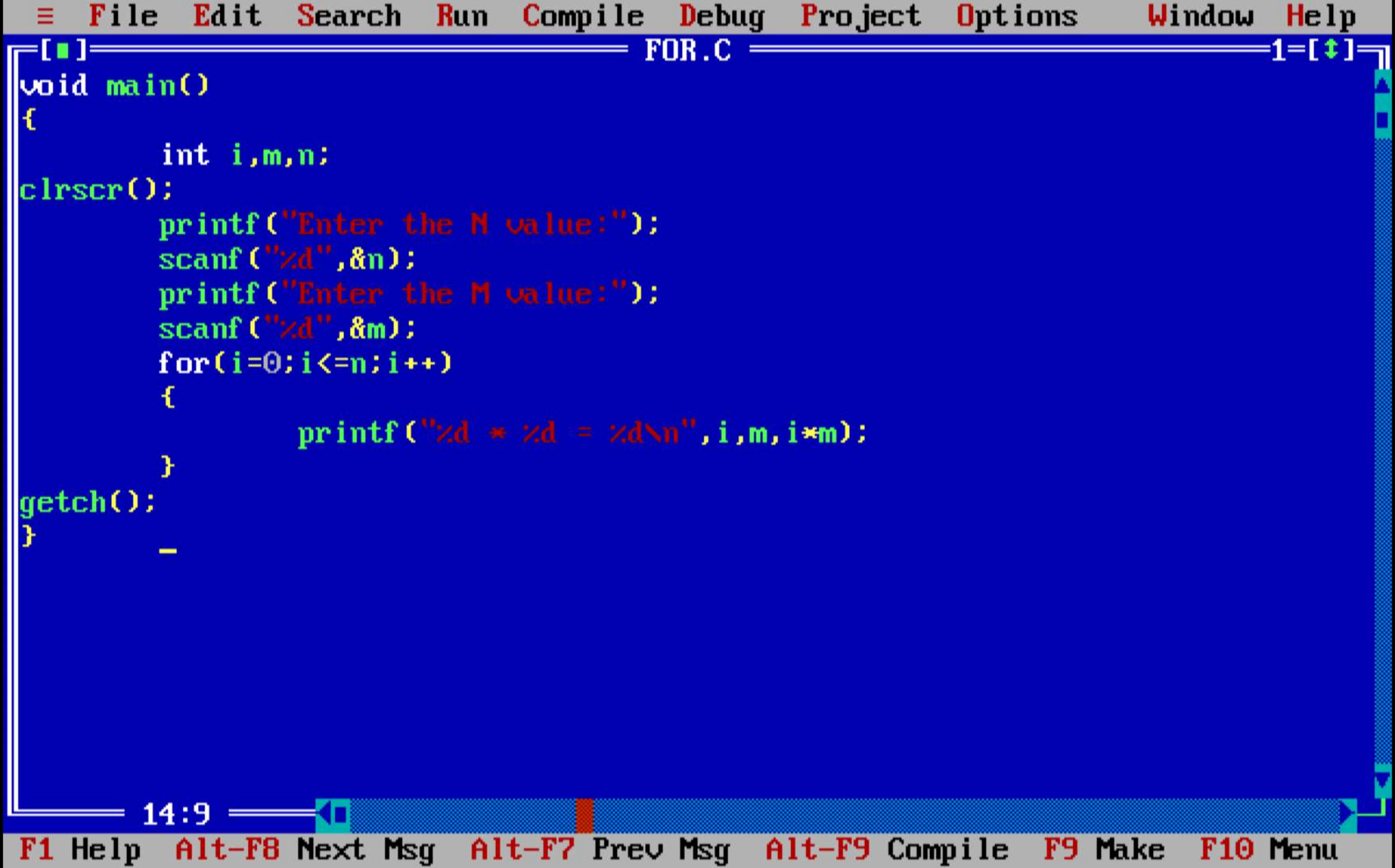
Syntax

The syntax of a **for** loop in C programming language is −

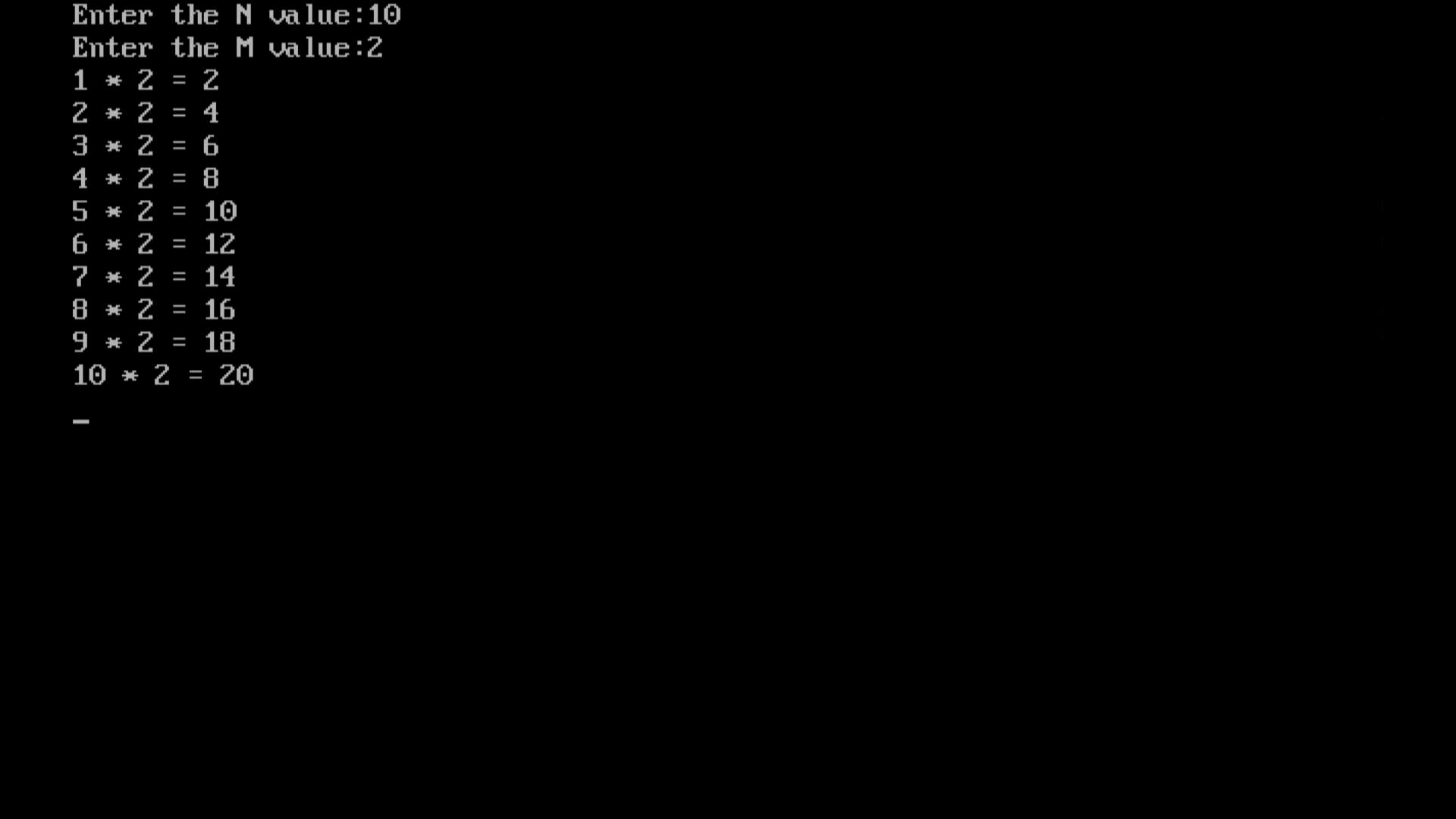
```
for ( init; condition; increment ) {
   statement(s);
}
```



```
for( init; condition; increment )
{
    conditional code ;
}
```


≡ File Edit Search Run Compile Debug Project Options Window Help
FOR.C
1
14:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int i,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        for(i=0;i<=n;i++)
        {
                printf("%d * %d = %d\n",i,m,i*m);
        }
getch();
}
```

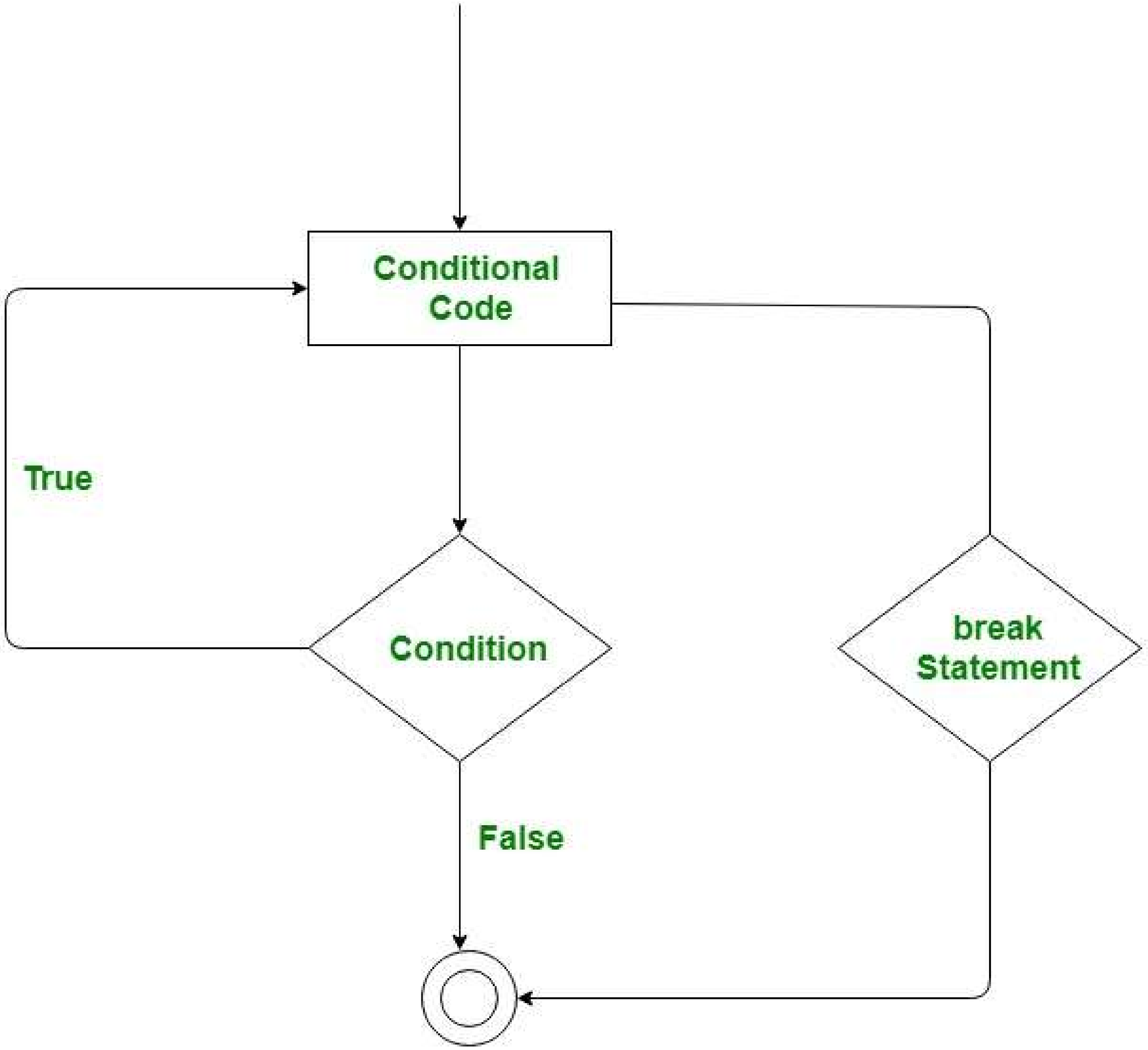


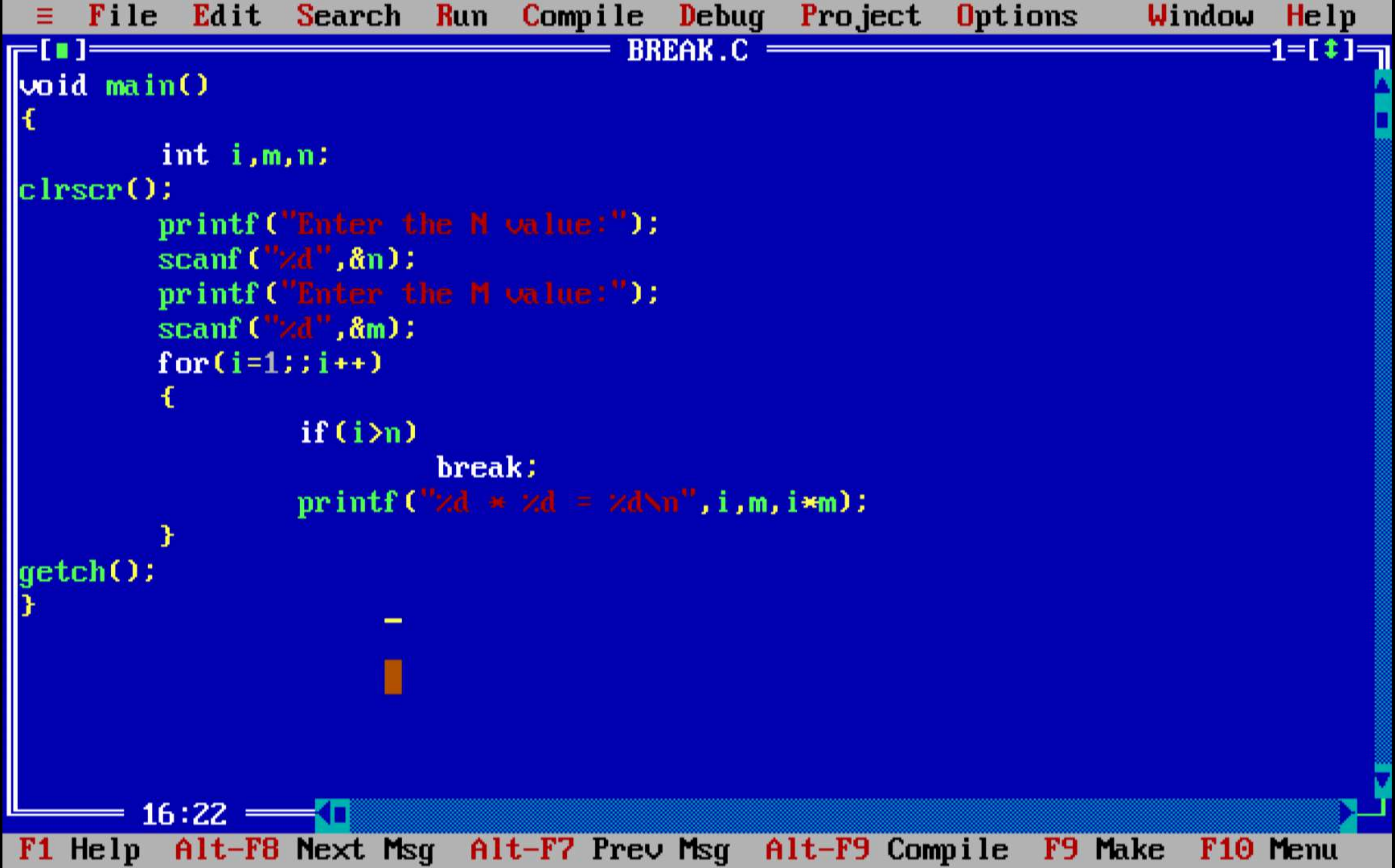
```
Enter the N value:10
Enter the M value:2
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14
8 * 2 = 16
9 * 2 = 18
10 * 2 = 20

_
```

break Statements

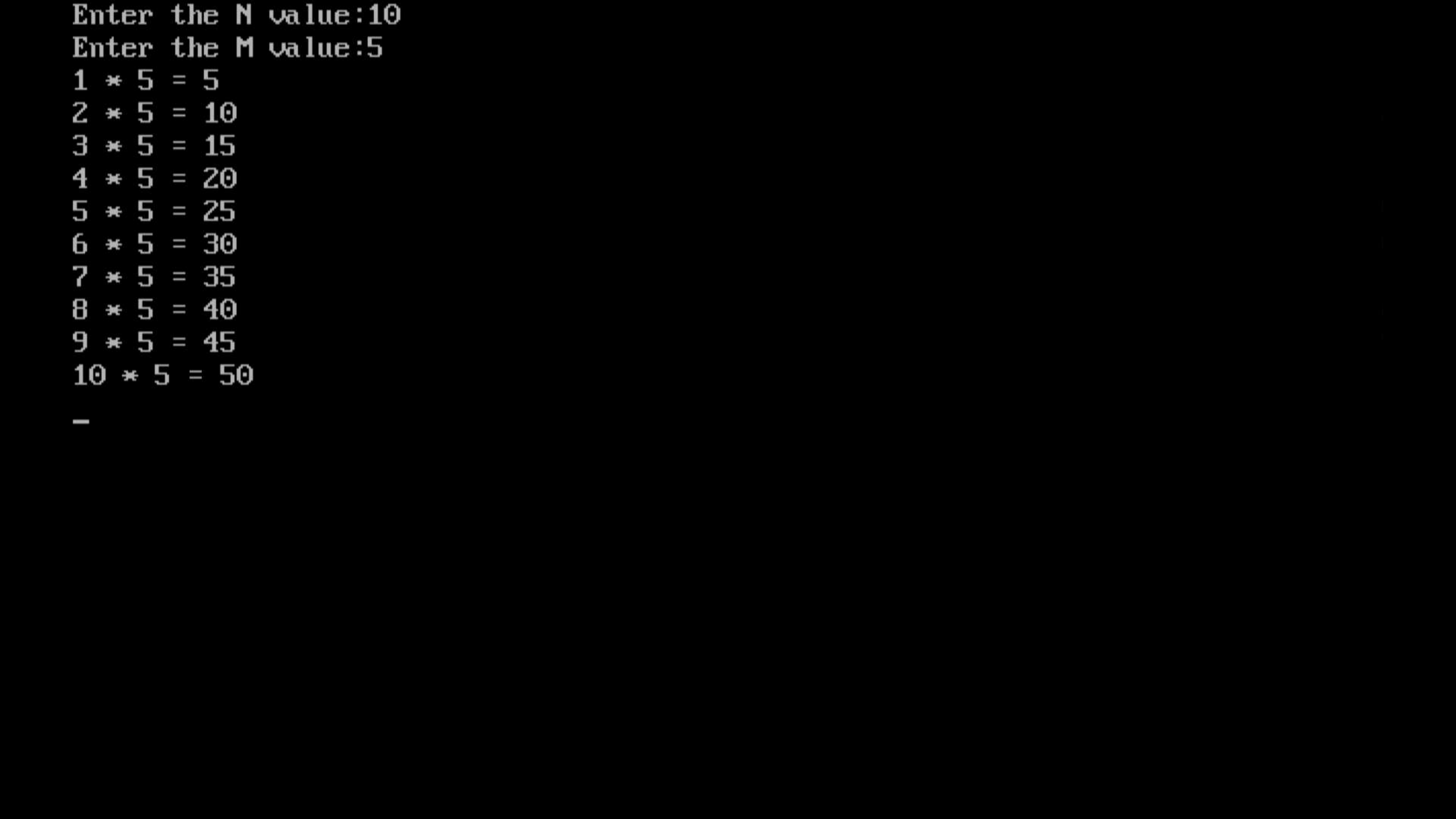
Break (breaks the loop/switch)

- Break statement is used to terminate the current loop iteration or terminate the switch statement in which it appears.
- When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
- Break statement can be used in the following scenarios:
  - for loop (For loop & nested for loop.
  - While (while loop & nested while loop).
  - Do while (do while loop and nested while loop)
  - Switch case (Switch cases and nested switch cases)


Conditional Code
True
Condition
break Statement
False


≡ File Edit Search Run Compile Debug Project Options Window Help
BREAK.C
1
16:22
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

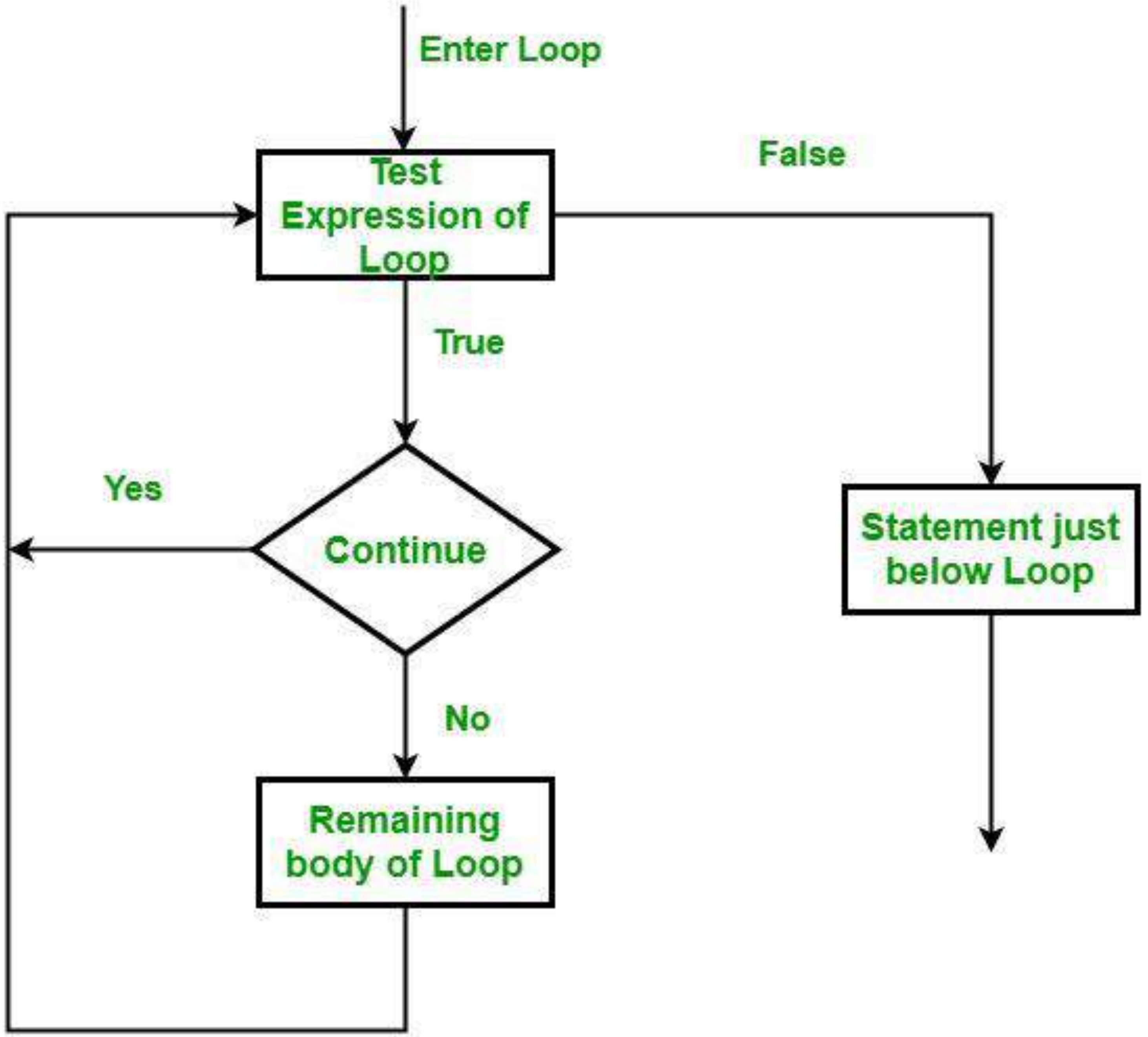
```
void main()
{
        int i,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        for(i=1;;i++)
        {
                if(i>n)
                        break;
                printf("%d * %d = %d\n",i,m,i*m);
        }
getch();
}
```

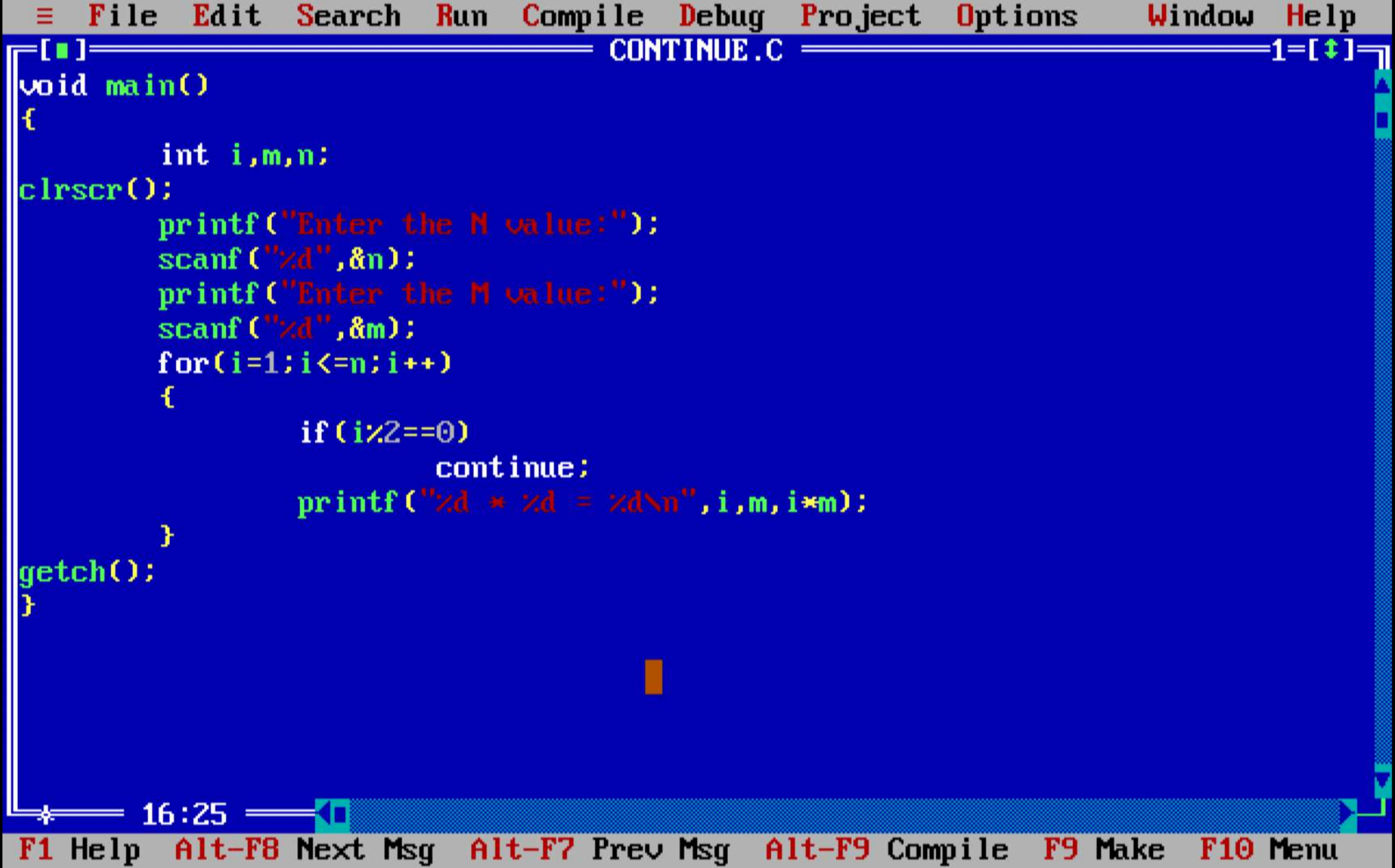


```
Enter the N value:10
Enter the M value:5
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50

_
```

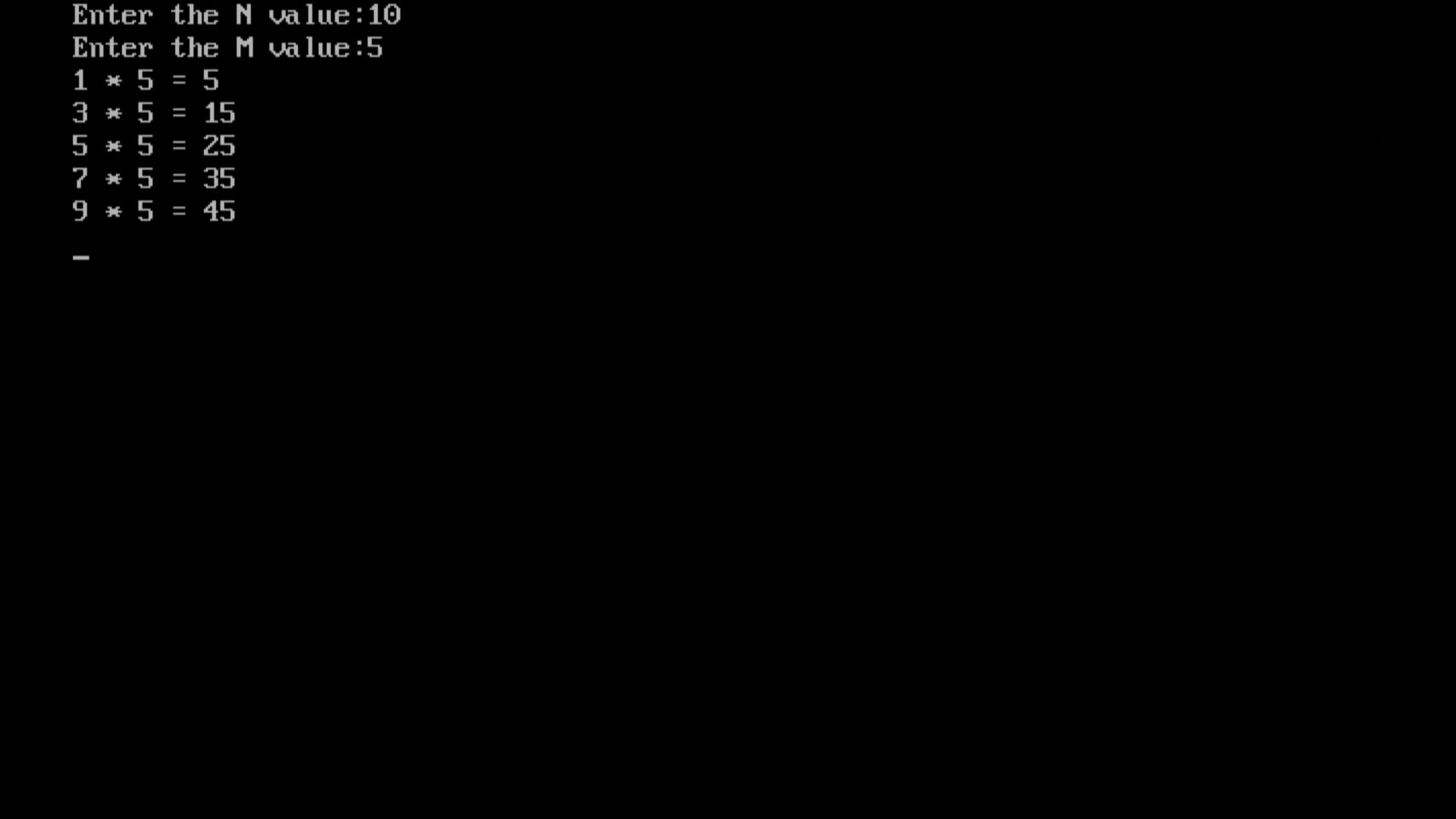
Continue Statements

- A Continue statement jumps out of the current loop condition and jumps back to the starting of the loop code.
- It is represented by continue;
- Continue statement can be used in the following scenarios:
  - for loop (For loop & nested for loop.
  - While (while loop & nested while loop).
  - Do while (do while loop and nested while loop)
  - Switch case (Switch cases and nested switch cases)


Enter Loop
Test Expression of Loop
False
True
Yes
Continue
No
Statement just below Loop
Remaining body of Loop


≡ File Edit Search Run Compile Debug Project Options Window Help
CONTINUE.C
1
16:25
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

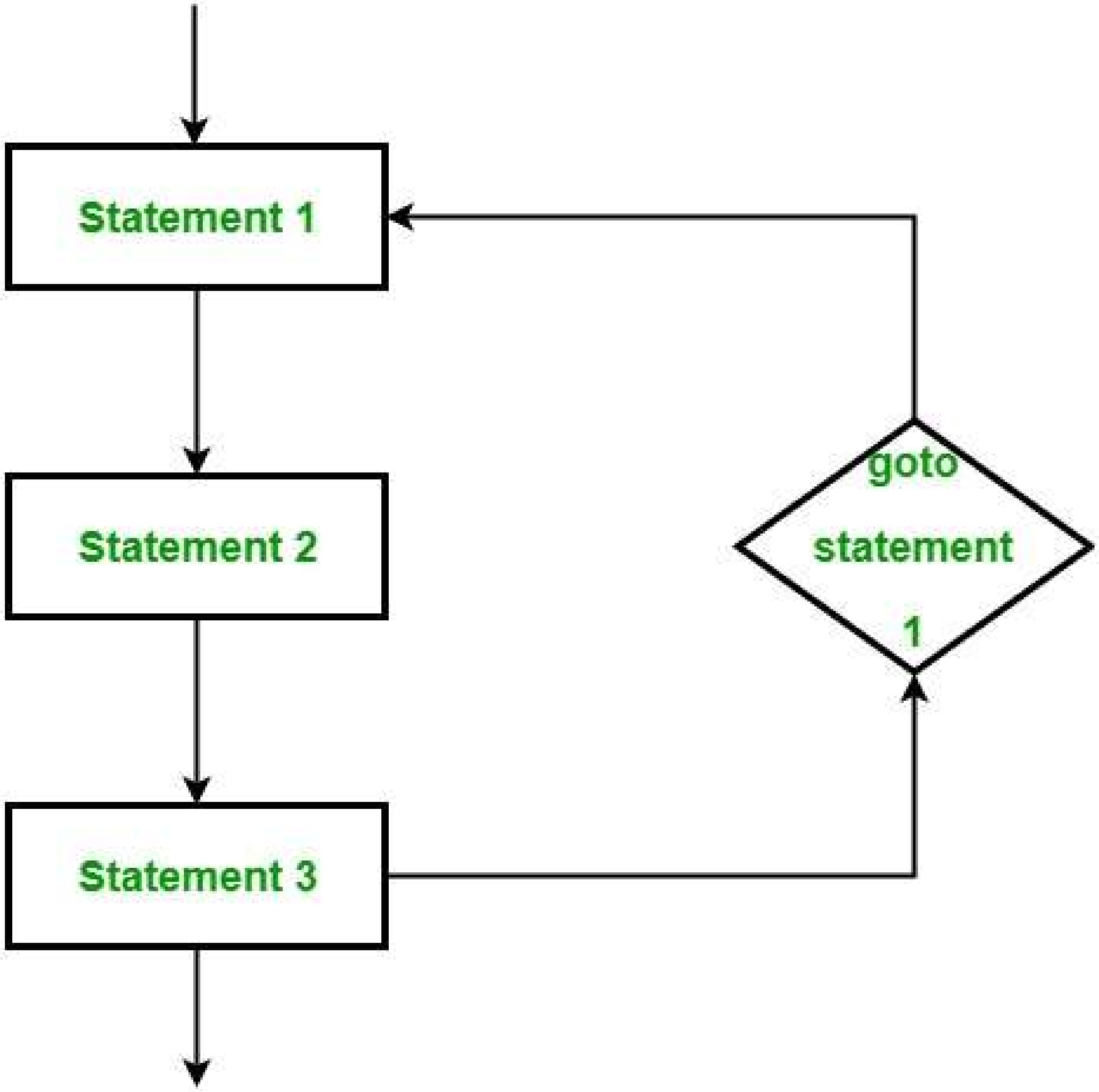
```
void main()
{
        int i,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        for(i=1;i<=n;i++)
        {
                if(i%2==0)
                        continue;
                printf("%d * %d = %d\n",i,m,i*m);
        }
getch();
}
```

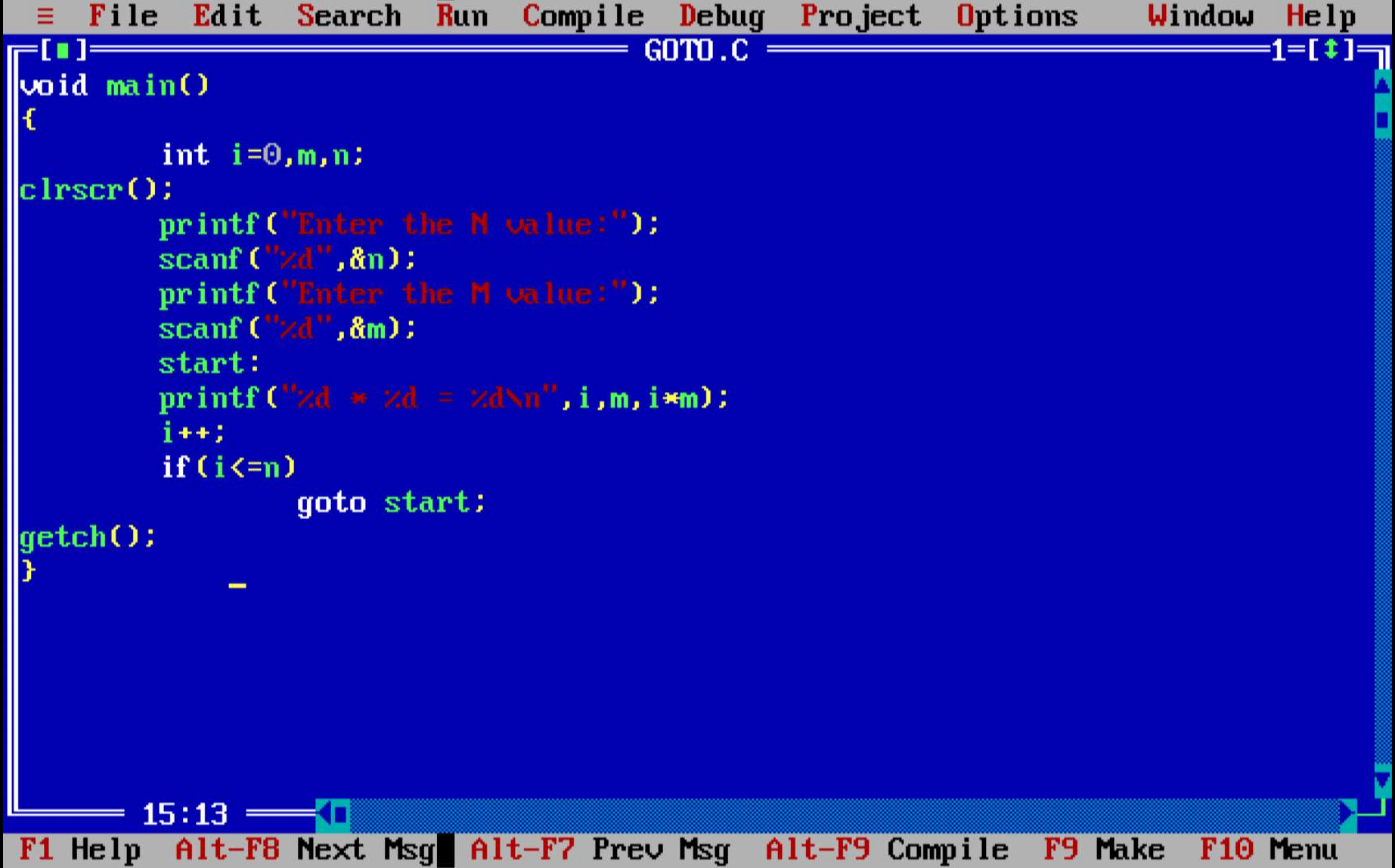


```
Enter the N value:10
Enter the M value:5
1 * 5 = 5
3 * 5 = 15
5 * 5 = 25
7 * 5 = 35
9 * 5 = 45

_
```

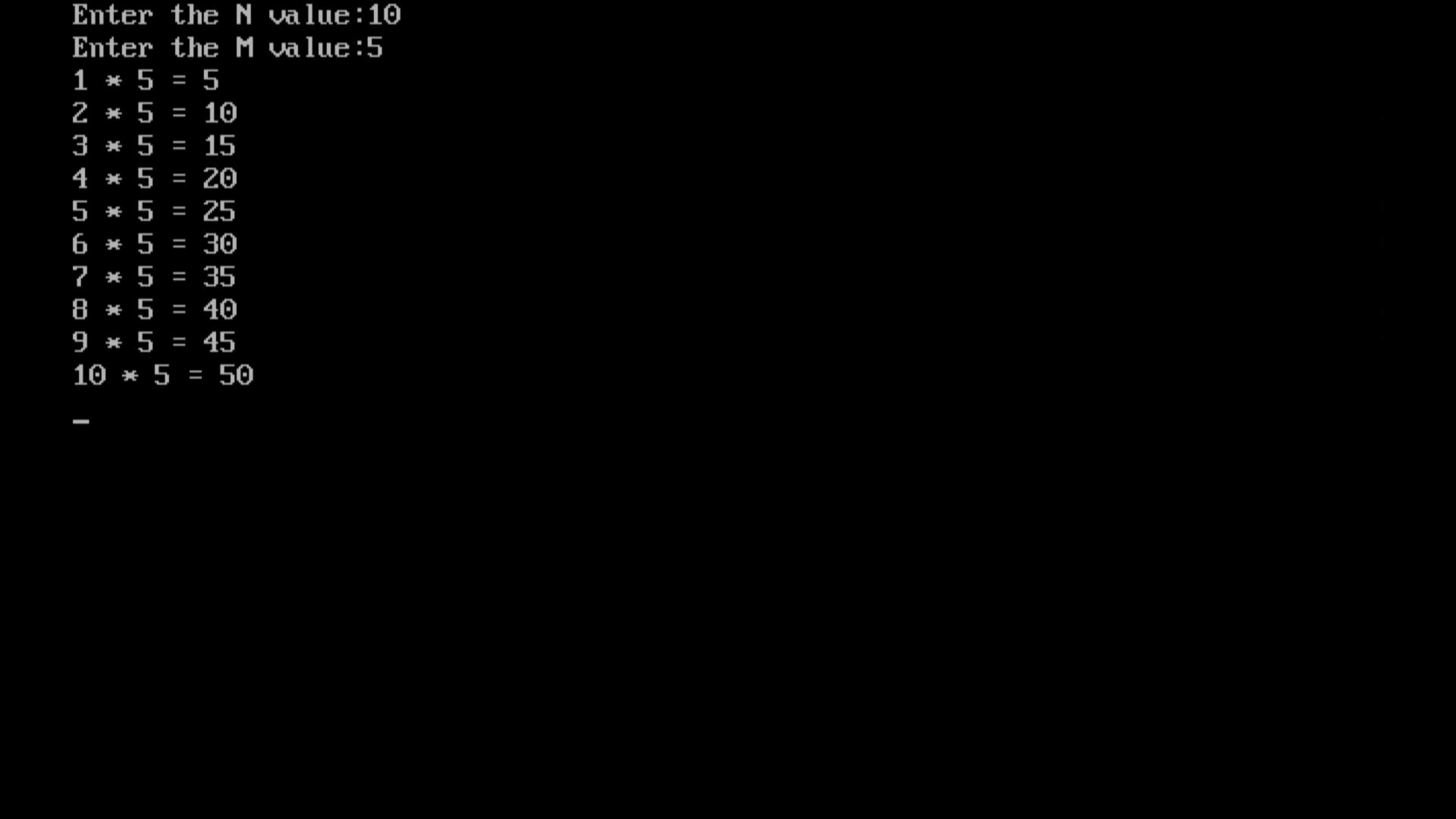
goto statement

- The goto statement transfers the program control directly to a labeled statement.
- The label is the valid identifier and placed just before the statement from where the control is transferred.
- A common use of goto is to transfer control to a specific switch-case label or the default label in a switch statement.
- The goto statement is also useful to get out of deeply nested loops.


Statement 1
Statement 2
Statement 3
goto
statement
1


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] GOTO.C 1=[↕]
15:13
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int i=0,m,n;
clrscr();
        printf("Enter the N value:");
        scanf("%d",&n);
        printf("Enter the M value:");
        scanf("%d",&m);
        start:
        printf("%d * %d = %d\n",i,m,i*m);
        i++;
        if(i<=n)
                goto start;
getch();
}
```



```
Enter the N value:10
Enter the M value:5
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50

_
```

ARRAYS IN C

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

20-01-2022

III Bsc Mathematics

Arrays-Introduction

- The variables are used to store data. These variables are the one of the basic building blocks in C.
- A single variable is used to store a single value that can be used anywhere in the memory.
- In some situations, we need to store multiple values of the same type. In that case, it needs multiple variables of the same data type. All the values are stored randomly anywhere in the memory.
- For example, to store the roll numbers of the 100 students, it needs 100 variables named as roll1, roll2, roll3,…….roll100 . It becomes very difficult to declare 100 variables and store all the roll numbers.
- In C, the concept of Array helps to store multiple values in a single variable.
- An array is also a derived data type in C.

Arrays

- "An array is a collection of variables of the same type that are referenced by a common name".
- In an array, the values are stored in a fixed number of elements of the same type sequentially in memory. Therefore, an integer array holds a sequence of integers; a character array holds a sequence of characters, and so on.
- Types of Array
  - Single Dimensional Array
  - Two Dimensional Array
  - Multi Dimensional Array
- A one dimensional array represents values that are stored in a single row or in a single column.

- Syntax

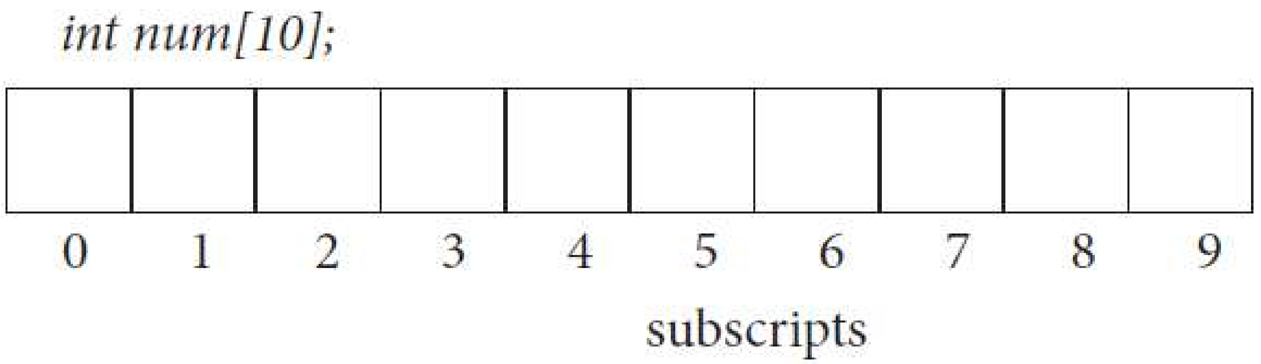
```
<data type><array_name> [<array_size>];
```

  - Where, data_type declares the basic type of the array, which is the type of each element in the array.
  - array_name specifies the name with which the array will be referenced.
  - array_size defines how many elements the array will hold. Size should be specified with square brackets [ ].

Example:

```
int num[10];
```

- In the above declaration, an array named “num” is declared with 10 elements (memory space to store 10 different values) as integer type. To the above declaration, the compiler allocated 10 memory locations (boxes) in the common name “num”.



- Each element (Memory box) has a unique index number starting from 0 which is known as “subscript”.
- The subscript always starts with 0 and it should be an unsigned integer value.
- Each element of an array is referred by its name with subscript index within the square bracket.
- For example, num[3] refers to the 4th element in the array.

- Some more array declarations with various data types:

  char emp_name[25]; // character array named emp_name with size 25

  float salary[20]; // floating-point array named salary with size 20

  int a[5], b[10], c[15]; // multiple arrays are declared of type int.

- The amount of storage required to hold an array is directly related with type and size.

Initialization

- An array can be initialized at the time of its declaration. Unless an array is initialized, all the array elements contain garbage values.

Initialization

An array can be initialized at the time of its declaration. Unless an array is initialized, all the array elements contain garbage values.

Syntax:

<datatype> <array_name> [size] = {value-1,value-2,…………… ,value-n};

Example

int age[5]={19,21,16,1,50};

In the above example, the array name is 'age' whose size is 5. In this case, the first element 19 is stored in age[0], the second element 21 is stored in age[1] and so on as shown in figure 12.1

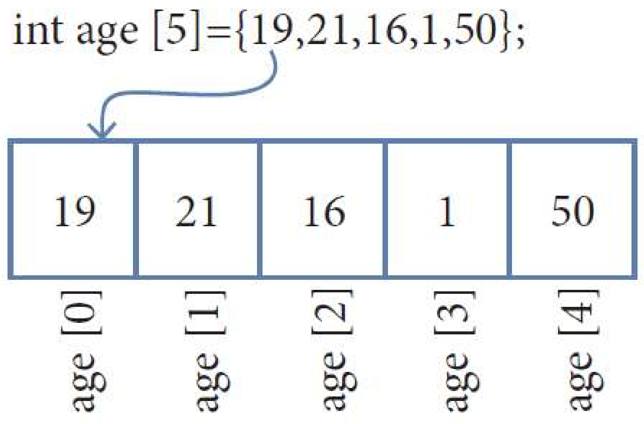


Figure 12.1

- While declaring and initializing values in an array, the values should be given within the curly braces ie. { ..... }
- The size of an array may be optional when the array is initialized during declaration.

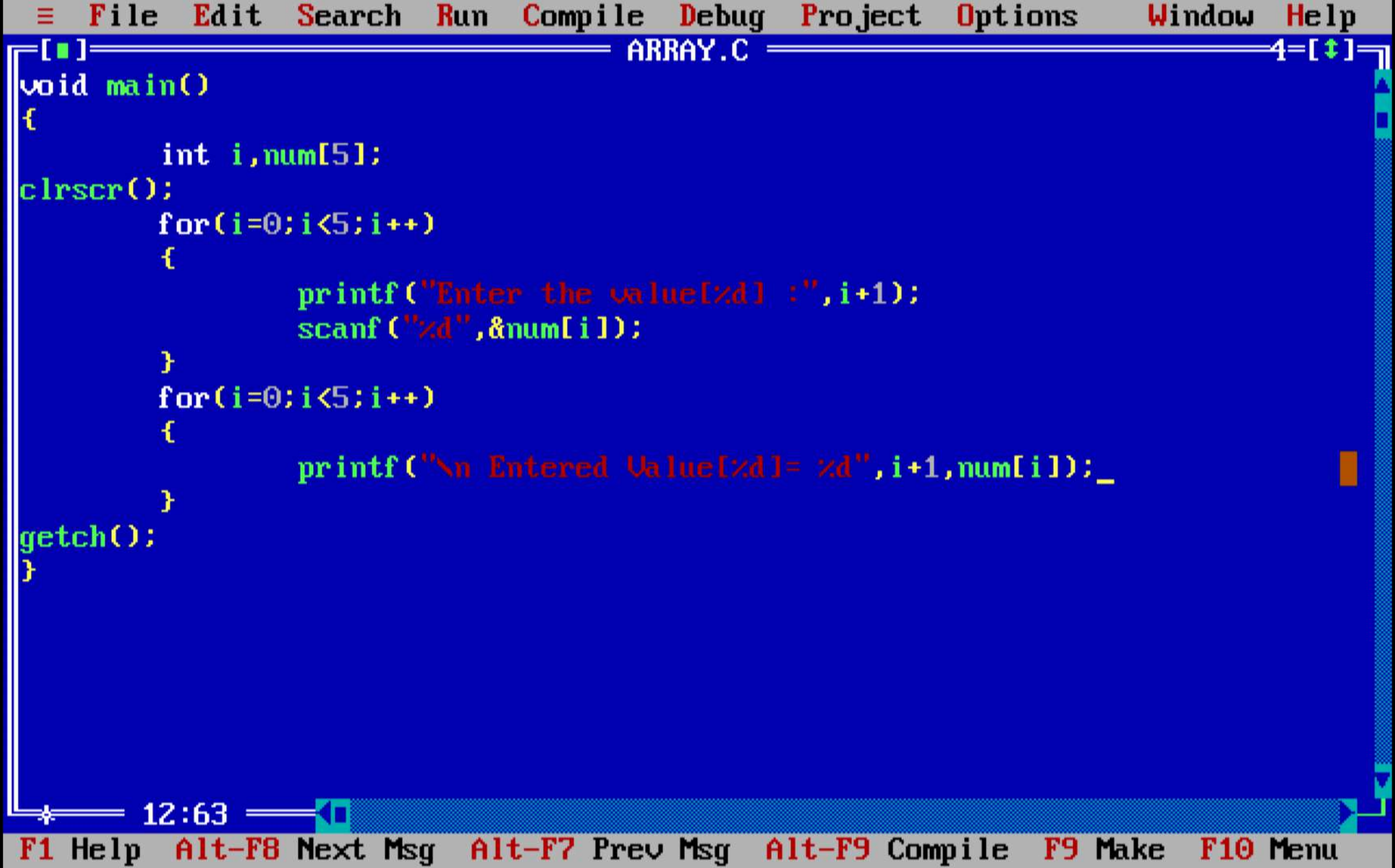
Example:

```
int age[]={ 19,21,16,1,50};
```

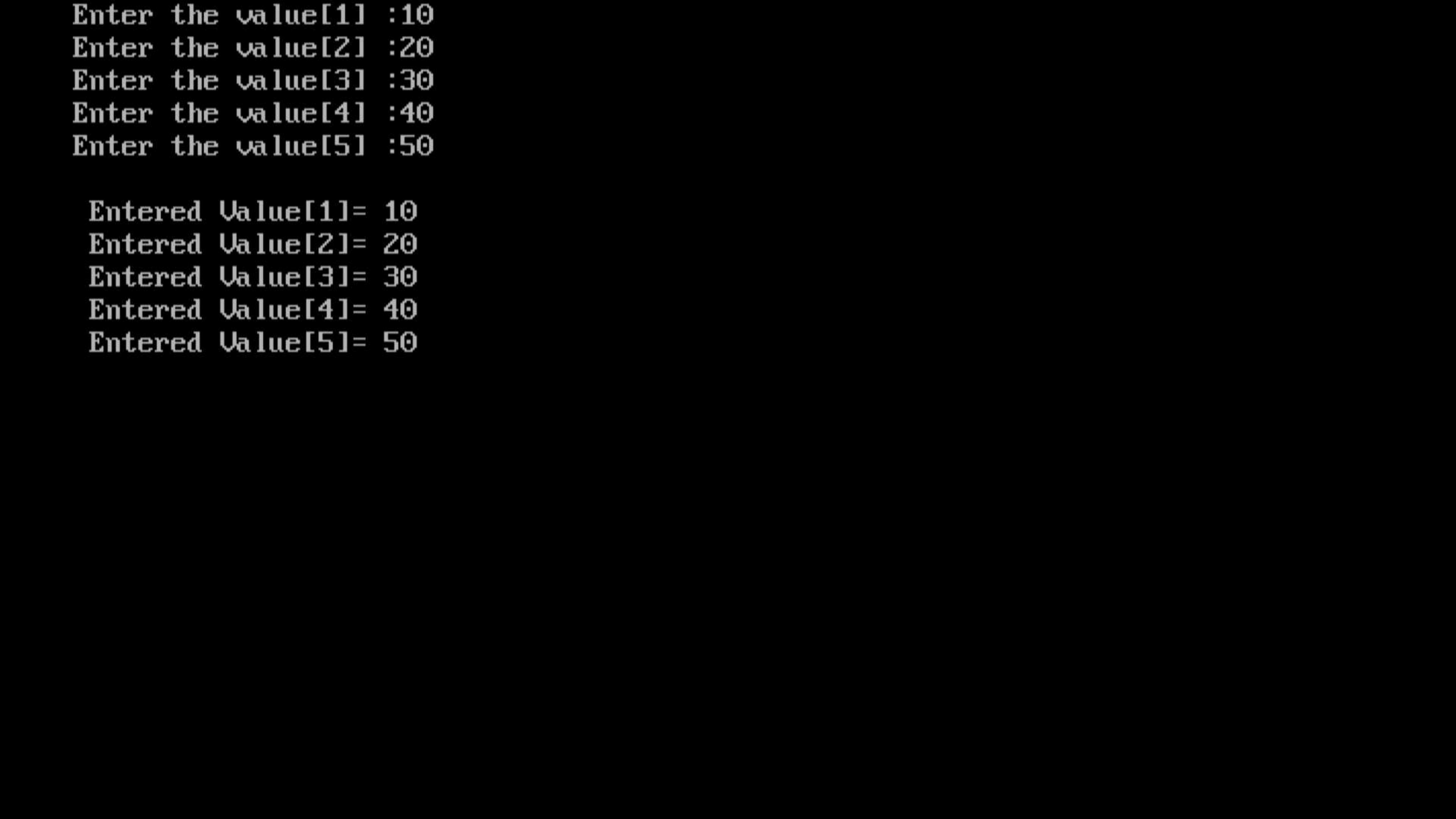
- In the above initialization, the size of the array is not specified directly in the declaration with initialization. So, the size is determined by compiler which depends on the total number of values. In this case, the size of the array is five.

Accepting values to an array during run time :

- Multiple assignment statements are required to insert values to the cells of the array during runtime. The for loop is ideally suited for iterating through the array elements.
- In the following program, a for loop has been constructed to execute the statements within the loop for 5 times.
- During each iteration of the loop, cout statement prompts you to “Enter value …….” and cin gets the value and stores it in num[i];


≡ File Edit Search Run Compile Debug Project Options Window Help
ARRAY.C
4
12:63
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int i,num[5];
clrscr();
        for(i=0;i<5;i++)
        {
                printf("Enter the value[%d] :",i+1);
                scanf("%d",&num[i]);
        }
        for(i=0;i<5;i++)
        {
                printf("\n Entered Value[%d]= %d",i+1,num[i]);
        }
getch();
}
```



```
    Enter the value[1] :10
    Enter the value[2] :20
    Enter the value[3] :30
    Enter the value[4] :40
    Enter the value[5] :50

     Entered Value[1]= 10
     Entered Value[2]= 20
     Entered Value[3]= 30
     Entered Value[4]= 40
     Entered Value[5]= 50
```

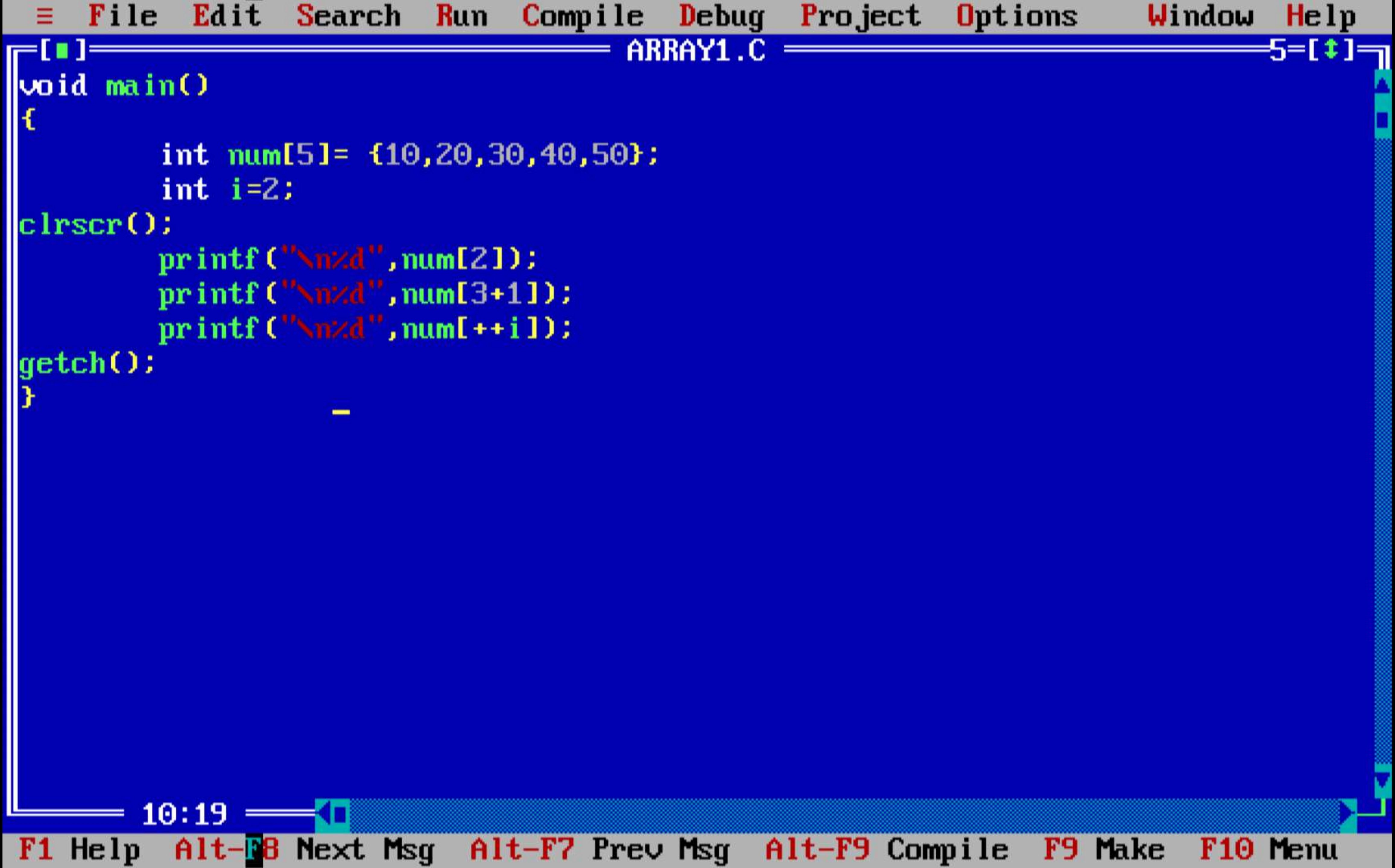
Accessing array elements

- Array elements can be used anywhere in a program as we do in case of a normal variable.
- The elements of an array are accessed with the array name followed by the subscript index within the square bracket.

Example:

```
cout<<num[3];
```

- In the above statement, num[3] refers to the 4th element of the array and cout statement displays the value of num[3].
- The subscript in bracket can be a variable, a constant or an expression that evaluates to an integer.


≡ File Edit Search Run Compile Debug Project Options Window Help
ARRAY1.C
5
10:19
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int num[5]= {10,20,30,40,50};
        int i=2;
clrscr();
        printf("\n%d",num[2]);
        printf("\n%d",num[3+1]);
        printf("\n%d",num[++i]);
getch();
}
```

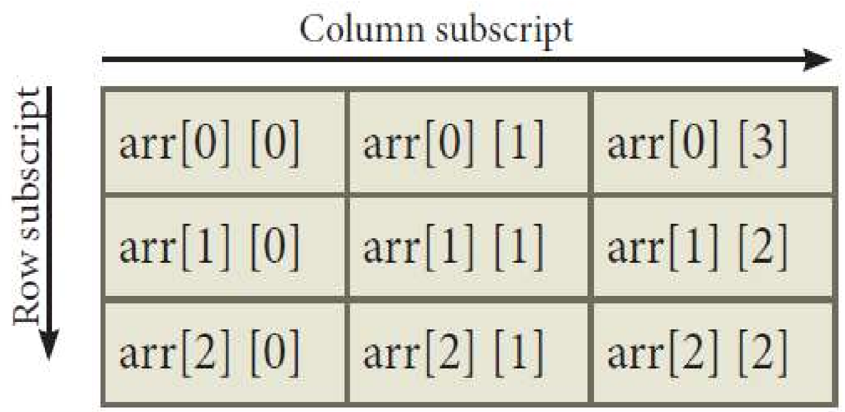
```
30
50
40_
```

Two-dimensional array

- Two-dimensional (2D) arrays are collection of similar elements where the elements are stored in certain number of rows and columns.
- An example m × n matrix where m denotes the number of rows and n denotes the number of columns.

int arr[3][3];

2D array conceptual memory representation



The array arr can be coneptually viewed in matrix form with 3 rows and coloumns. point to be noted here is since the subscript starts with 0 arr [0][0] represents the first element.

Figure 12.4

- The declaration of a 2-D array is

  data-type array_name[row-size][col-size];

- In the above declaration, data-type refers to any valid C data-type, array_name refers to the name of the 2-D array, row-size refers to the number of rows and col-size refers to the number of columns in the 2-D array.

For example

  int A[3][4];

- In the above example, A is a 2-D array, 3 denotes the number of rows and 4 denotes the number of columns. This array can hold a maximum of 12 elements.
- Array size must be an unsigned integer value which is greater than 0. In arrays, column size is compulsory but row size is optional.

12.3.2 Initialization of Two-Dimensional array

The array can be initialized in more than one way at the time of 2-D array declaration.

For example

```
int matrix[4][3]={
{10,20,30},// Initializes row 0
{40,50,60},// Initializes row 1
{70,80,90},// Initializes row 2
{100,110,120}// Initializes row 3
};
int matrix[4][3]={10,20,30,40,50,60,70,80,90,100,110,120};
```

Array's row size is optional but column size is compulsory.

```
int matrix[][3]={
{10,20,30},// row 0
{40,50,60},// row 1
{70,80,90},// row 2
{100,110,120}// row 3
};
```

12.3.3 Accessing the two-dimensional array

Two-dimensional array uses two index values to access a particular element in it, where the first index specifies the row value and second index specifies the column value.

matrix[0][0]=10;// Assign 10 to the first element of the first row

matrix[0][1]=20;// Assign 20 to the second element of the first row

matrix[1][2]=60;// Assign 60 to the third element of the second row

12.3.4 Memory representation of 2-D array

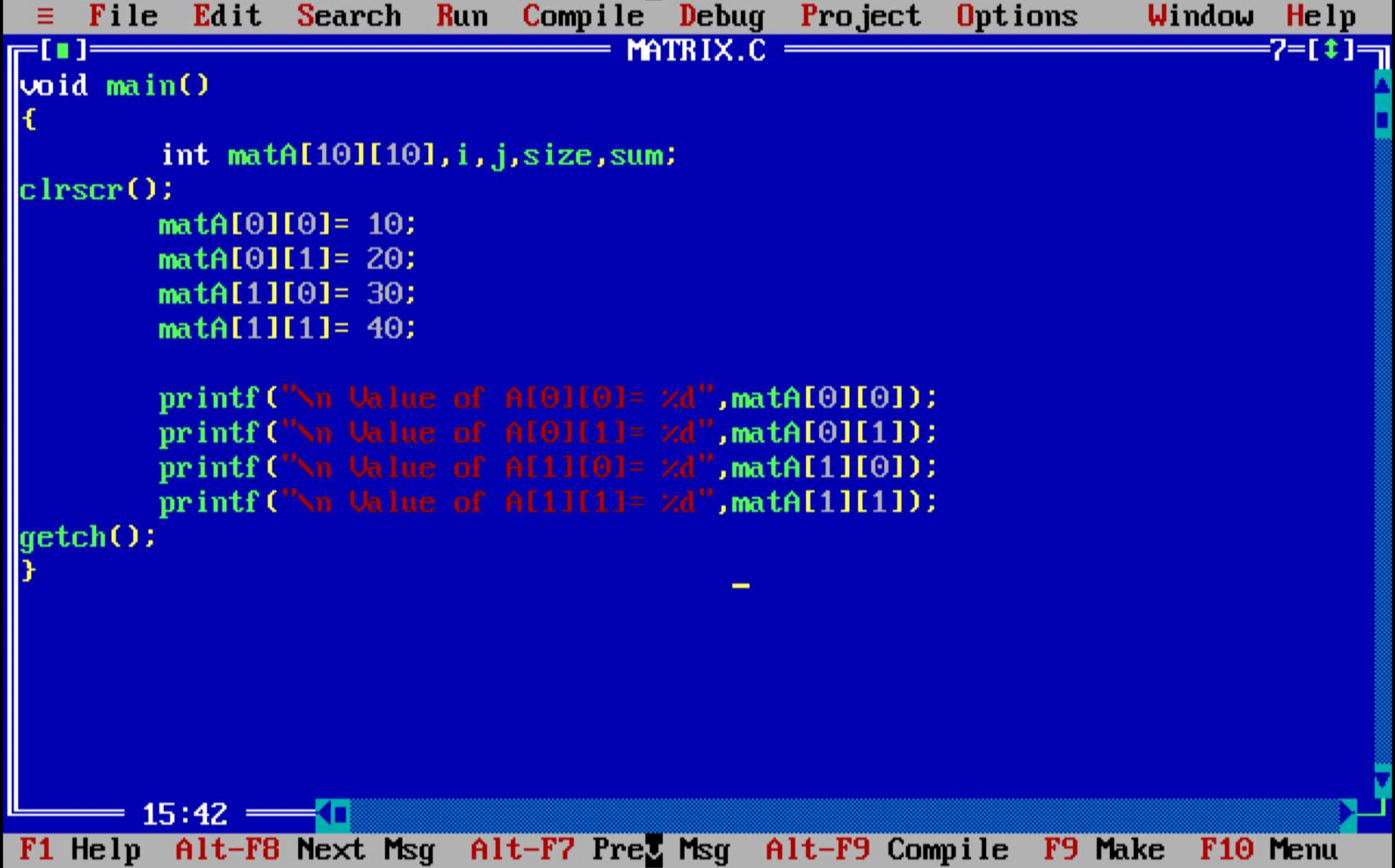
Normally, the two-dimensional array can be viewed as a matrix. The conceptual view of a 2-D array is shown below:

int A[4][3];

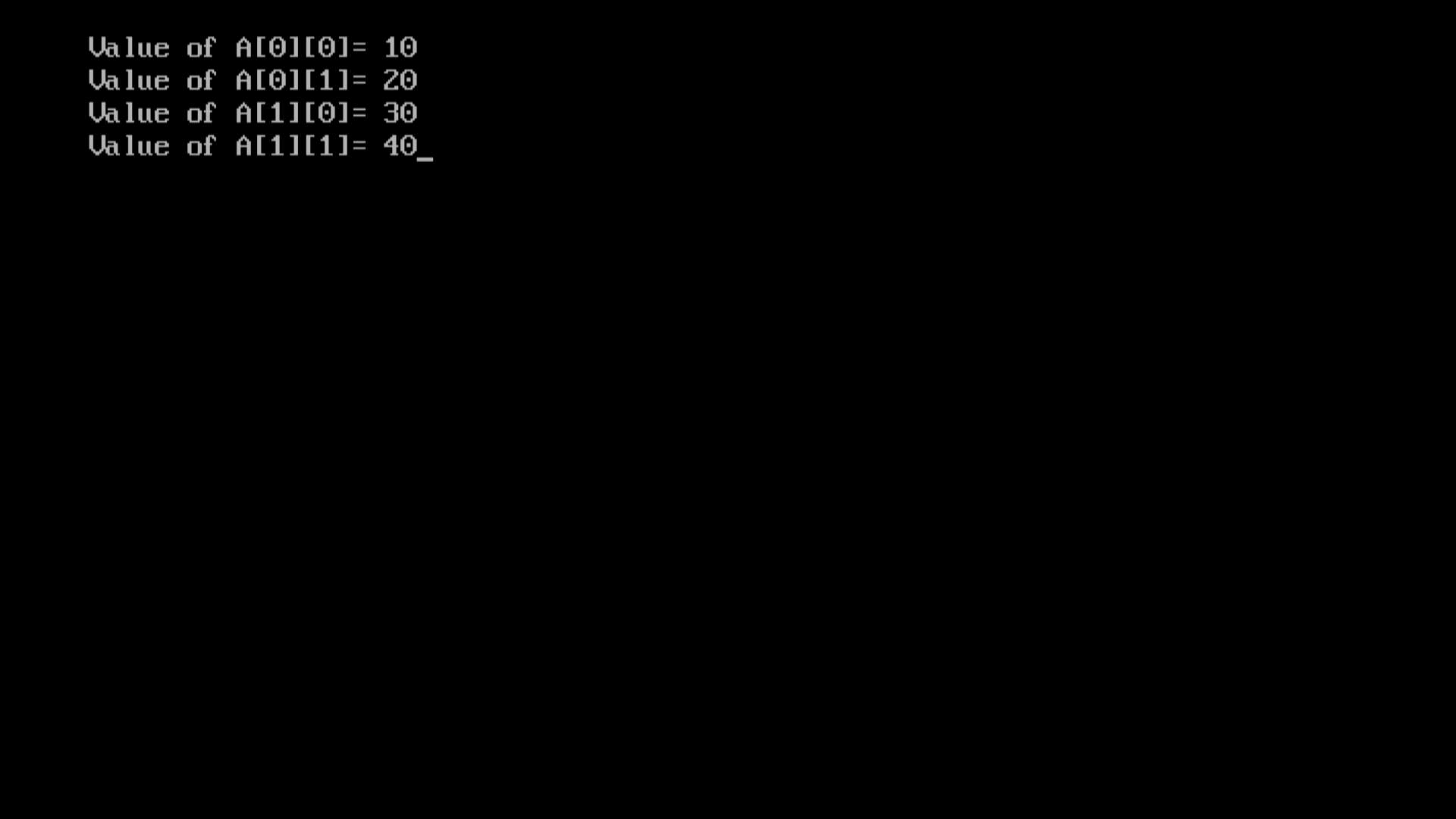
A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

In the above example, the 2-D array name A has 4 rows and 3 columns.

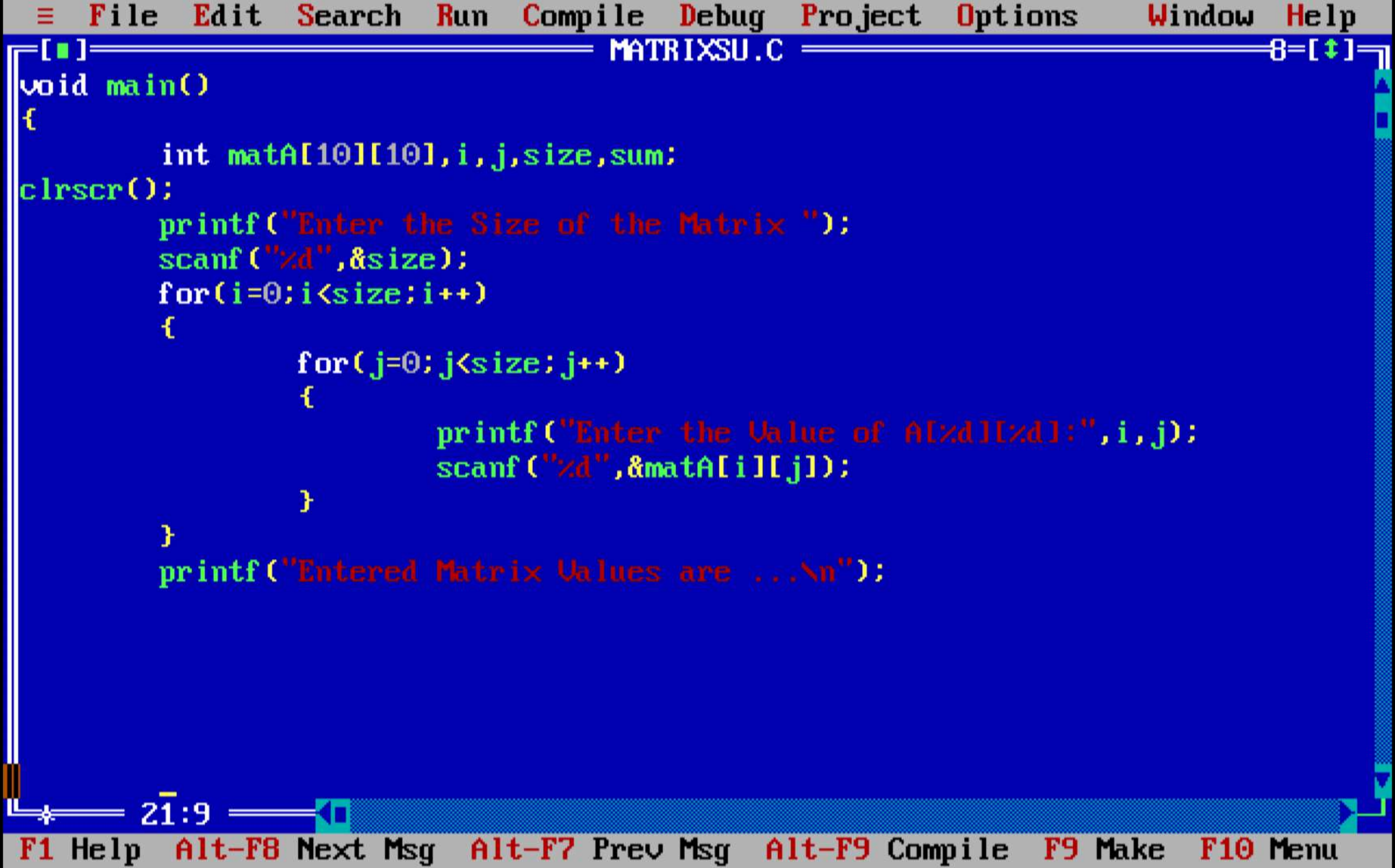
Like one-dimensional, the 2-D array elements are stored in continuous memory.


≡ File Edit Search Run Compile Debug Project Options Window Help
MATRIX.C
7
15:42
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

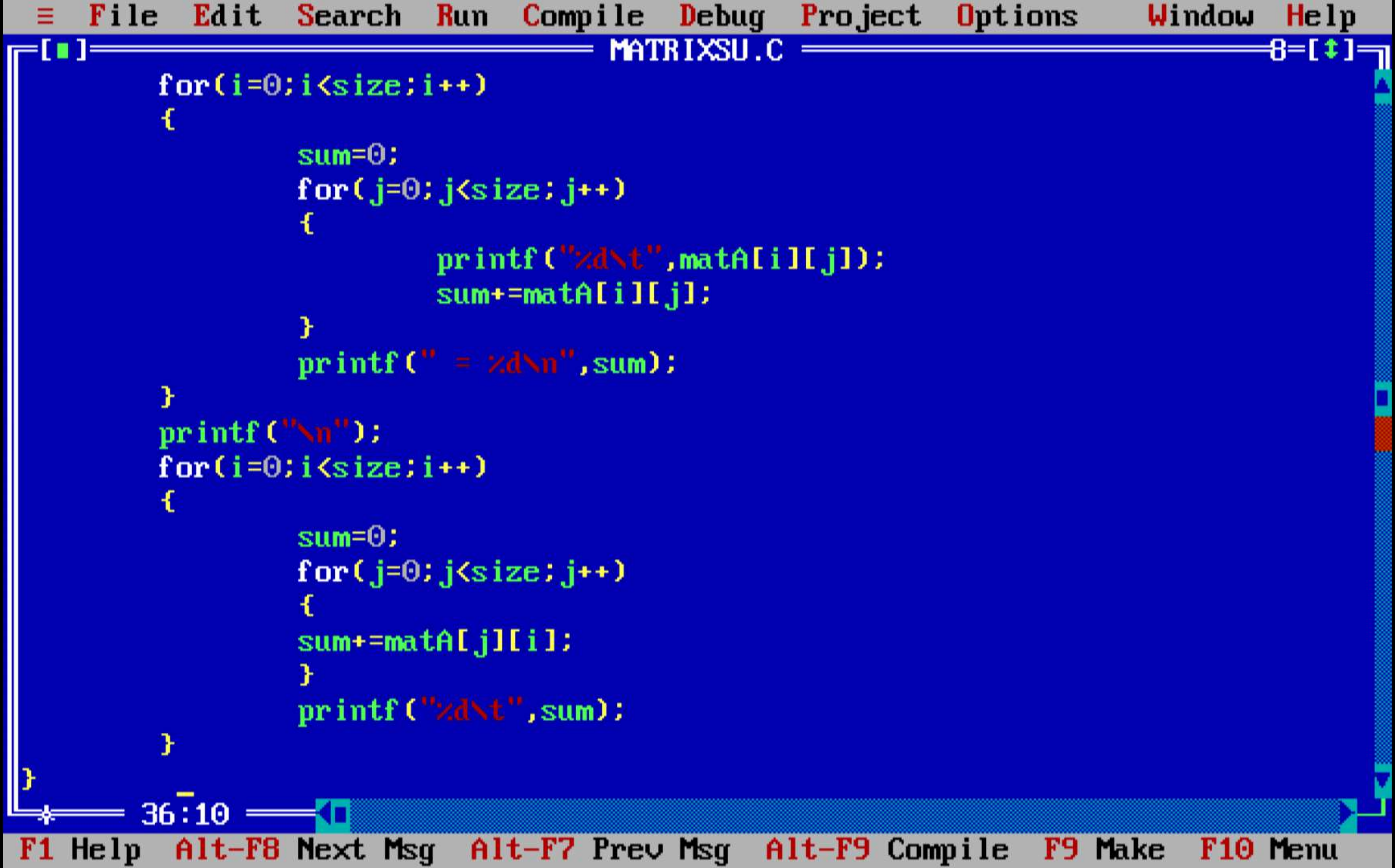
```
void main()
{
        int matA[10][10],i,j,size,sum;
clrscr();
        matA[0][0]= 10;
        matA[0][1]= 20;
        matA[1][0]= 30;
        matA[1][1]= 40;

        printf("\n Value of A[0][0]= %d",matA[0][0]);
        printf("\n Value of A[0][1]= %d",matA[0][1]);
        printf("\n Value of A[1][0]= %d",matA[1][0]);
        printf("\n Value of A[1][1]= %d",matA[1][1]);
getch();
}
```



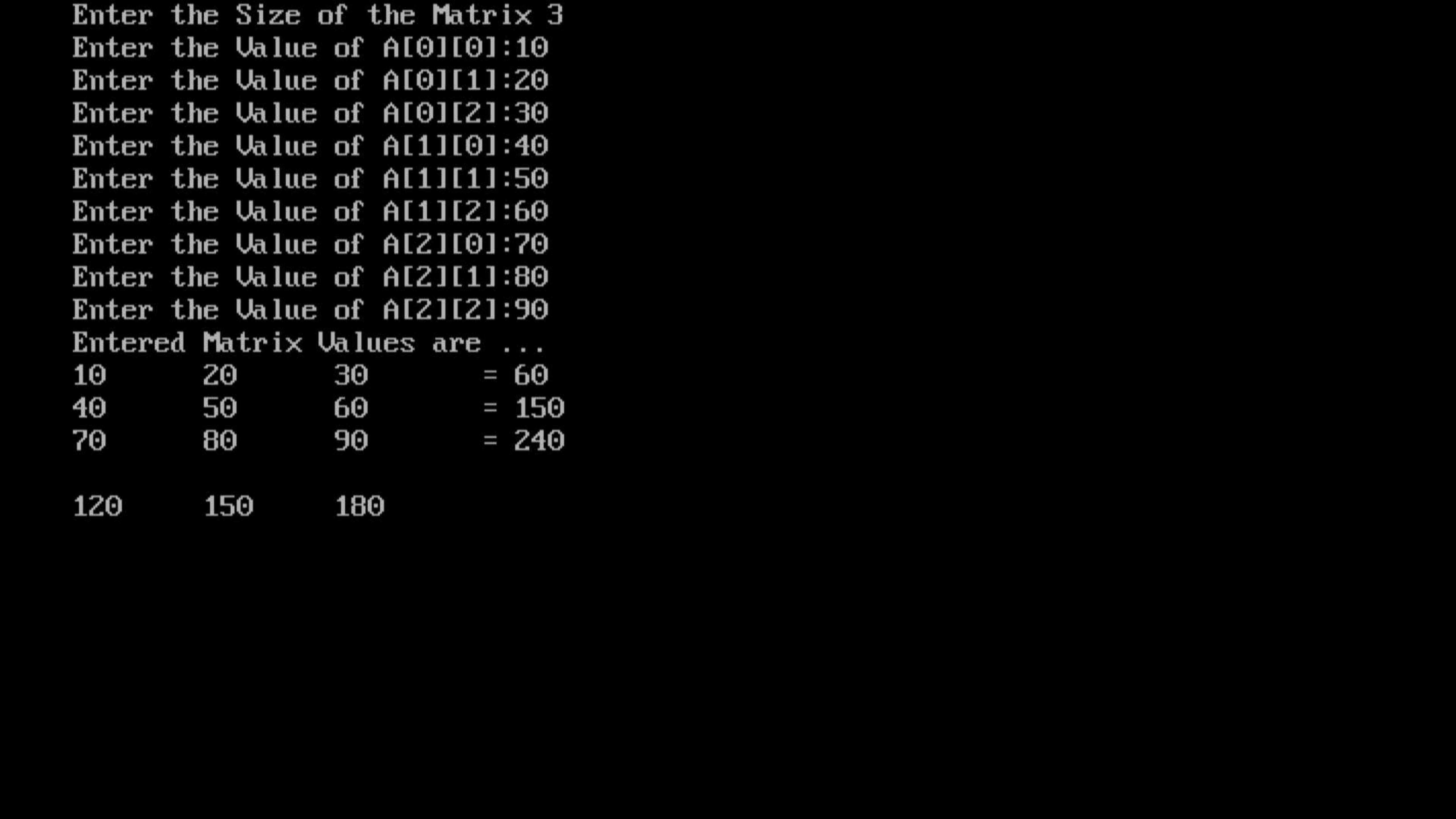
```
Value of A[0][0]= 10
Value of A[0][1]= 20
Value of A[1][0]= 30
Value of A[1][1]= 40_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
MATRIXSU.C
8
21:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int matA[10][10],i,j,size,sum;
clrscr();
        printf("Enter the Size of the Matrix ");
        scanf("%d",&size);
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        printf("Enter the Value of A[%d][%d]:",i,j);
                        scanf("%d",&matA[i][j]);
                }
        }
        printf("Entered Matrix Values are ...\n");
```


≡ File Edit Search Run Compile Debug Project Options Window Help
MATRIXSU.C
8
36:10
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
        for(i=0;i<size;i++)
        {
                sum=0;
                for(j=0;j<size;j++)
                {
                        printf("%d\t",matA[i][j]);
                        sum+=matA[i][j];
                }
                printf(" = %d\n",sum);
        }
        printf("\n");
        for(i=0;i<size;i++)
        {
                sum=0;
                for(j=0;j<size;j++)
                {
                sum+=matA[j][i];
                }
                printf("%d\t",sum);
        }
}
```



```
Enter the Size of the Matrix 3
Enter the Value of A[0][0]:10
Enter the Value of A[0][1]:20
Enter the Value of A[0][2]:30
Enter the Value of A[1][0]:40
Enter the Value of A[1][1]:50
Enter the Value of A[1][2]:60
Enter the Value of A[2][0]:70
Enter the Value of A[2][1]:80
Enter the Value of A[2][2]:90
Entered Matrix Values are ...
10      20      30       = 60
40      50      60       = 150
70      80      90       = 240

120     150     180
```

Advantages of an Array in C:

- Random access of elements using array index.
- Use of less line of code as it creates a single array of multiple elements.
- Easy access to all the elements.
- Traversal through the array becomes easy using a single loop.
- Sorting becomes easy as it can be accomplished by writing less line of code.

Disadvantages of an Array in C:

- Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
- Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.

Multidimensional Arrays

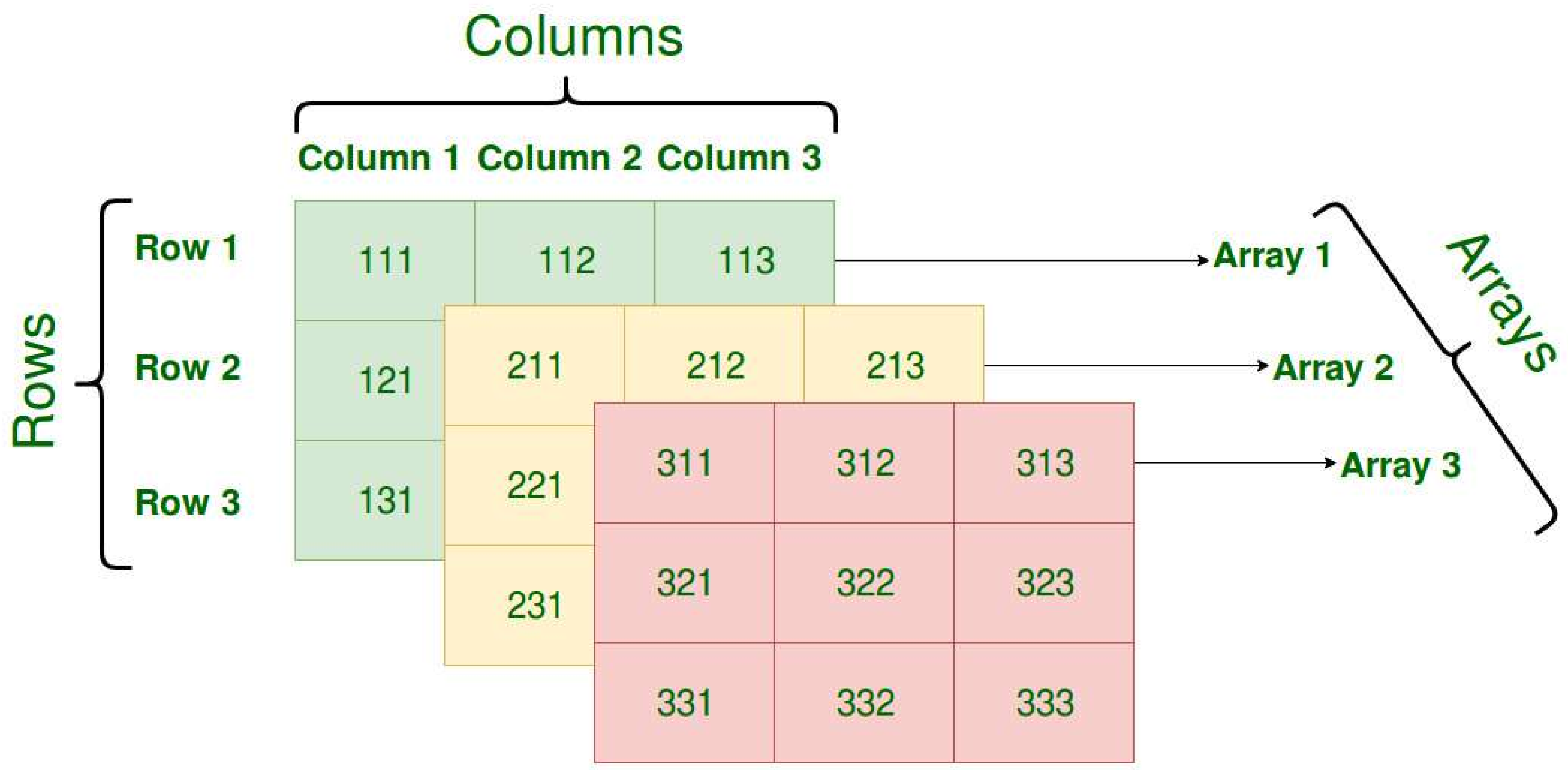
- In C, a 3d array is a multidimensional array used to store 3-dimensional information.
- In simple words, a three-dimensional array is an array of arrays.
- In three dimensional array, we have three rows and three columns.
- In multidimensional arrays data in the form of a table, that is in row-major order.
- The general syntax of a 3-dimensional array is as below.

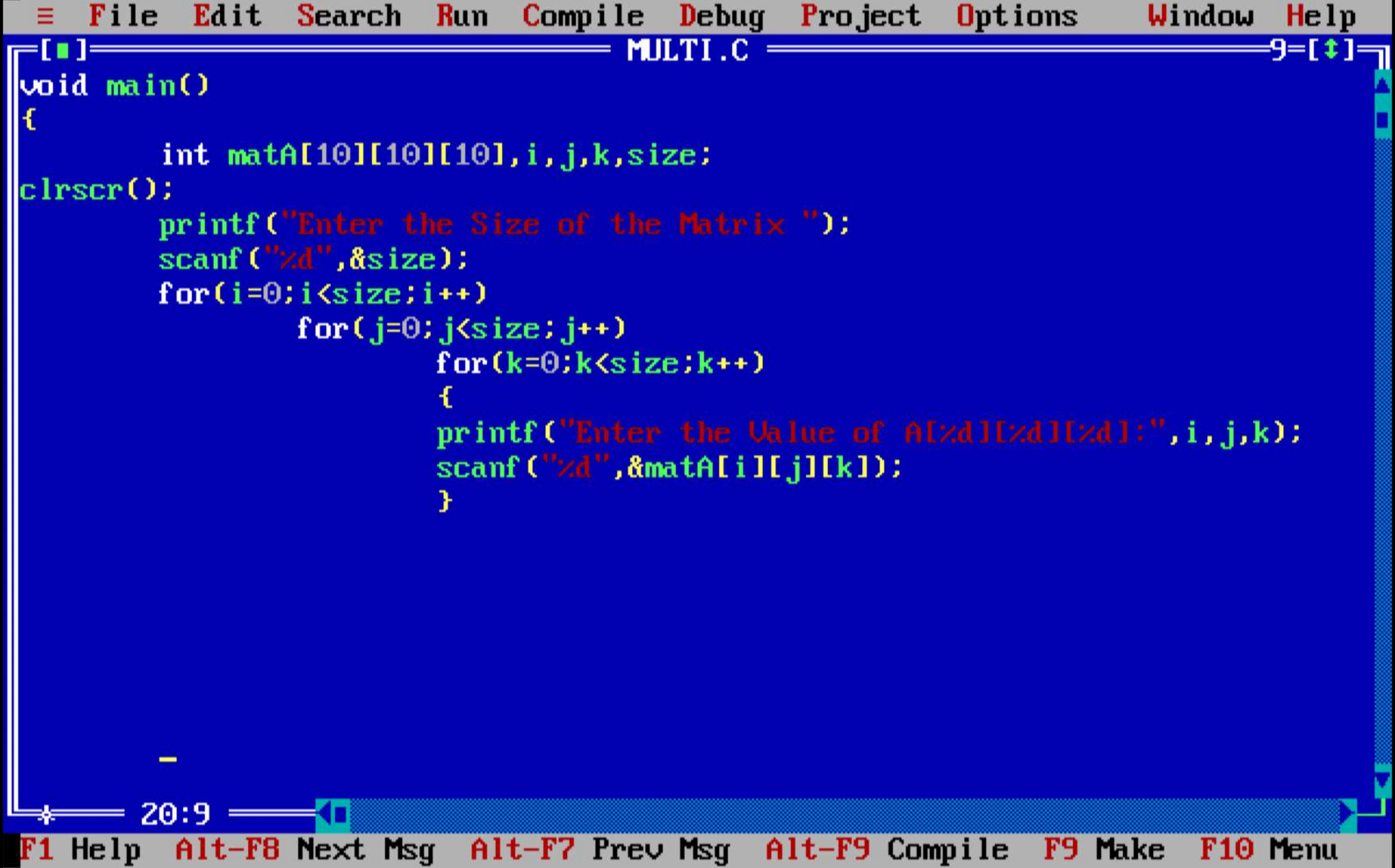
```
data_type array_name[size1][size2][size3];
```

Example

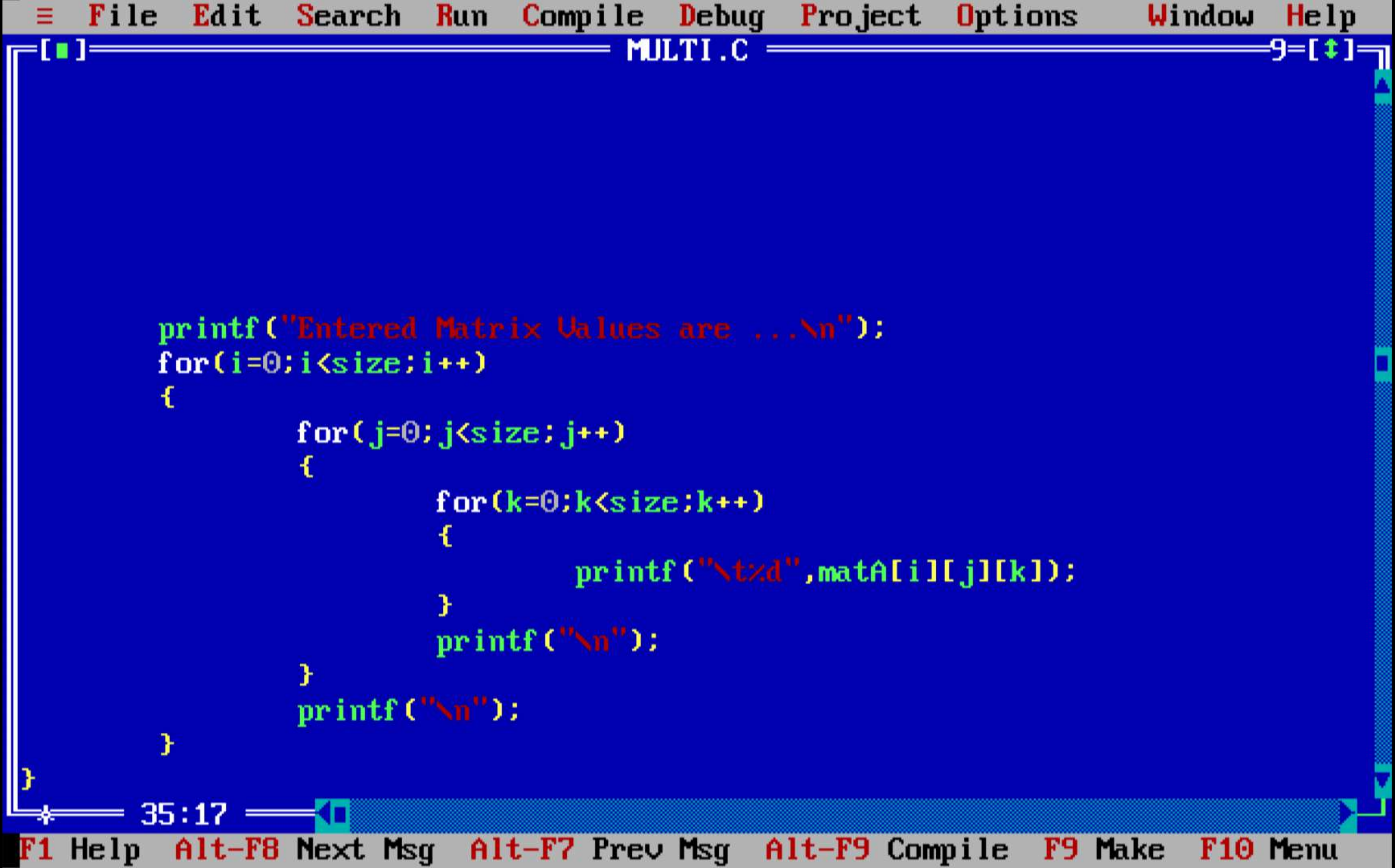
```
int 3DArray[2][3][4];
```

- where,3DArray is a three-dimensional array, having a maximum of 24 elements.

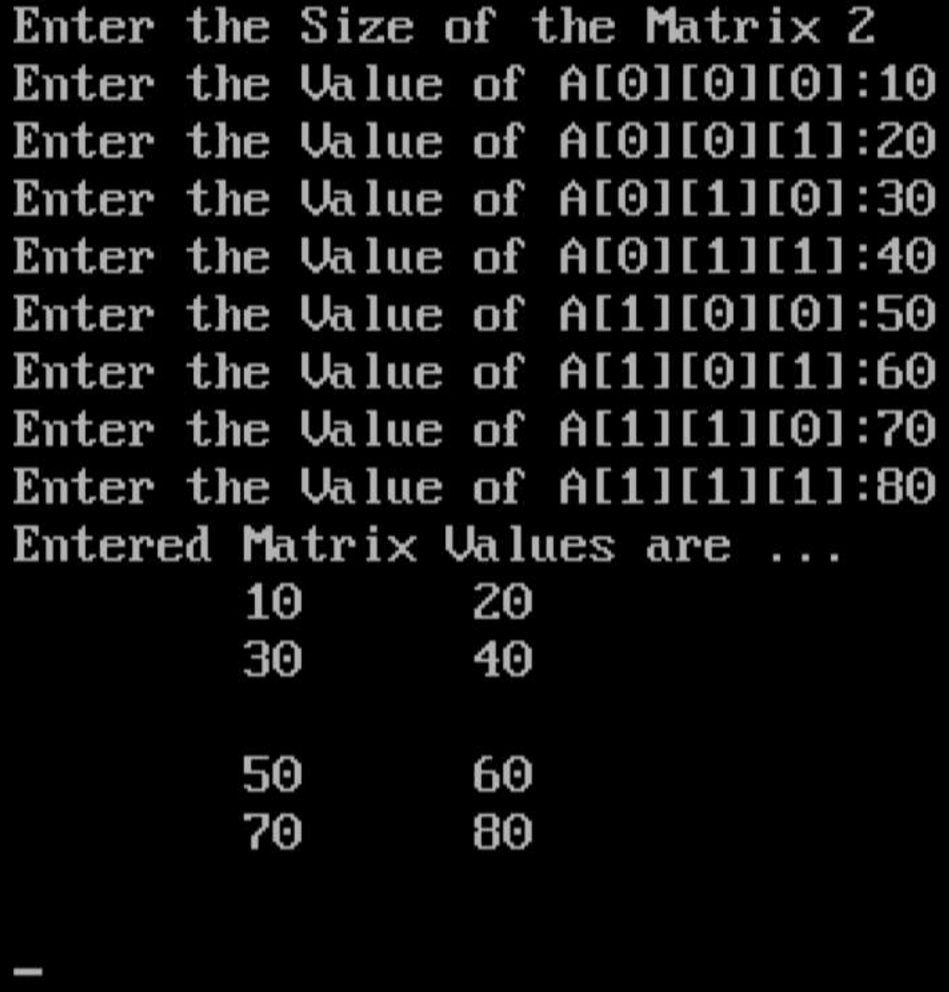

Columns
Column 1
Column 2
Column 3
Rows
Row 1
Row 2
Row 3
111
112
113
121
131
211
212
213
221
231
311
312
313
321
322
323
331
332
333
Array 1
Array 2
Array 3
Arrays


≡ File Edit Search Run Compile Debug Project Options Window Help
MULTI.C
9
20:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void main()
{
        int matA[10][10][10],i,j,k,size;
clrscr();
        printf("Enter the Size of the Matrix ");
        scanf("%d",&size);
        for(i=0;i<size;i++)
                for(j=0;j<size;j++)
                        for(k=0;k<size;k++)
                        {
                        printf("Enter the Value of A[%d][%d][%d]:",i,j,k);
                        scanf("%d",&matA[i][j][k]);
                        }
```


≡ File Edit Search Run Compile Debug Project Options Window Help
MULTI.C
9
35:17
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

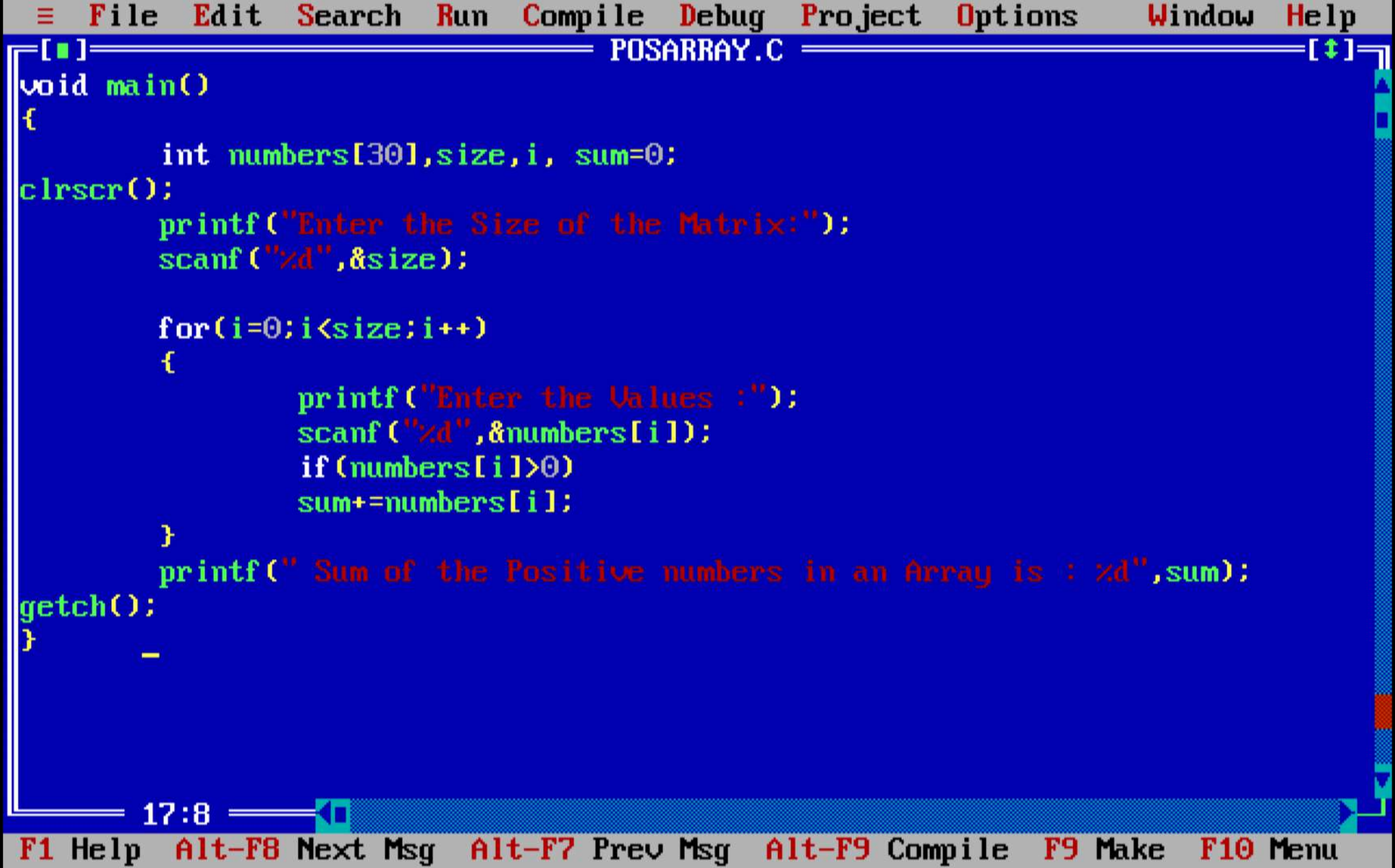
```
        printf("Entered Matrix Values are ...\n");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        for(k=0;k<size;k++)
                        {
                                printf("\t%d",matA[i][j][k]);
                        }
                        printf("\n");
                }
                printf("\n");
        }
}
```



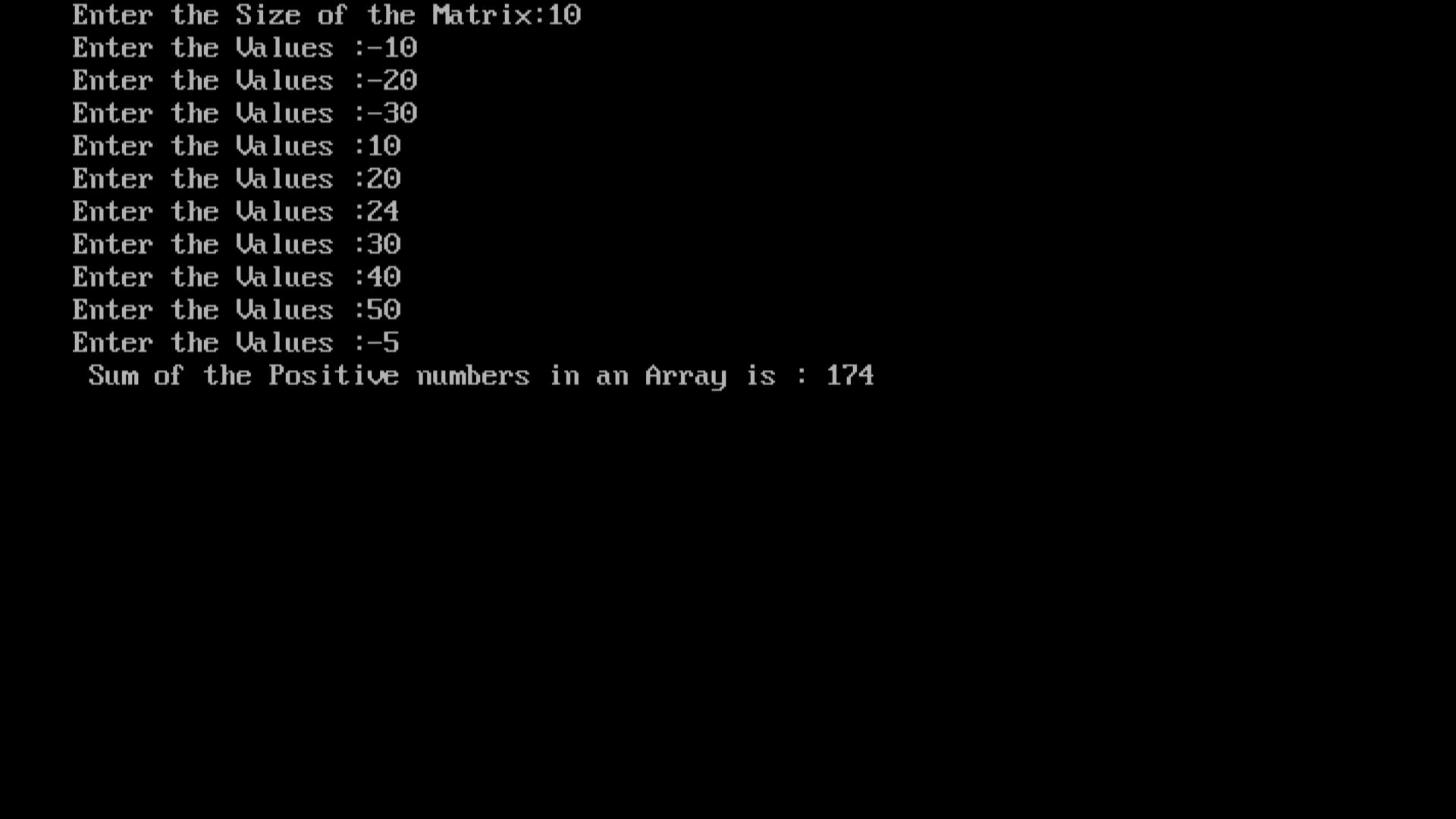
```
Enter the Size of the Matrix 2
Enter the Value of A[0][0][0]:10
Enter the Value of A[0][0][1]:20
Enter the Value of A[0][1][0]:30
Enter the Value of A[0][1][1]:40
Enter the Value of A[1][0][0]:50
Enter the Value of A[1][0][1]:60
Enter the Value of A[1][1][0]:70
Enter the Value of A[1][1][1]:80
Entered Matrix Values are ...
        10      20
        30      40

        50      60
        70      80


_
```

Program Exercise

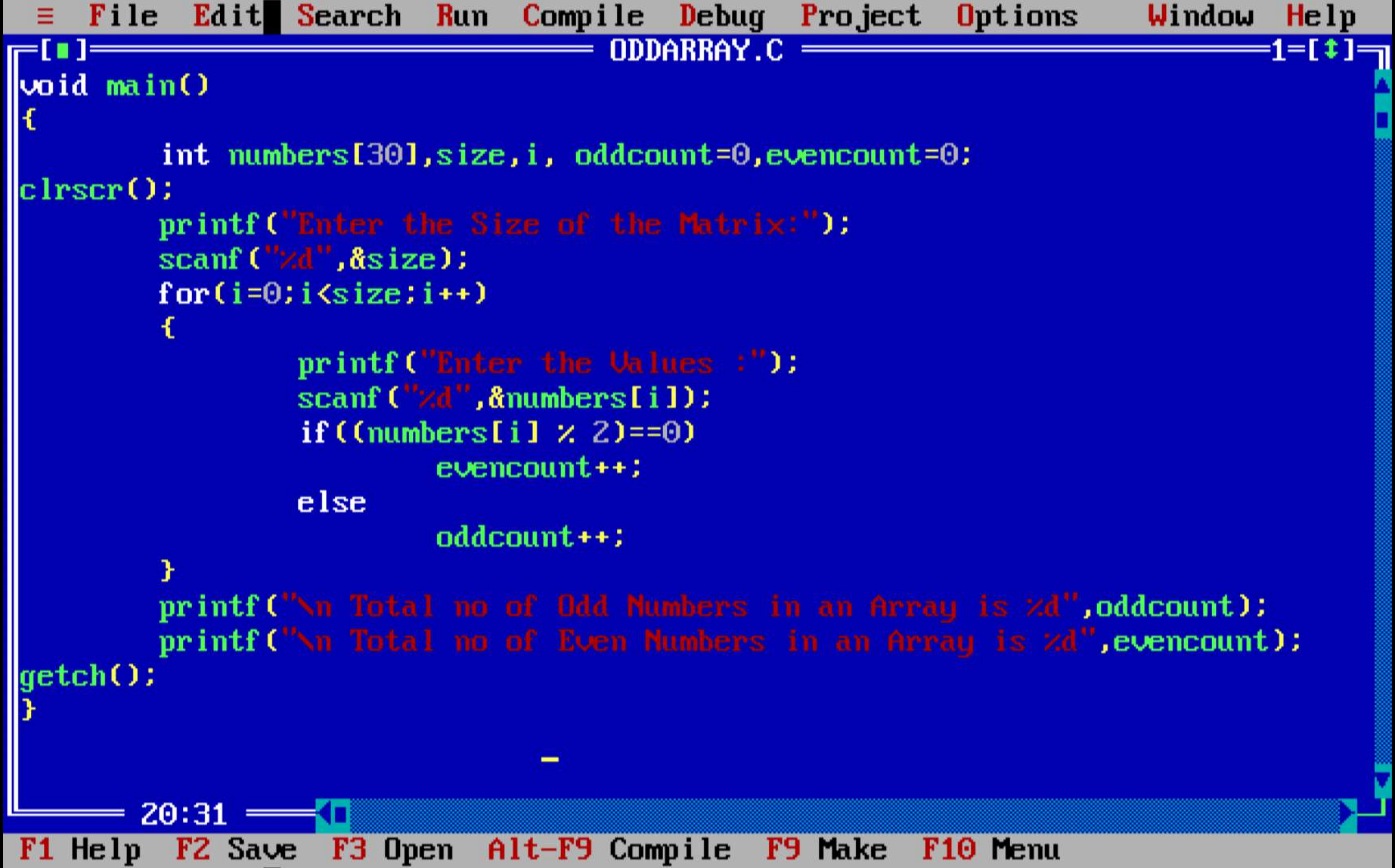
1. To Calculate the sum of positive numbers in an array.
2. To count the number of odd and even numbers in an array.
3. To perform Matrix Addition and Matrix Subtraction


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] POSARRAY.C [↕]
17:8
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

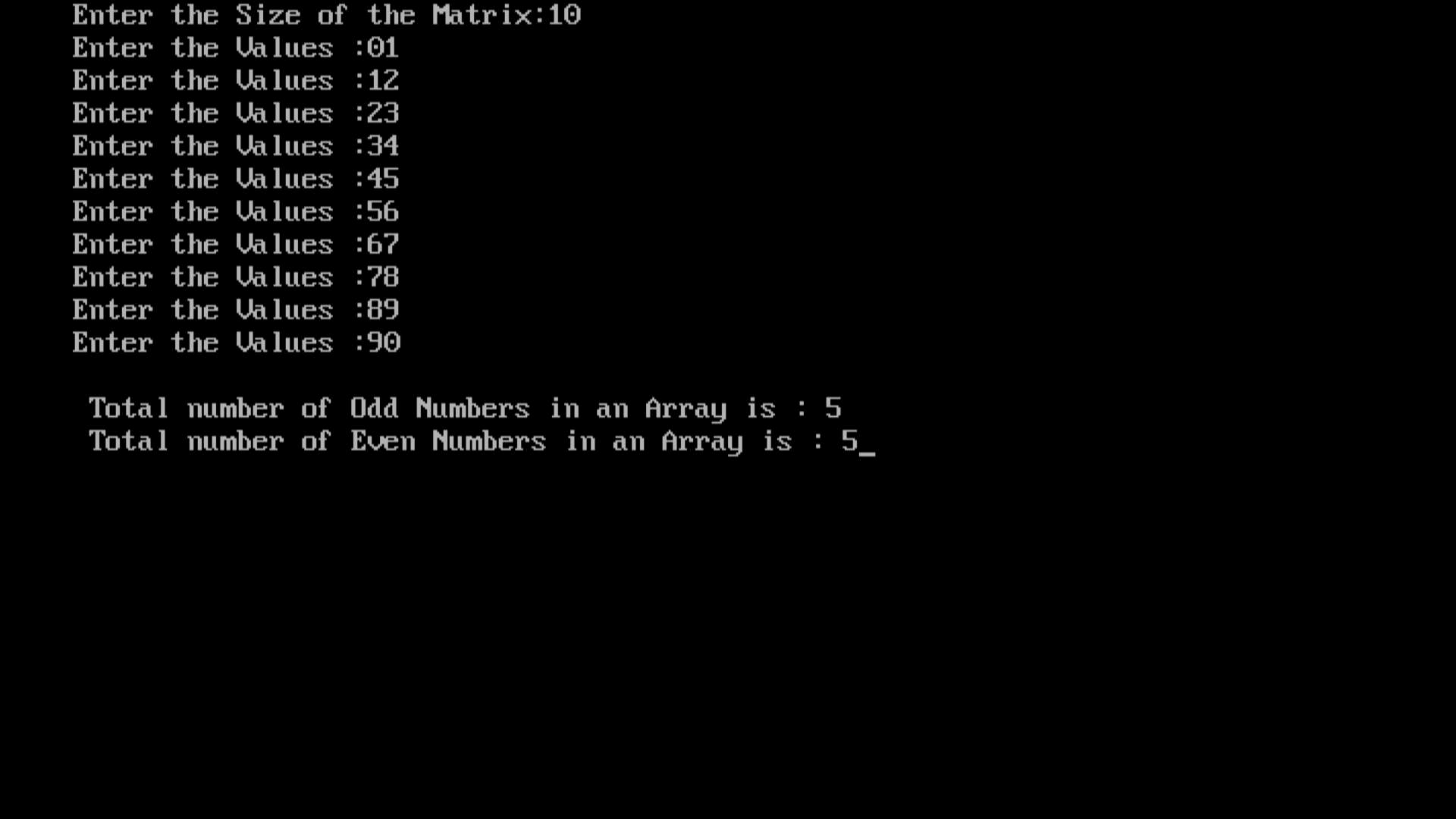
```
void main()
{
        int numbers[30],size,i, sum=0;
clrscr();
        printf("Enter the Size of the Matrix:");
        scanf("%d",&size);

        for(i=0;i<size;i++)
        {
                printf("Enter the Values :");
                scanf("%d",&numbers[i]);
                if(numbers[i]>0)
                sum+=numbers[i];
        }
        printf(" Sum of the Positive numbers in an Array is : %d",sum);
getch();
}
```



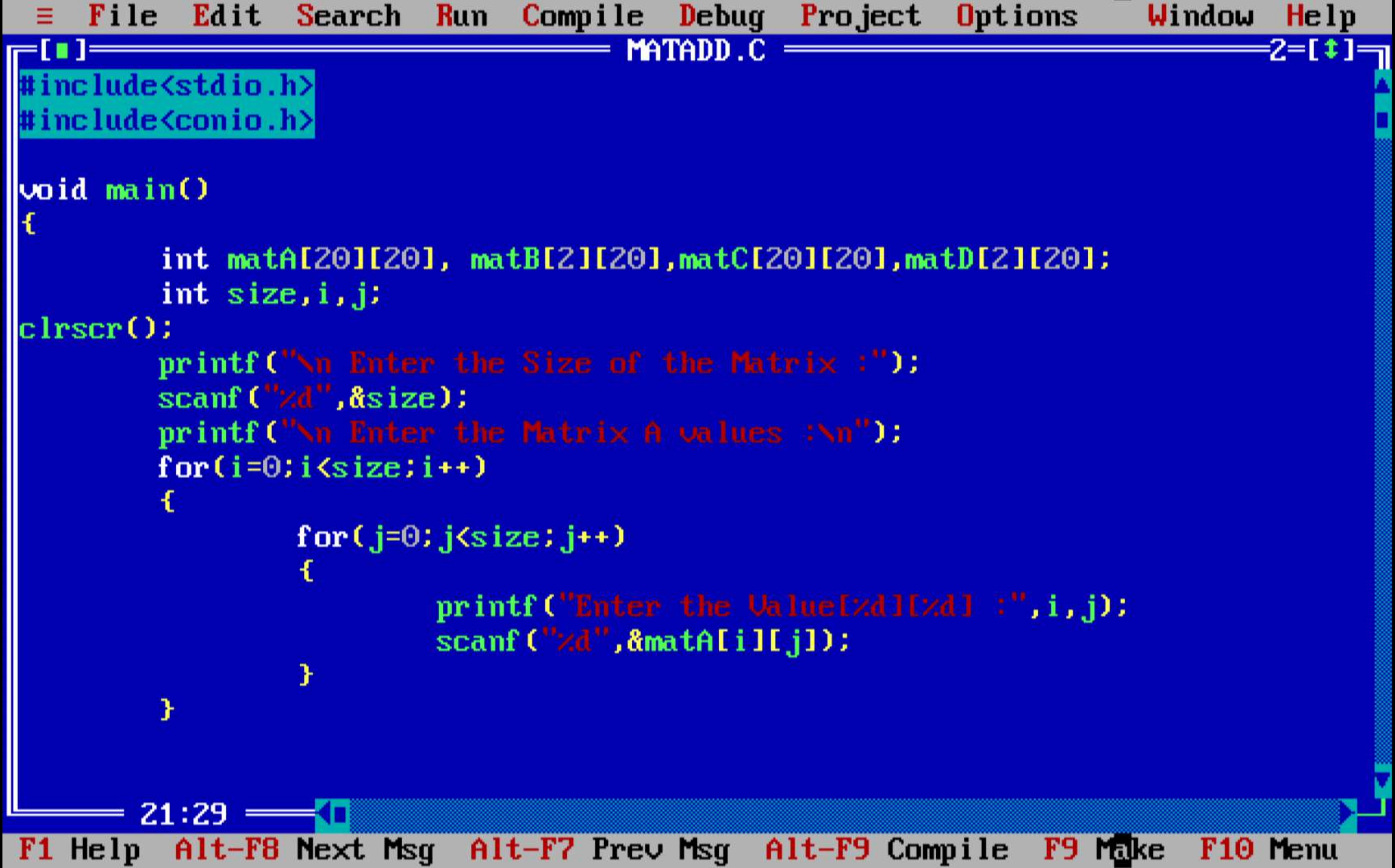
```
Enter the Size of the Matrix:10
Enter the Values :-10
Enter the Values :-20
Enter the Values :-30
Enter the Values :10
Enter the Values :20
Enter the Values :24
Enter the Values :30
Enter the Values :40
Enter the Values :50
Enter the Values :-5
 Sum of the Positive numbers in an Array is : 174
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] ODDARRAY.C 1=[↕]
20:31
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

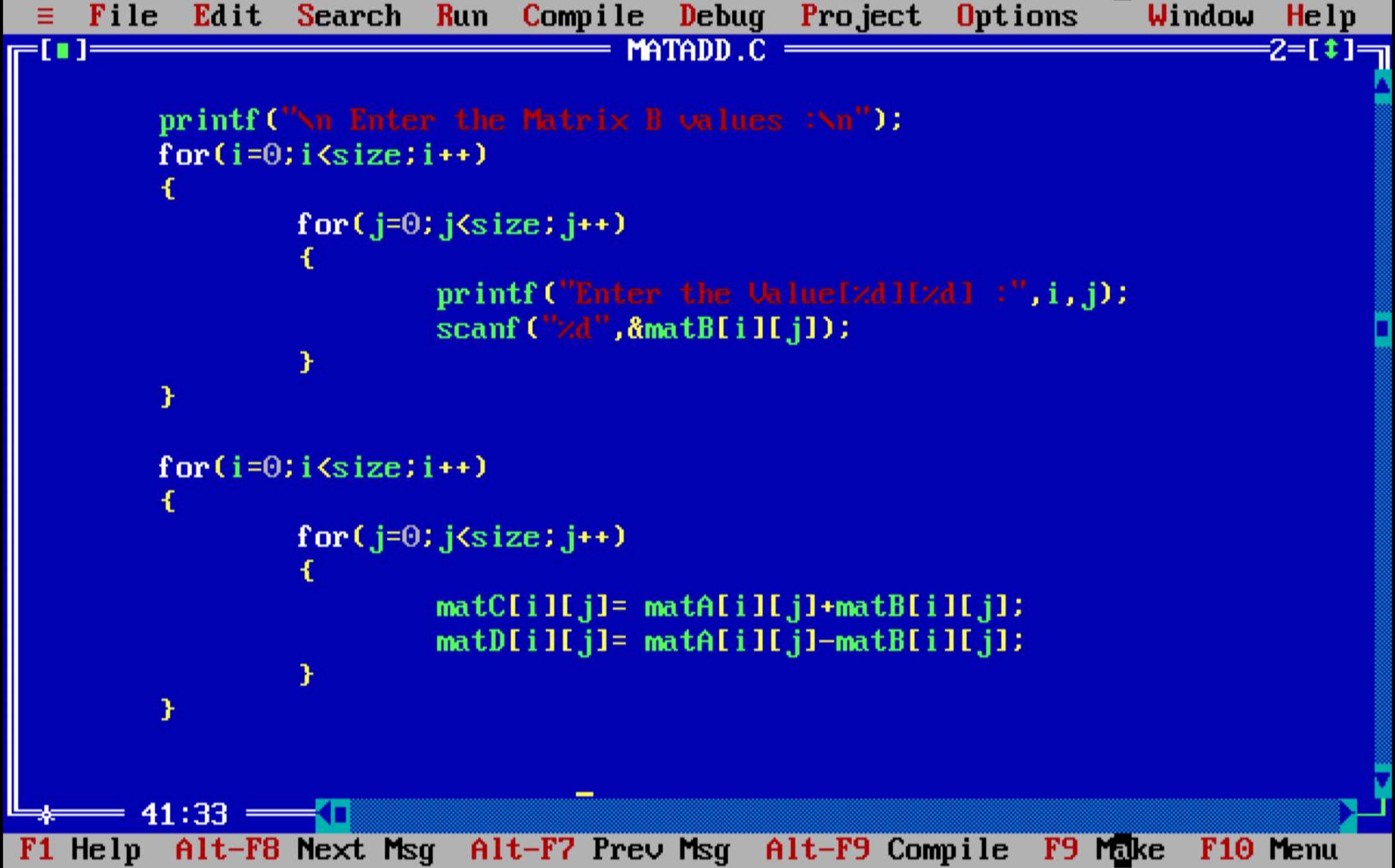
```
void main()
{
        int numbers[30],size,i, oddcount=0,evencount=0;
clrscr();
        printf("Enter the Size of the Matrix:");
        scanf("%d",&size);
        for(i=0;i<size;i++)
        {
                printf("Enter the Values :");
                scanf("%d",&numbers[i]);
                if((numbers[i] % 2)==0)
                        evencount++;
                else
                        oddcount++;
        }
        printf("\n Total no of Odd Numbers in an Array is %d",oddcount);
        printf("\n Total no of Even Numbers in an Array is %d",evencount);
getch();
}
```



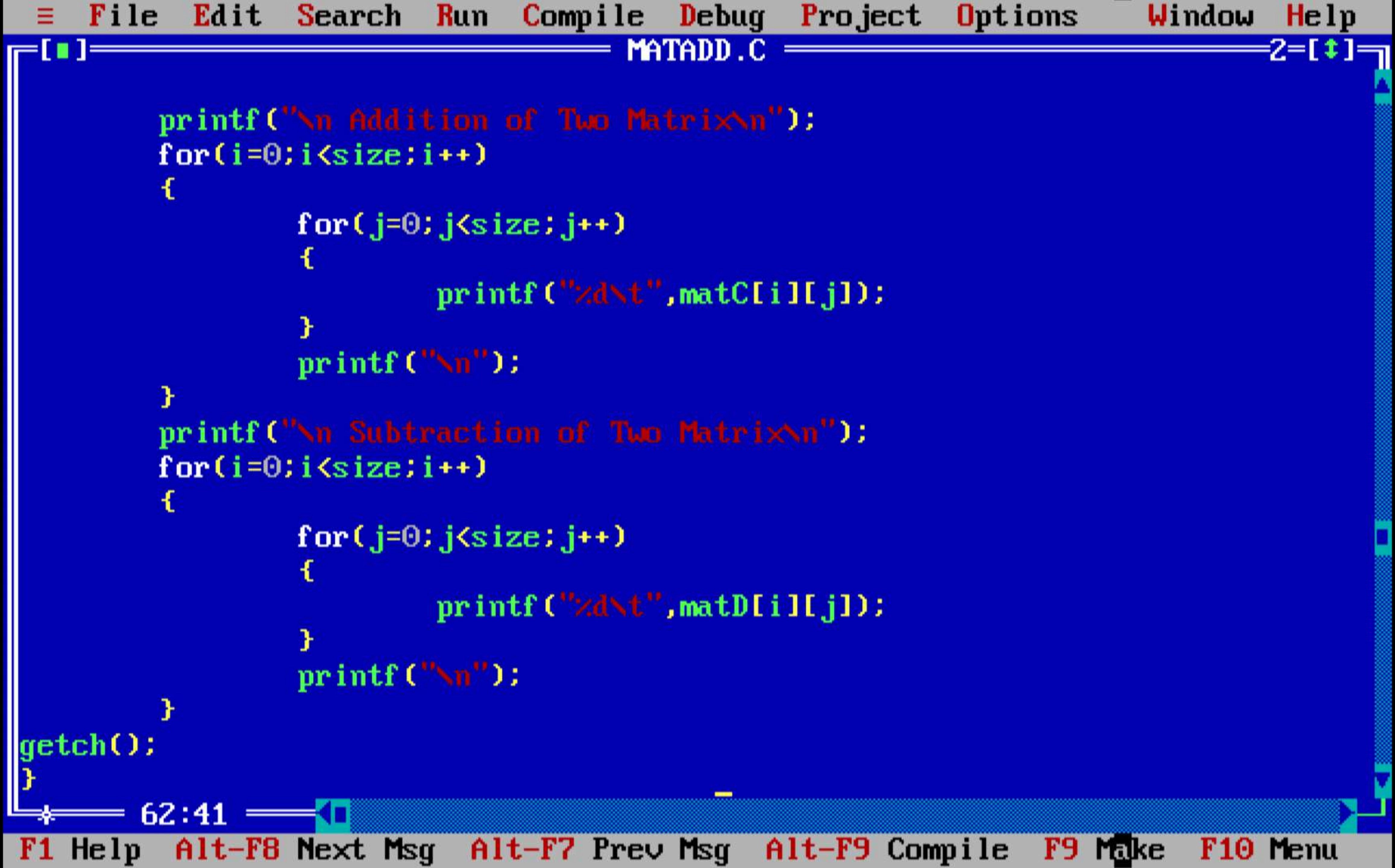
```
Enter the Size of the Matrix:10
Enter the Values :01
Enter the Values :12
Enter the Values :23
Enter the Values :34
Enter the Values :45
Enter the Values :56
Enter the Values :67
Enter the Values :78
Enter the Values :89
Enter the Values :90

 Total number of Odd Numbers in an Array is : 5
 Total number of Even Numbers in an Array is : 5_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] MATADD.C 2=[↕]
21:29
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

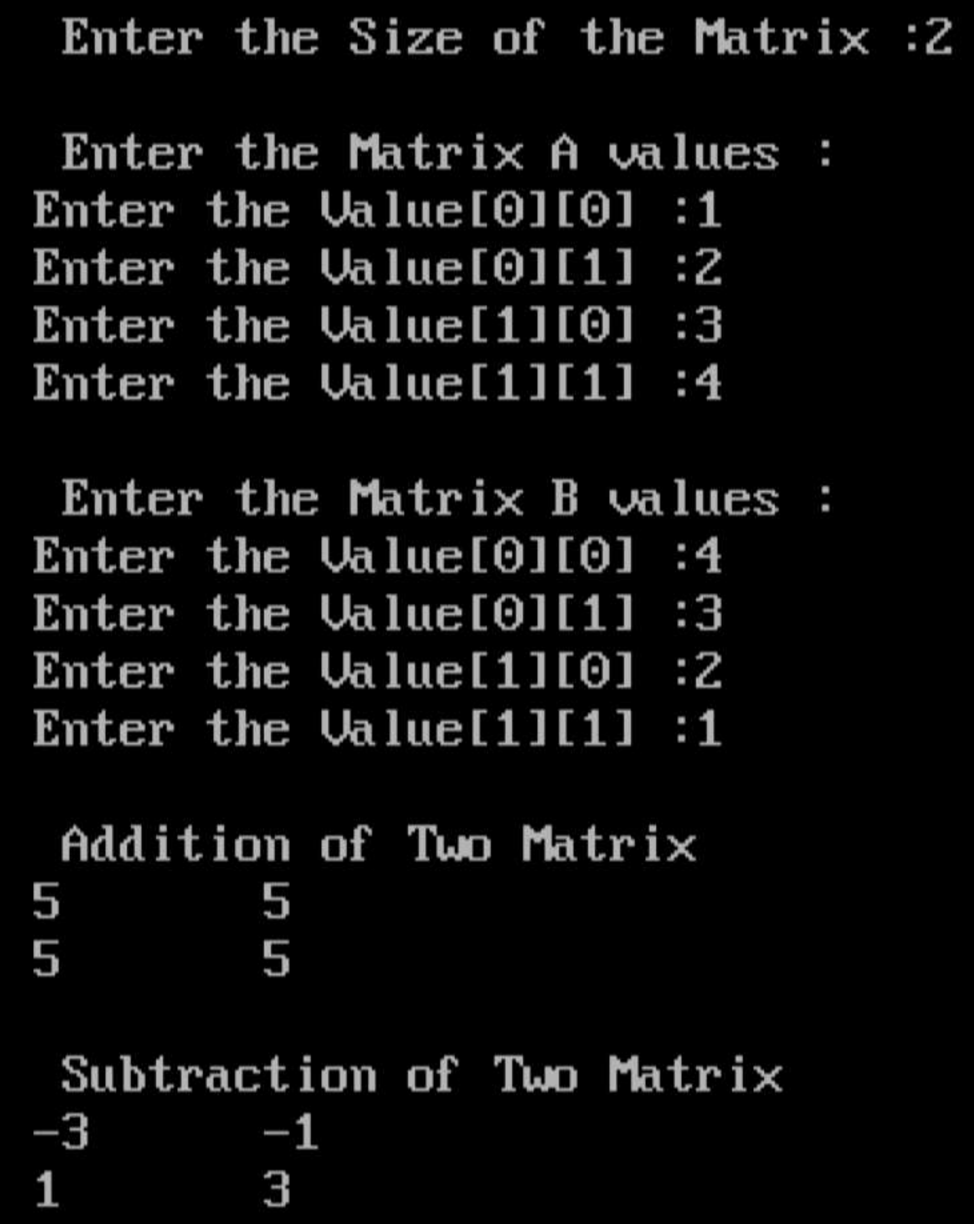
```
#include<stdio.h>
#include<conio.h>

void main()
{
        int matA[20][20], matB[2][20],matC[20][20],matD[2][20];
        int size,i,j;
clrscr();
        printf("\n Enter the Size of the Matrix :");
        scanf("%d",&size);
        printf("\n Enter the Matrix A values :\n");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        printf("Enter the Value[%d][%d] :",i,j);
                        scanf("%d",&matA[i][j]);
                }
        }
```


≡ File Edit Search Run Compile Debug Project Options Window Help
MATADD.C
2
41:33
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
    printf("\n Enter the Matrix B values :\n");
    for(i=0;i<size;i++)
    {
            for(j=0;j<size;j++)
            {
                    printf("Enter the Value[%d][%d] :",i,j);
                    scanf("%d",&matB[i][j]);
            }
    }

    for(i=0;i<size;i++)
    {
            for(j=0;j<size;j++)
            {
                    matC[i][j]= matA[i][j]+matB[i][j];
                    matD[i][j]= matA[i][j]-matB[i][j];
            }
    }
```


≡ File Edit Search Run Compile Debug Project Options Window Help
MATADD.C
62:41
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
        printf("\n Addition of Two Matrix\n");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        printf("%d\t",matC[i][j]);
                }
                printf("\n");
        }
        printf("\n Subtraction of Two Matrix\n");
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        printf("%d\t",matD[i][j]);
                }
                printf("\n");
        }
getch();
}
```



```
 Enter the Size of the Matrix :2

 Enter the Matrix A values :
Enter the Value[0][0] :1
Enter the Value[0][1] :2
Enter the Value[1][0] :3
Enter the Value[1][1] :4

 Enter the Matrix B values :
Enter the Value[0][0] :4
Enter the Value[0][1] :3
Enter the Value[1][0] :2
Enter the Value[1][1] :1

 Addition of Two Matrix
5       5
5       5

 Subtraction of Two Matrix
-3      -1
1       3
```

FUNCTIONS IN C

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

24-01-2022

III Bsc Mathematics

Functions

- A function is a group of statements that together perform a task.
- For example, Let’s consider you are writing a large C program and, in that program you want to do a particular task several number of times, like displaying value from 1 to 10, in order to do that you have to write few lines of code and you need to repeat these lines every time you display values.
- Another way of doing this is that you write these lines inside a function and call that function every time you want to display values. This would make you code simple, readable and reusable.

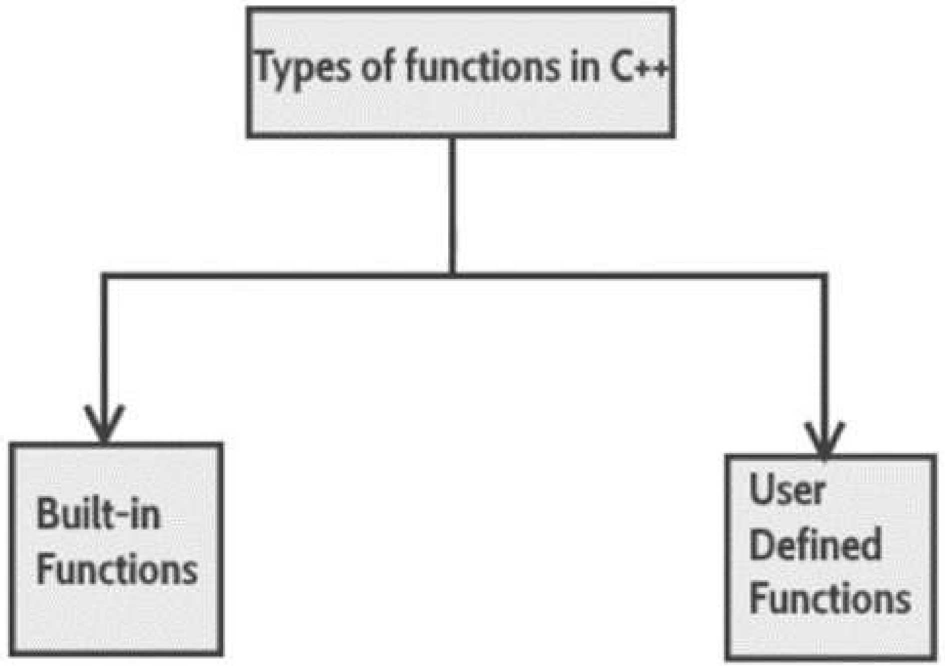
- A large program can typically be split into small sub-programs (blocks) called as functions where each sub-program can perform some specific functionality.
- Functions reduce the size and complexity of a program, makes it easier to understand, test, and check for errors.
- The functions which are available by default known as "Built-in" functions and user can create their own functions known as "User-defined" functions.
- To reduce size and complexity of the program we use Functions. The programmers can make use of sub programs either writing their own functions or calling them from standard library.
- Few lines of code may be repeatedly used in different contexts. Duplication of the same code can be eliminated by using functions which improves the maintenance and reduce program size.
- Some functions can be called multiple times with different inputs.

- Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.
- Syntax of Function

```
return_type function_name (parameter_list)
{
   //C++ Statements
}
```

Types of Function

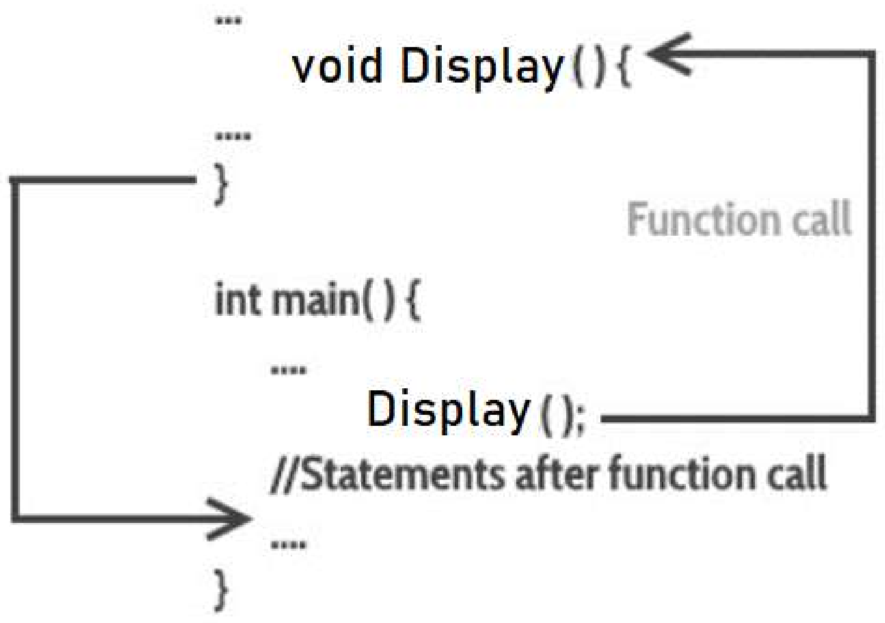
- C++ Supports two types of function.
  - Built in Functions
  - User Defined Functions



- **Built-in functions** are also known as library functions.
- Functions such as puts(), gets(), printf(), scanf() etc are standard library functions. These functions are already defined in header files (files with .h extensions are called header files such as stdio.h), so we just call them whenever there is a need to use them.
- For example, printf() function is defined in <stdio.h> header file so in order to use the printf() function, we need to include the <stdio.h> header file in our program using #include <stdio.h>.

User Defined Functions

- **User Defined Functions:** The functions that we declare and write in our programs are user-defined functions.



```
...
    void Display(){
....
}

int main(){
    ....
        Display();
    //Statements after function call
    ....
}
```

Function Declaration

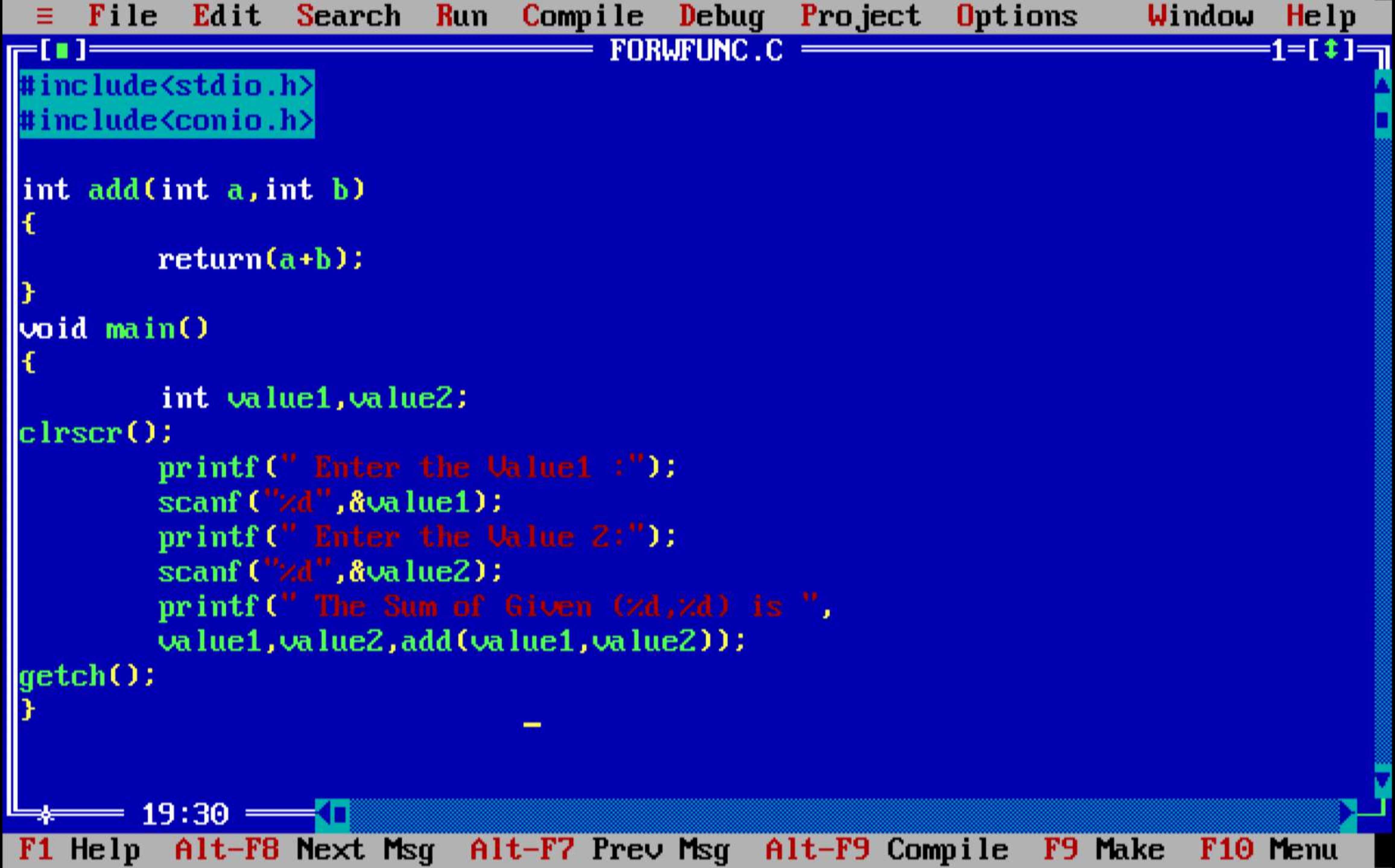
- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following parts –

return_type function_name(parameter list);

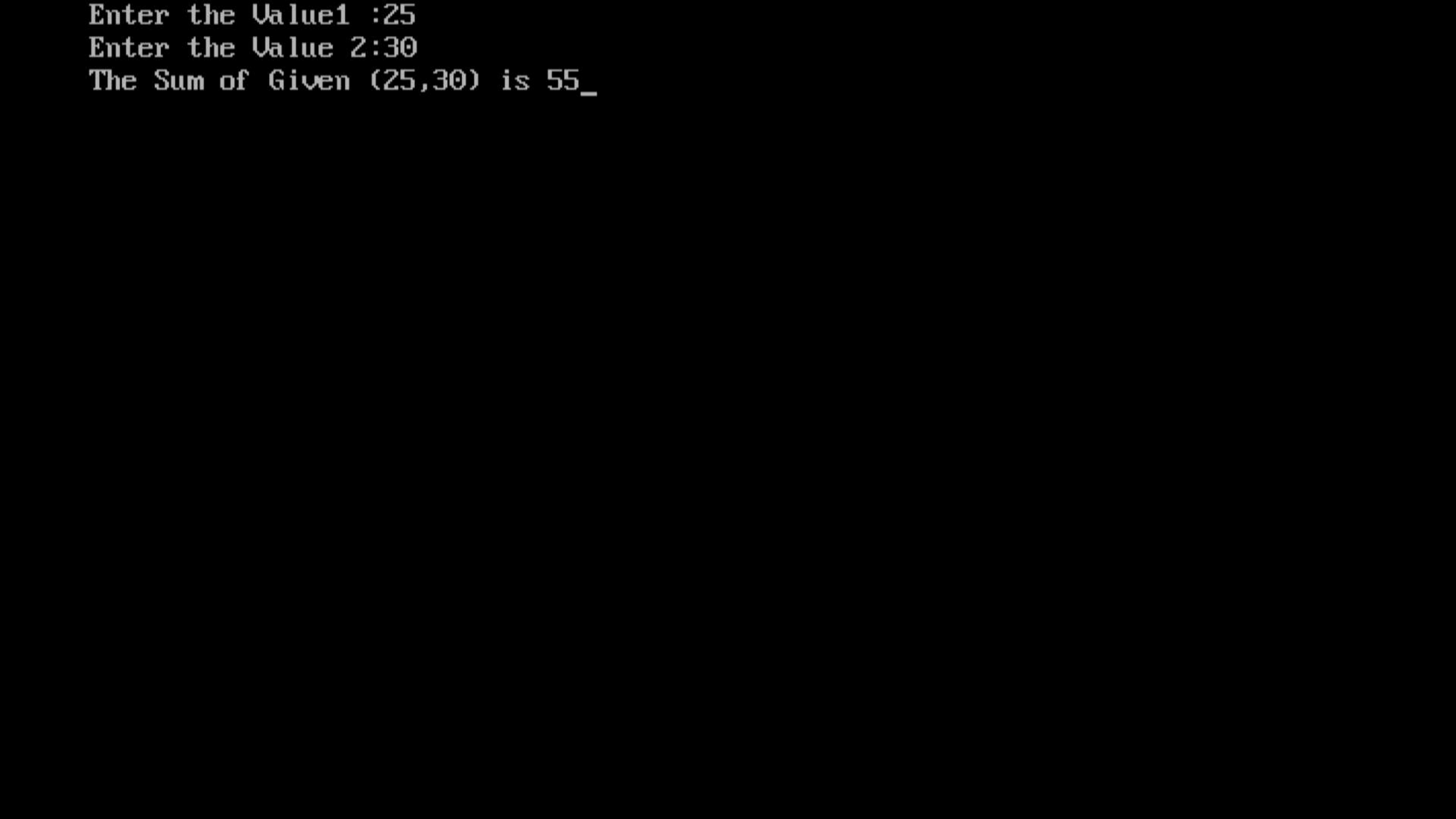
- For the above defined function max(), following is the function declaration –

int max(int, int);

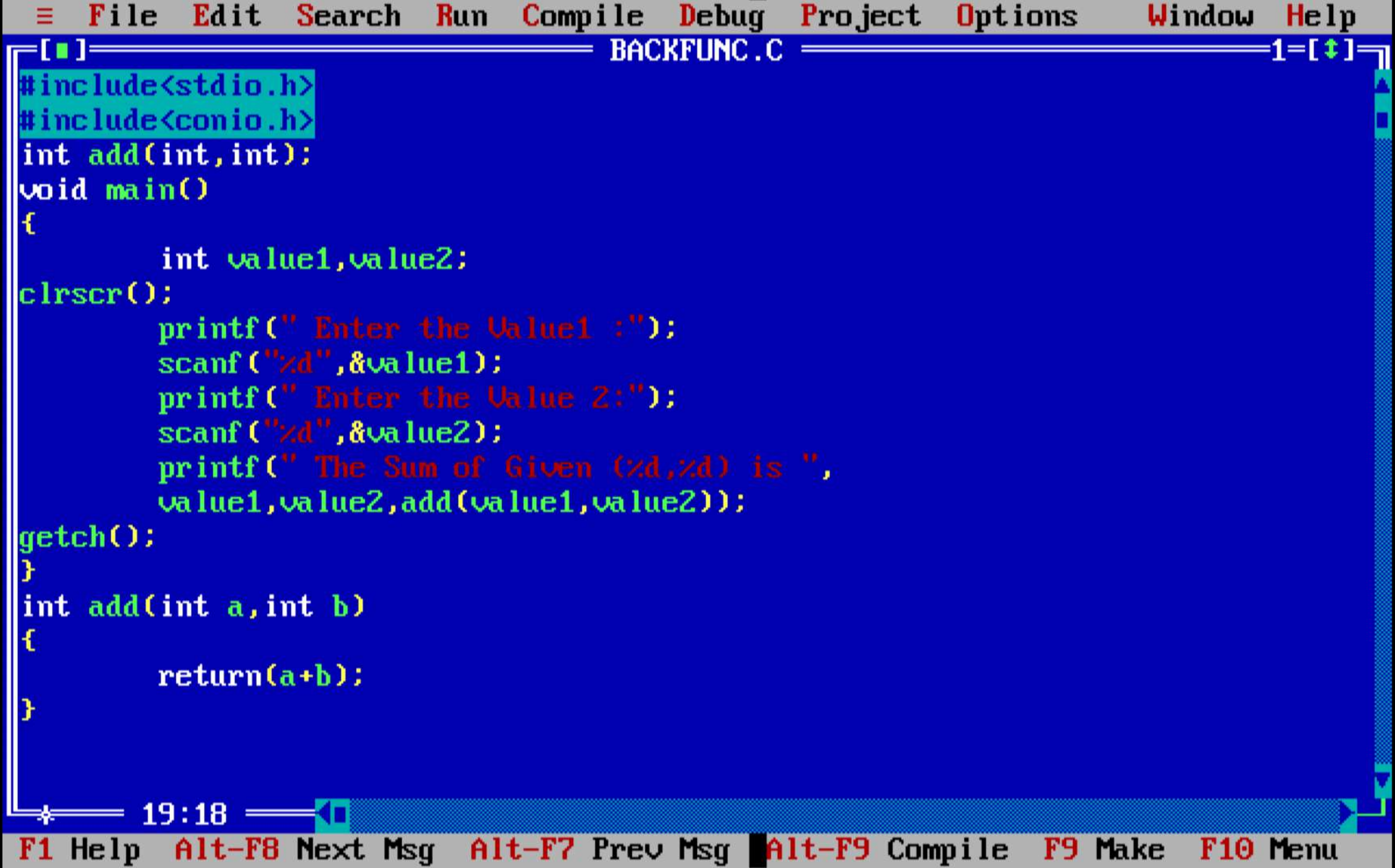
- Parameter names are not important in function declaration only their type is required, so following is also valid declaration


≡ File Edit Search Run Compile Debug Project Options Window Help
FORWFUNC.C
1
19:30
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

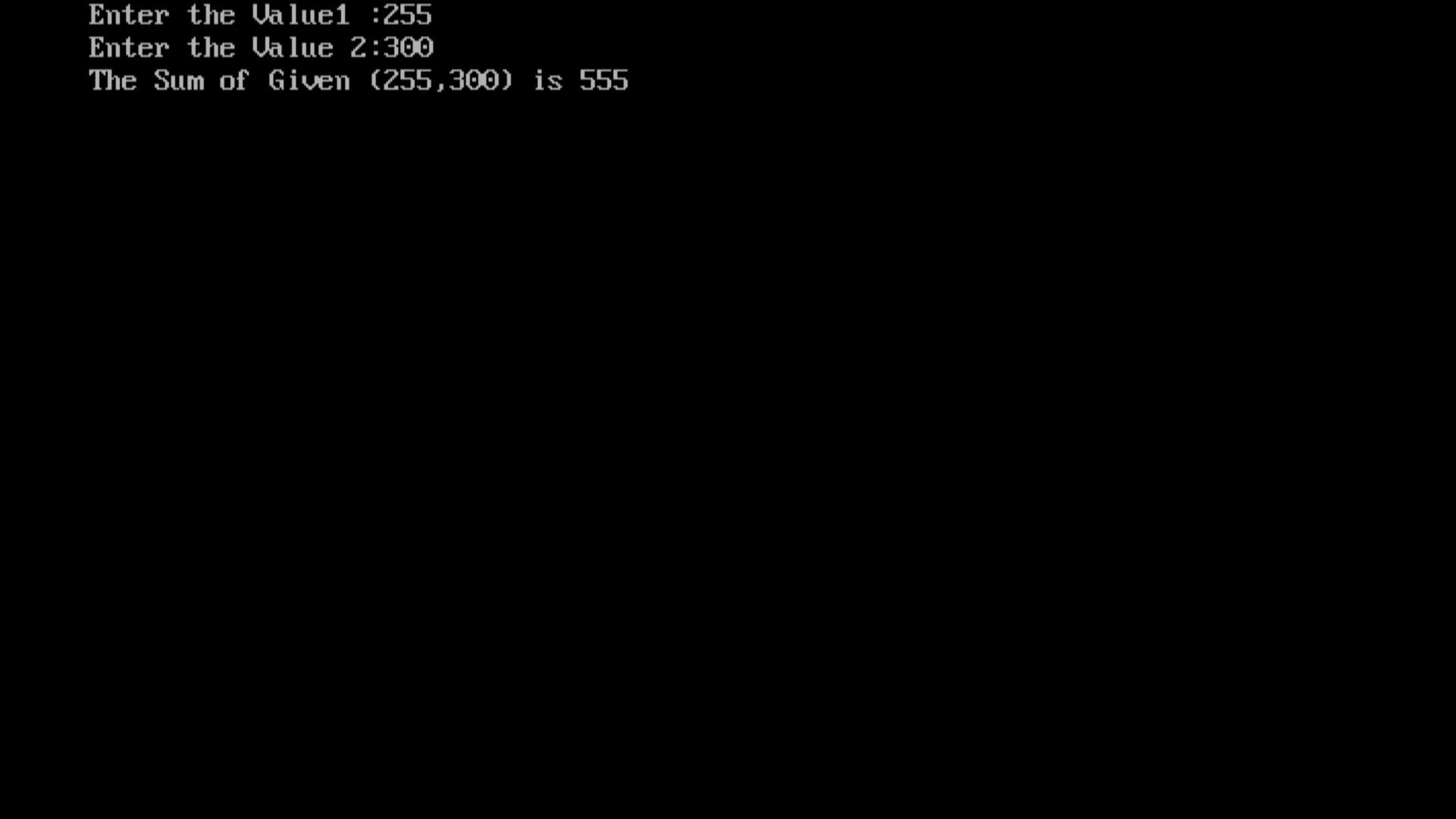
```
#include<stdio.h>
#include<conio.h>

int add(int a,int b)
{
        return(a+b);
}
void main()
{
        int value1,value2;
clrscr();
        printf(" Enter the Value1 :");
        scanf("%d",&value1);
        printf(" Enter the Value 2:");
        scanf("%d",&value2);
        printf(" The Sum of Given (%d,%d) is ",
        value1,value2,add(value1,value2));
getch();
}
```



```
Enter the Value1 :25
Enter the Value 2:30
The Sum of Given (25,30) is 55_
```


≡ File Edit Search Run Compile Debug Project Options Window Help
BACKFUNC.C
1
19:18
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<stdio.h>
#include<conio.h>
int add(int,int);
void main()
{
        int value1,value2;
clrscr();
        printf(" Enter the Value1 :");
        scanf("%d",&value1);
        printf(" Enter the Value 2:");
        scanf("%d",&value2);
        printf(" The Sum of Given (%d,%d) is ",
        value1,value2,add(value1,value2));
getch();
}
int add(int a,int b)
{
        return(a+b);
}
```



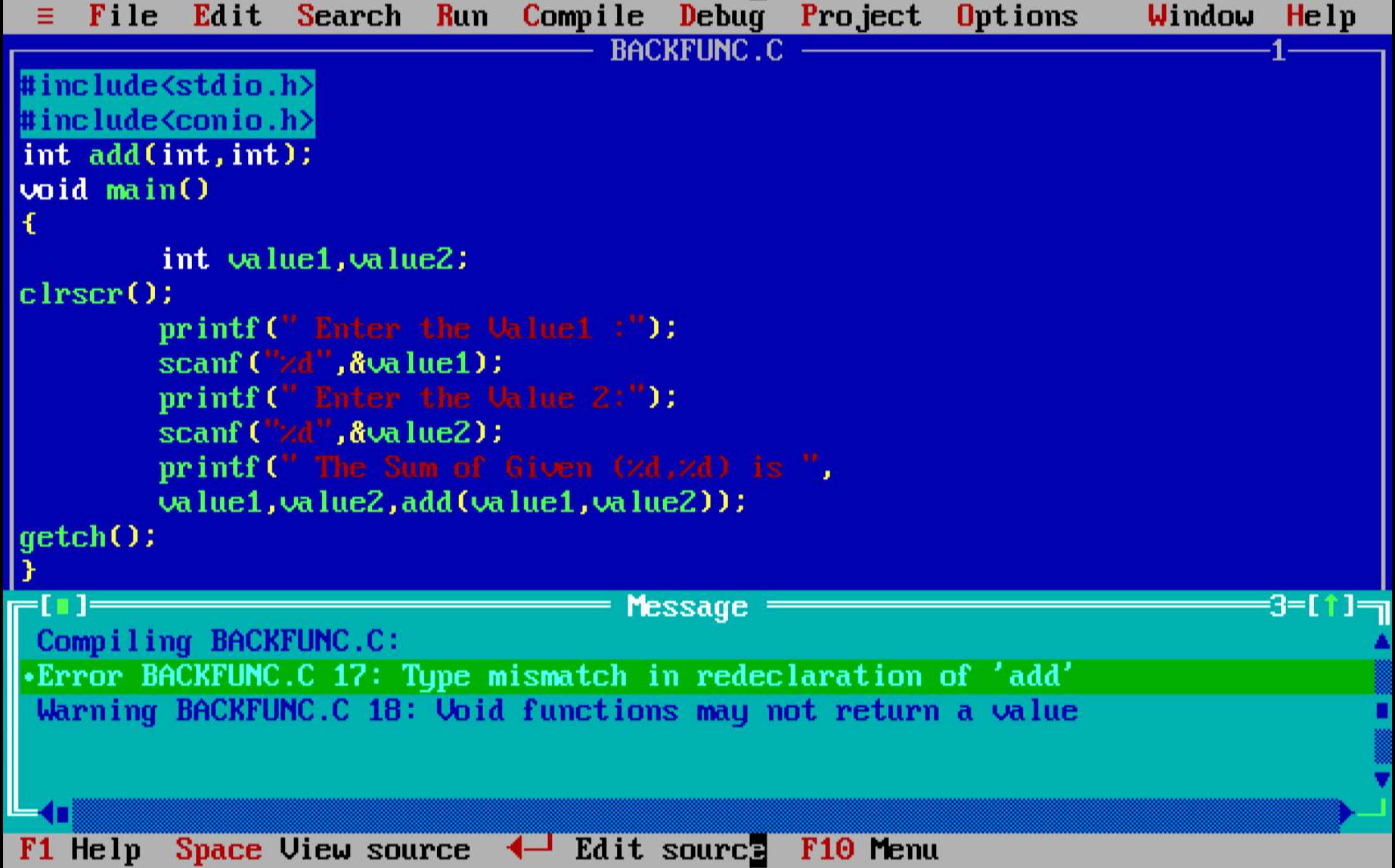
```
Enter the Value1 :255
Enter the Value 2:300
The Sum of Given (255,300) is 555
```

Function Definition

- A **Function Definition** provides the actual body of the function.
- The general form of a C function definition is as follows −

```
return_type function_name( parameter list )
{
   body of the function
}
```

- A C function definition consists of a function header and a function body. Here are all the parts of a function
  - **Return Type** − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
  - **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
  - **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
  - **Function Body** − The function body contains a collection of statements that define what the function does.


≡ File Edit Search Run Compile Debug Project Options Window Help
BACKFUNC.C
1
Message
3
F1 Help Space View source ↵ Edit source F10 Menu

```
#include<stdio.h>
#include<conio.h>
int add(int,int);
void main()
{
        int value1,value2;
clrscr();
        printf(" Enter the Value1 :");
        scanf("%d",&value1);
        printf(" Enter the Value 2:");
        scanf("%d",&value2);
        printf(" The Sum of Given (%d,%d) is ",
        value1,value2,add(value1,value2));
getch();
}
```

```
 Compiling BACKFUNC.C:
•Error BACKFUNC.C 17: Type mismatch in redeclaration of 'add'
 Warning BACKFUNC.C 18: Void functions may not return a value
```

Calling a Function

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

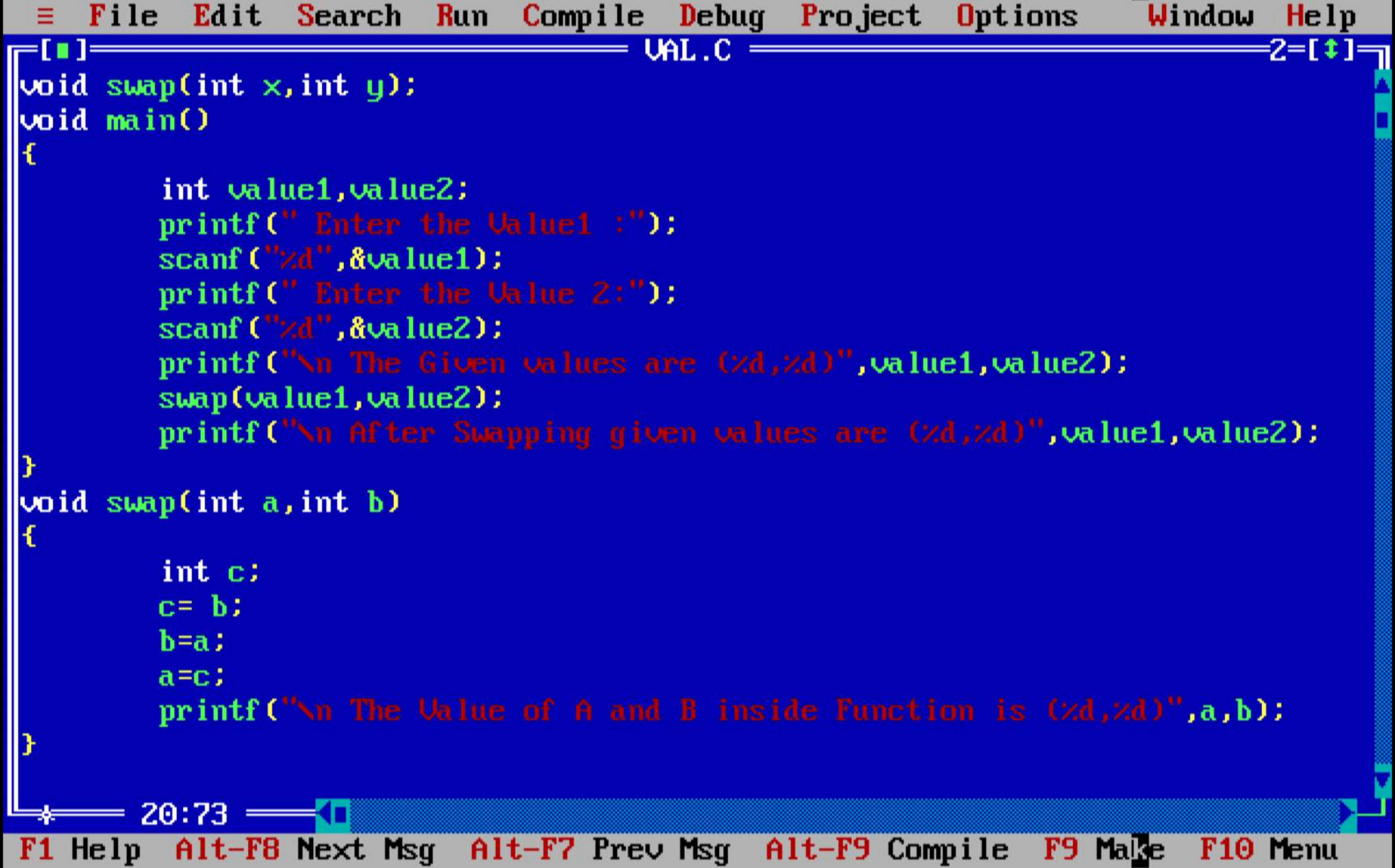
Function Arguments

- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.
- The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.
- While calling a function, there are two ways that arguments can be passed to a function –
  - Call by Value
  - Call by Reference

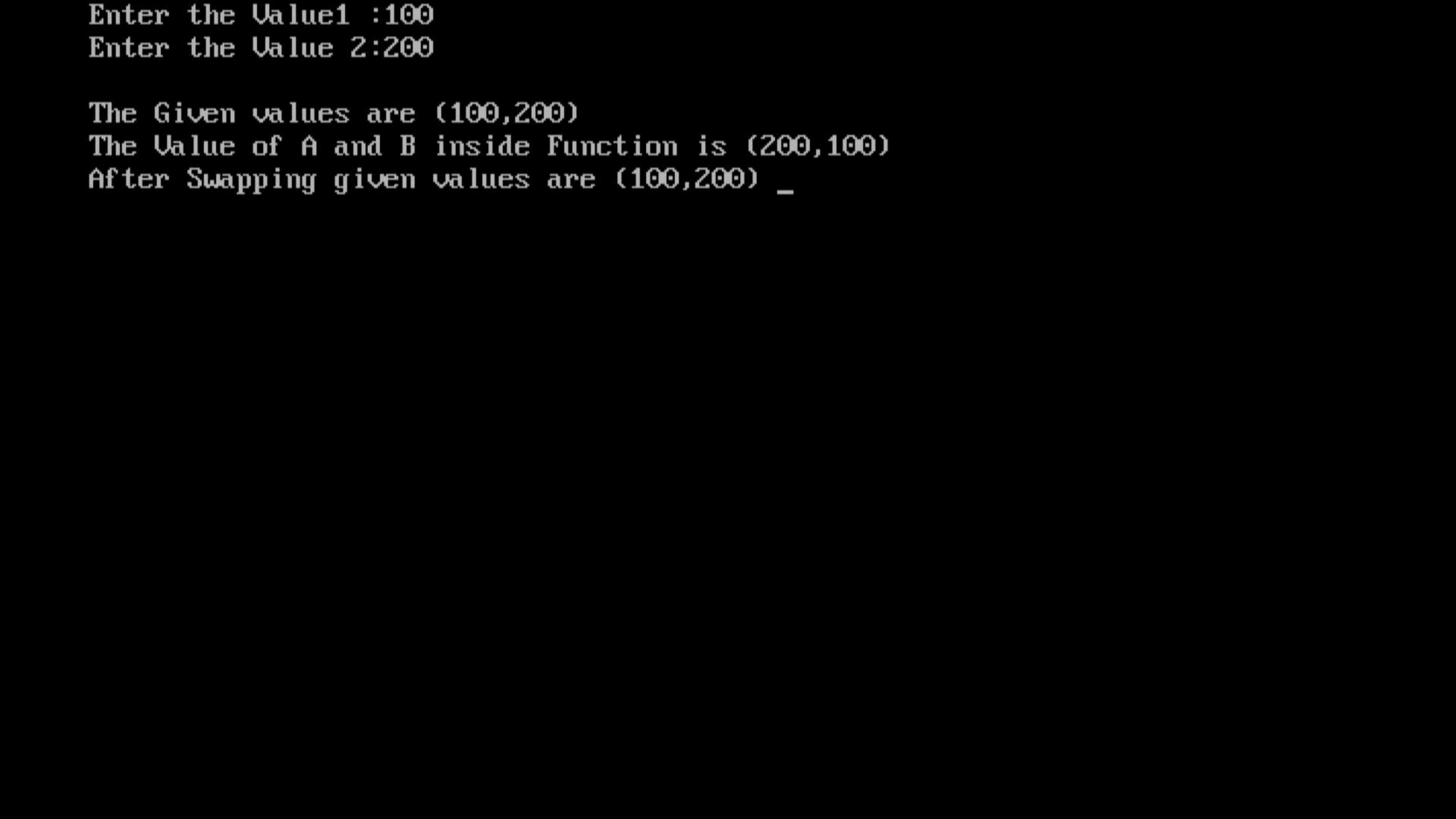
- **Call by Value :** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- **Call by Reference:** This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Call by Value

- The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- By default, C uses call by value to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function.


≡ File Edit Search Run Compile Debug Project Options Window Help
VAL.C
2
20:73
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

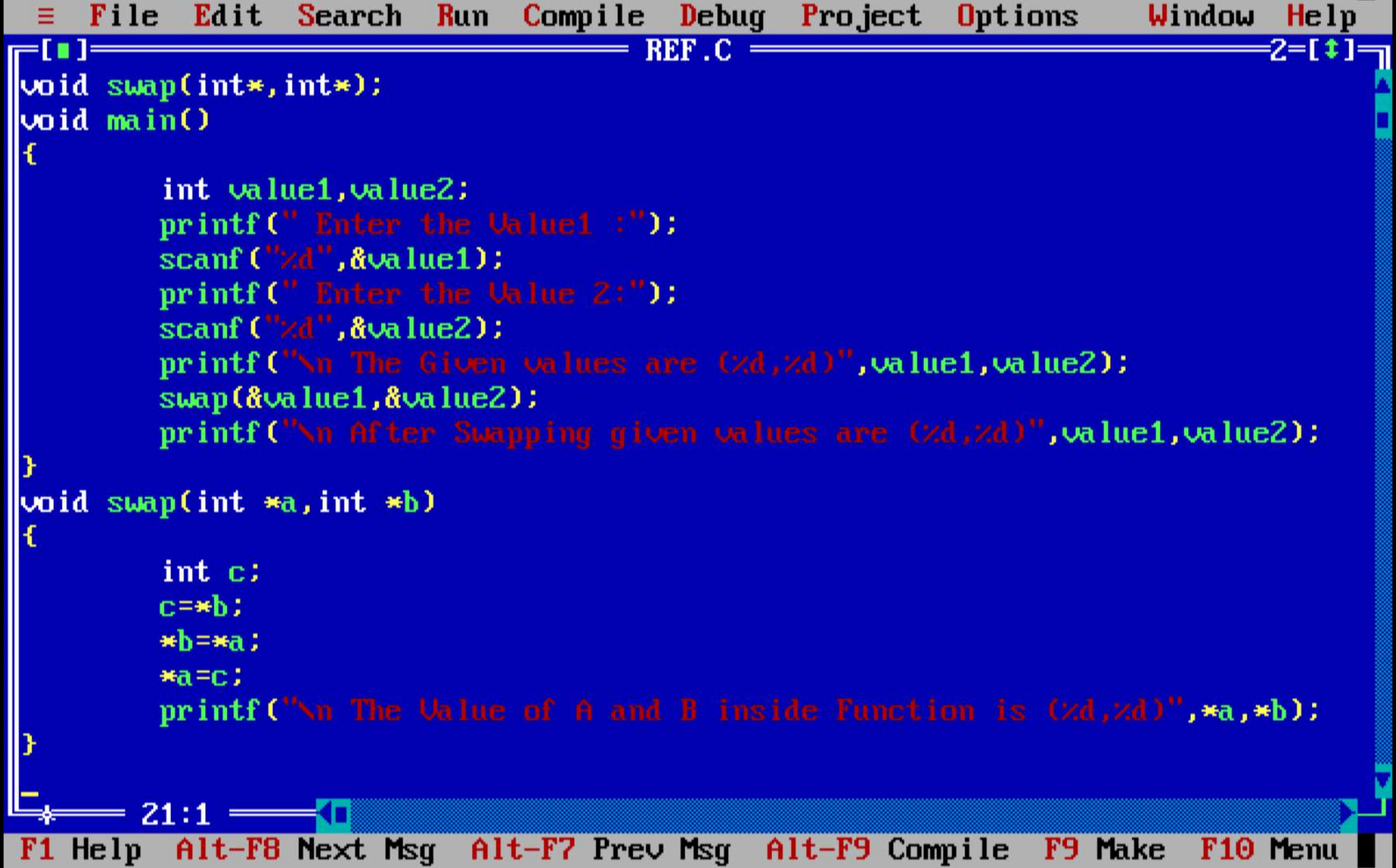
```
void swap(int x,int y);
void main()
{
        int value1,value2;
        printf(" Enter the Value1 :");
        scanf("%d",&value1);
        printf(" Enter the Value 2:");
        scanf("%d",&value2);
        printf("\n The Given values are (%d,%d)",value1,value2);
        swap(value1,value2);
        printf("\n After Swapping given values are (%d,%d)",value1,value2);
}
void swap(int a,int b)
{
        int c;
        c= b;
        b=a;
        a=c;
        printf("\n The Value of A and B inside Function is (%d,%d)",a,b);
}
```



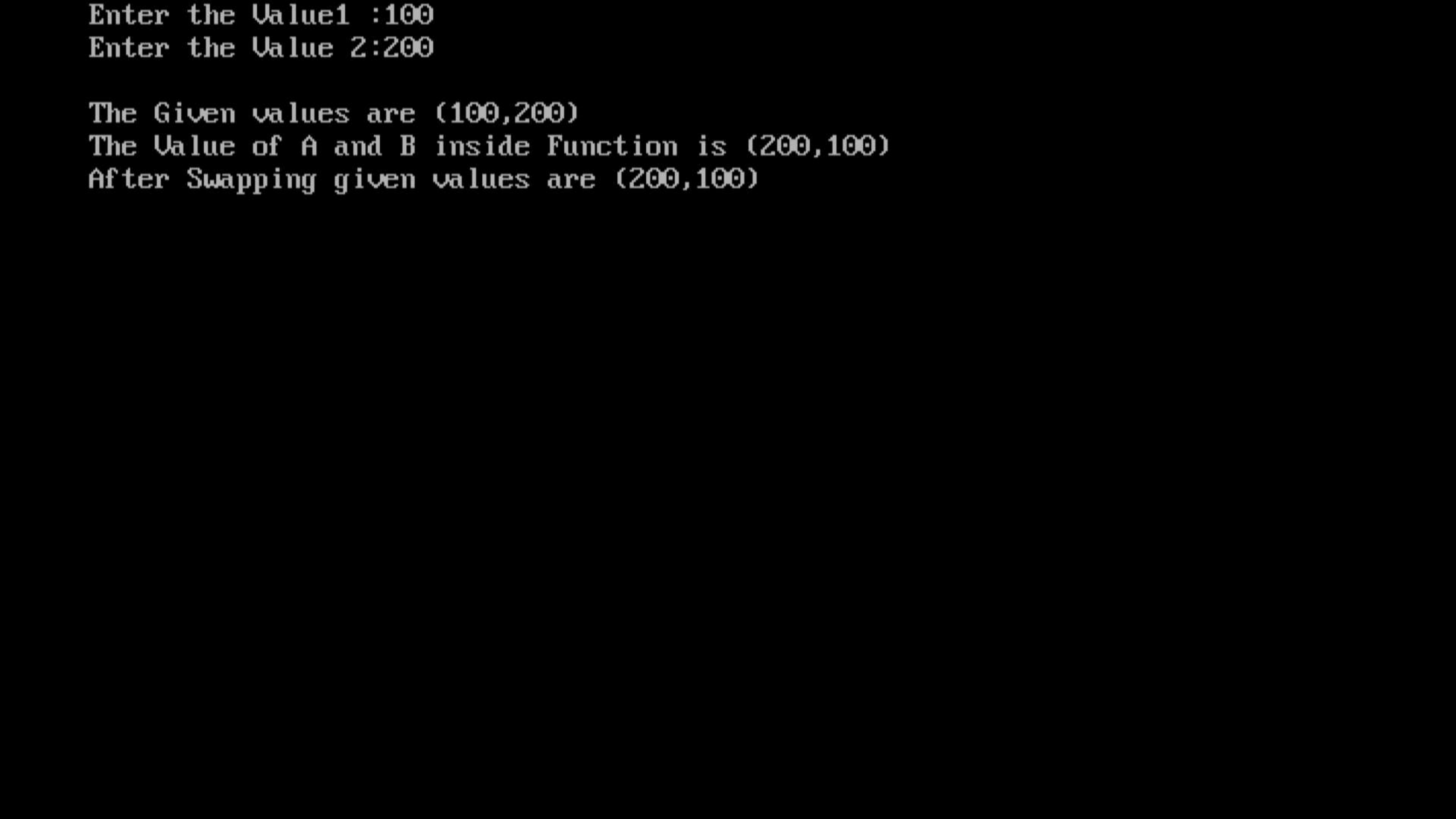
```
Enter the Value1 :100
Enter the Value 2:200

The Given values are (100,200)
The Value of A and B inside Function is (200,100)
After Swapping given values are (100,200) _
```

Call by Reference

- The call by reference method of passing arguments to a function copies the reference of an argument into the formal parameter.
- Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the passed argument.
- To pass the value by reference, argument reference is passed to the functions just like any other value. So accordingly you need to declare the function parameters as reference types as in the following function swap(), which exchanges the values of the two integer variables pointed to by its arguments.


≡ File Edit Search Run Compile Debug Project Options Window Help
REF.C
2
21:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
void swap(int*,int*);
void main()
{
        int value1,value2;
        printf(" Enter the Value1 :");
        scanf("%d",&value1);
        printf(" Enter the Value 2:");
        scanf("%d",&value2);
        printf("\n The Given values are (%d,%d)",value1,value2);
        swap(&value1,&value2);
        printf("\n After Swapping given values are (%d,%d)",value1,value2);
}
void swap(int *a,int *b)
{
        int c;
        c=*b;
        *b=*a;
        *a=c;
        printf("\n The Value of A and B inside Function is (%d,%d)",*a,*b);
}
```



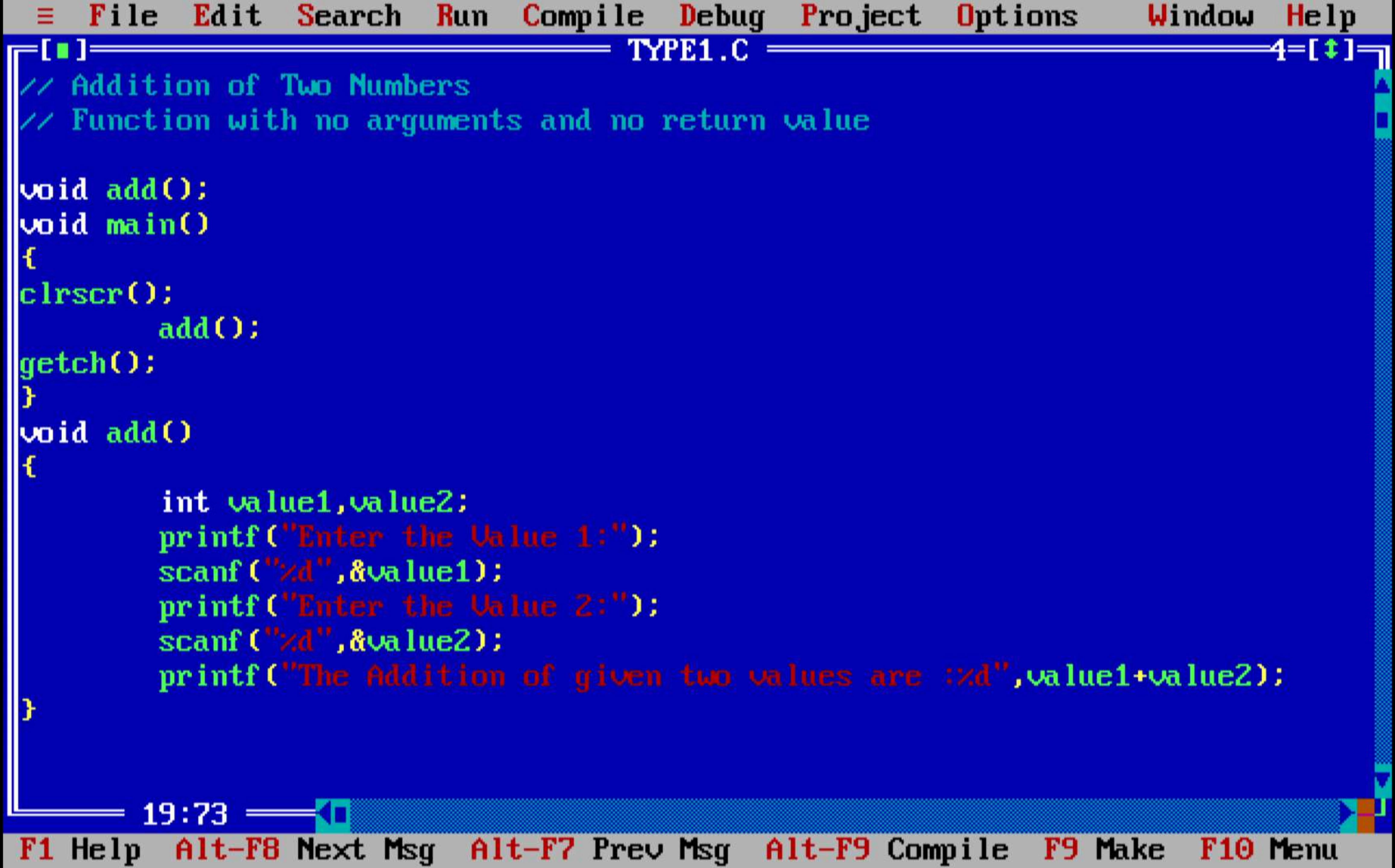
```
Enter the Value1 :100
Enter the Value 2:200

The Given values are (100,200)
The Value of A and B inside Function is (200,100)
After Swapping given values are (200,100)
```

Types of Function

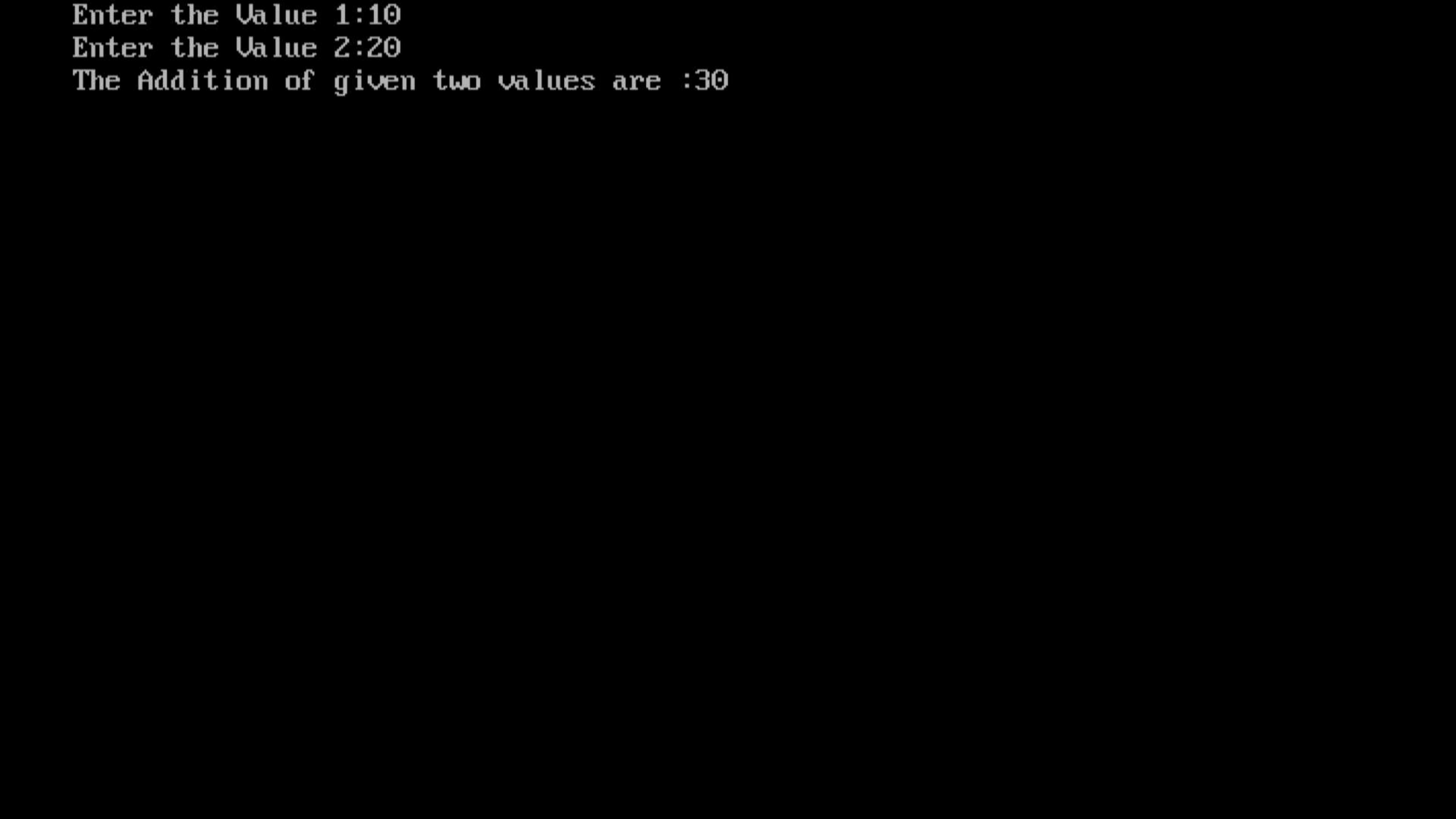
- A function depending upon the arguments present or not and whether a value is returned or not , maybe classified as
  - Function with no arguments and no return values.
  - Function with no arguments and having return values.
  - Function with arguments and no return values.
  - Function with arguments and having return values.

Function with no arguments and no return values

- When a function has no arguments, it does not receive any data from the calling function, similarly when a function has no return values, the calling function does not receive any data from the called function.
- Hence, there is no data transfer between the calling function and the called function.
- Those functions are used to perform any operation, they read the input and display the output in same block.
- In the following example, main() is the calling function which calls the function add().
- The function add() receives it data directly from keyboard and displays the output directly to the screen in the function itself, so there is no need of return statement.
- The closing brace of the function indicates the end of the execution of the function, thus returning the control back to the calling function.
- The keyword void is used before the function add() to indicate that there are no return values.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] TYPE1.C 4=[↕]
19:73
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

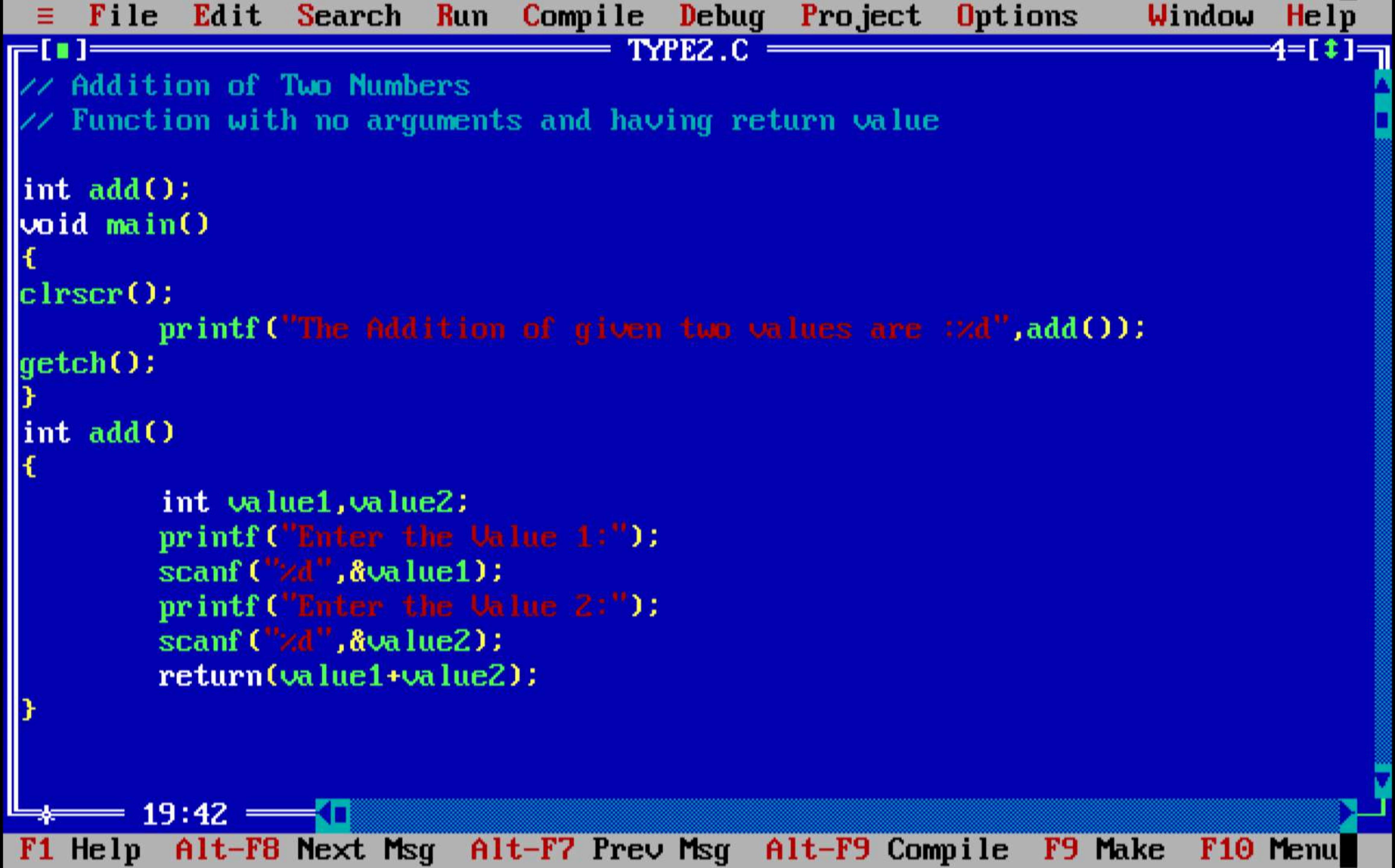
```
// Addition of Two Numbers
// Function with no arguments and no return value

void add();
void main()
{
clrscr();
        add();
getch();
}
void add()
{
        int value1,value2;
        printf("Enter the Value 1:");
        scanf("%d",&value1);
        printf("Enter the Value 2:");
        scanf("%d",&value2);
        printf("The Addition of given two values are :%d",value1+value2);
}
```



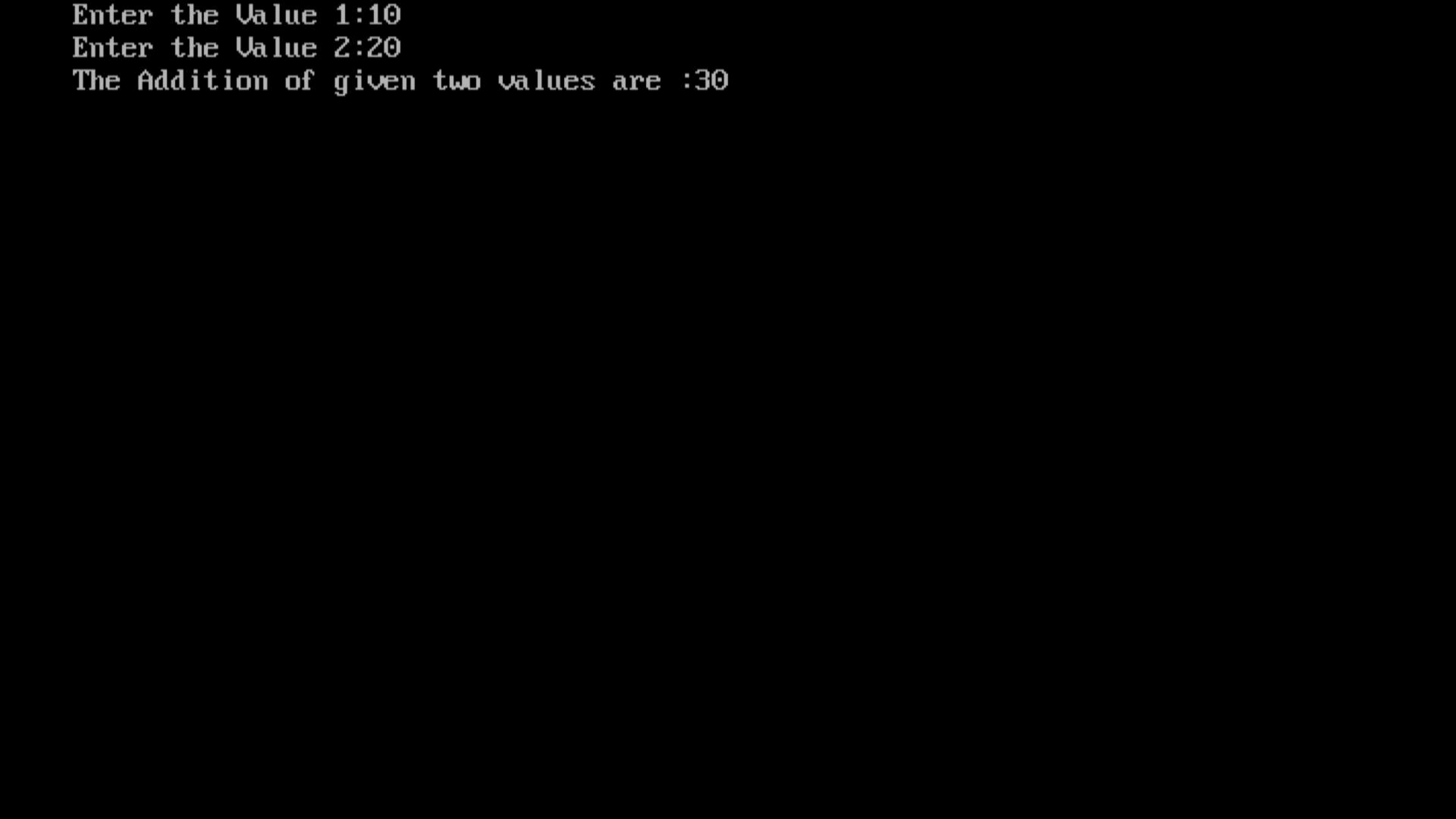
```
Enter the Value 1:10
Enter the Value 2:20
The Addition of given two values are :30
```

Function with no argument and having return values

- When a function has no arguments, it does not receive any data from the calling function, but it can do some process and then return the result to the called function. Hence, there is data transfer between the calling function and the called function.
- In the below program, main() is the calling function which calls the function add().
- The function add() contains no arguments and receives it data directly from keyboard.
- The return statement is employed in this function to return the sum of given two numbers calculated and the result is displayed from the main() to the standard output device.
- The keyword int is used before the function add() to indicate that it returns a value of type int to the called function.


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] TYPE2.C 4=[↕]
19:42
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

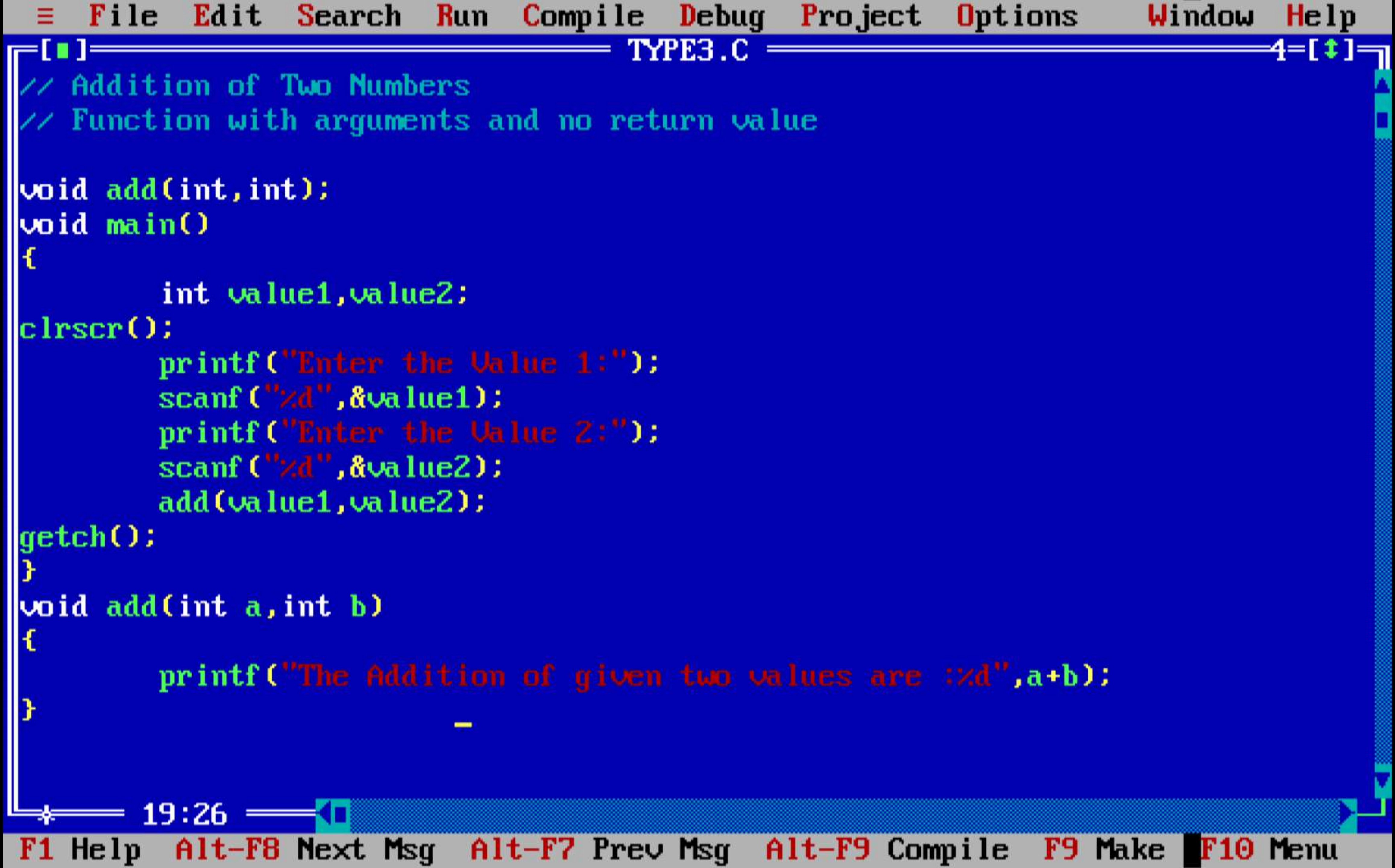
```
// Addition of Two Numbers
// Function with no arguments and having return value

int add();
void main()
{
clrscr();
        printf("The Addition of given two values are :%d",add());
getch();
}
int add()
{
        int value1,value2;
        printf("Enter the Value 1:");
        scanf("%d",&value1);
        printf("Enter the Value 2:");
        scanf("%d",&value2);
        return(value1+value2);
}
```



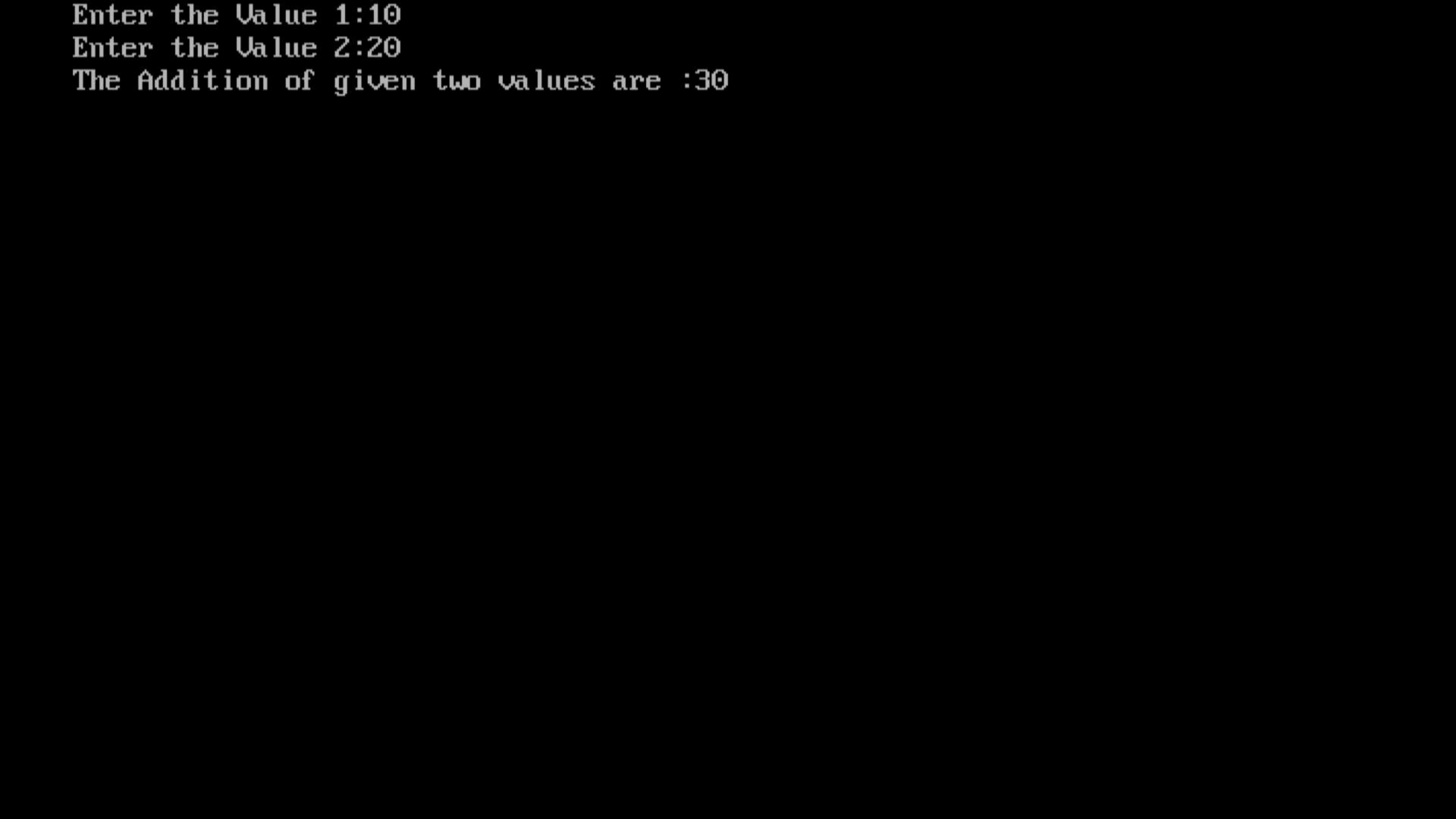
```
Enter the Value 1:10
Enter the Value 2:20
The Addition of given two values are :30
```

Function with arguments but no return value

- When a function has arguments, it receive any data from the calling function but it returns no values.
- In the below program, main() is the calling function which calls the function add().
- The function receives two arguments of type int from the calling function (main()) after calculating the sum of two numbers, it displays the output directly to the screen in the function itself, so there is no need of return statement.
- The closing brace of the function indicates the end of the execution of the function, thus returning the control back to the calling function.
- The keyword void is used before the function add() to indicate that there are no return values.


≡ File Edit Search Run Compile Debug Project Options Window Help
TYPE3.C
4
19:26
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

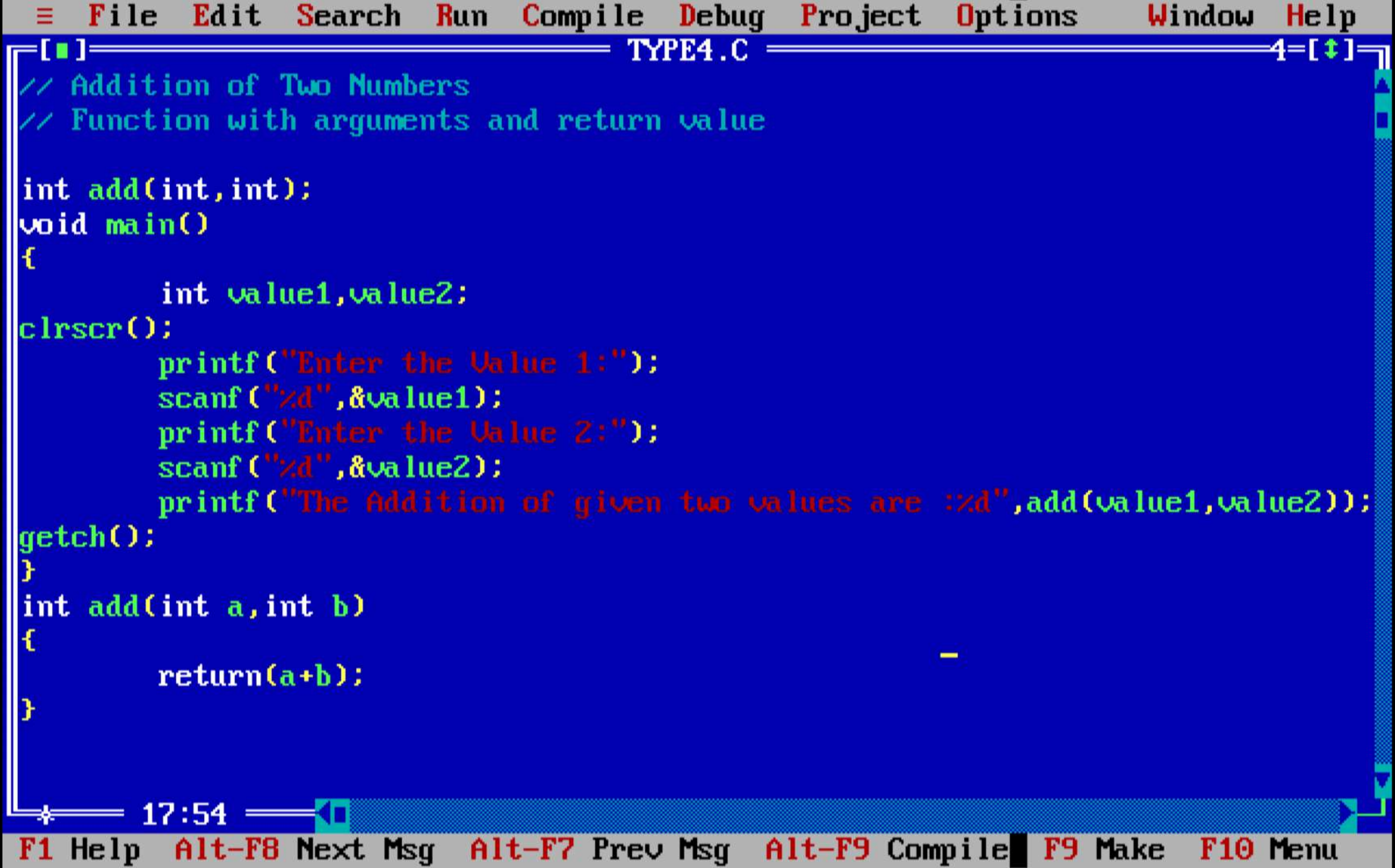
```
// Addition of Two Numbers
// Function with arguments and no return value

void add(int,int);
void main()
{
        int value1,value2;
clrscr();
        printf("Enter the Value 1:");
        scanf("%d",&value1);
        printf("Enter the Value 2:");
        scanf("%d",&value2);
        add(value1,value2);
getch();
}
void add(int a,int b)
{
        printf("The Addition of given two values are :%d",a+b);
}
```



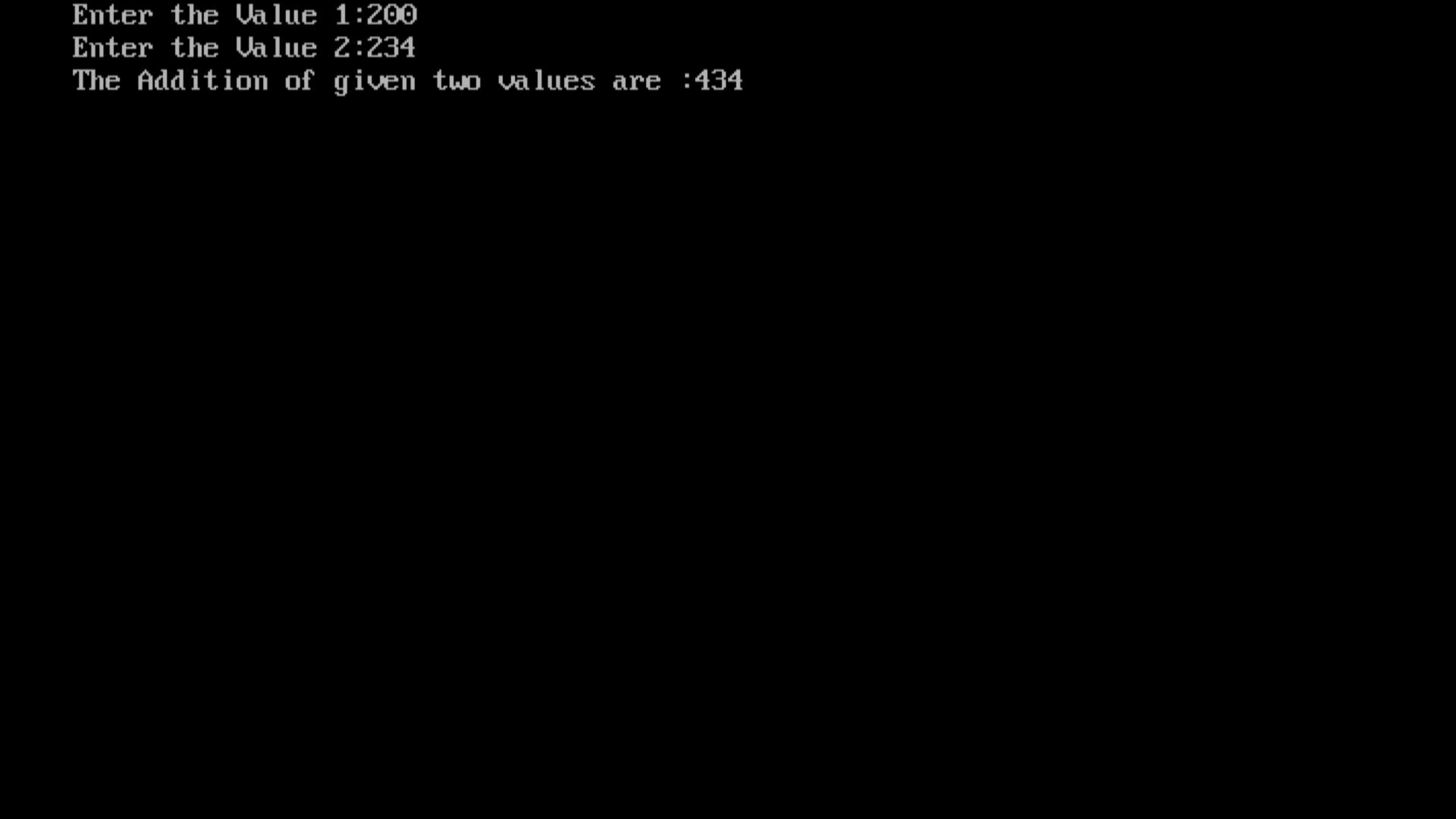
```
Enter the Value 1:10
Enter the Value 2:20
The Addition of given two values are :30
```

Function with arguments and return value

- Function with arguments and return value means both the calling function and called function will receive data from each other. It’s like a dual communication.
- In the below program, main() is the calling function which calls the function add().
- The function receives two arguments of type int from the calling function (main()) after calculating the sum of two numbers, The return statement is employed in this function to return the sum of given two numbers calculated and the result is displayed from the main() to the standard output device.
- The keyword int is used before the function add() to indicate that it returns a value of type int to the called function.


≡ File Edit Search Run Compile Debug Project Options Window Help
TYPE4.C
17:54
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```c
// Addition of Two Numbers
// Function with arguments and return value

int add(int,int);
void main()
{
        int value1,value2;
clrscr();
        printf("Enter the Value 1:");
        scanf("%d",&value1);
        printf("Enter the Value 2:");
        scanf("%d",&value2);
        printf("The Addition of given two values are :%d",add(value1,value2));
getch();
}
int add(int a,int b)
{
        return(a+b);
}
```



```
Enter the Value 1:200
Enter the Value 2:234
The Addition of given two values are :434
```

Constant Arguments

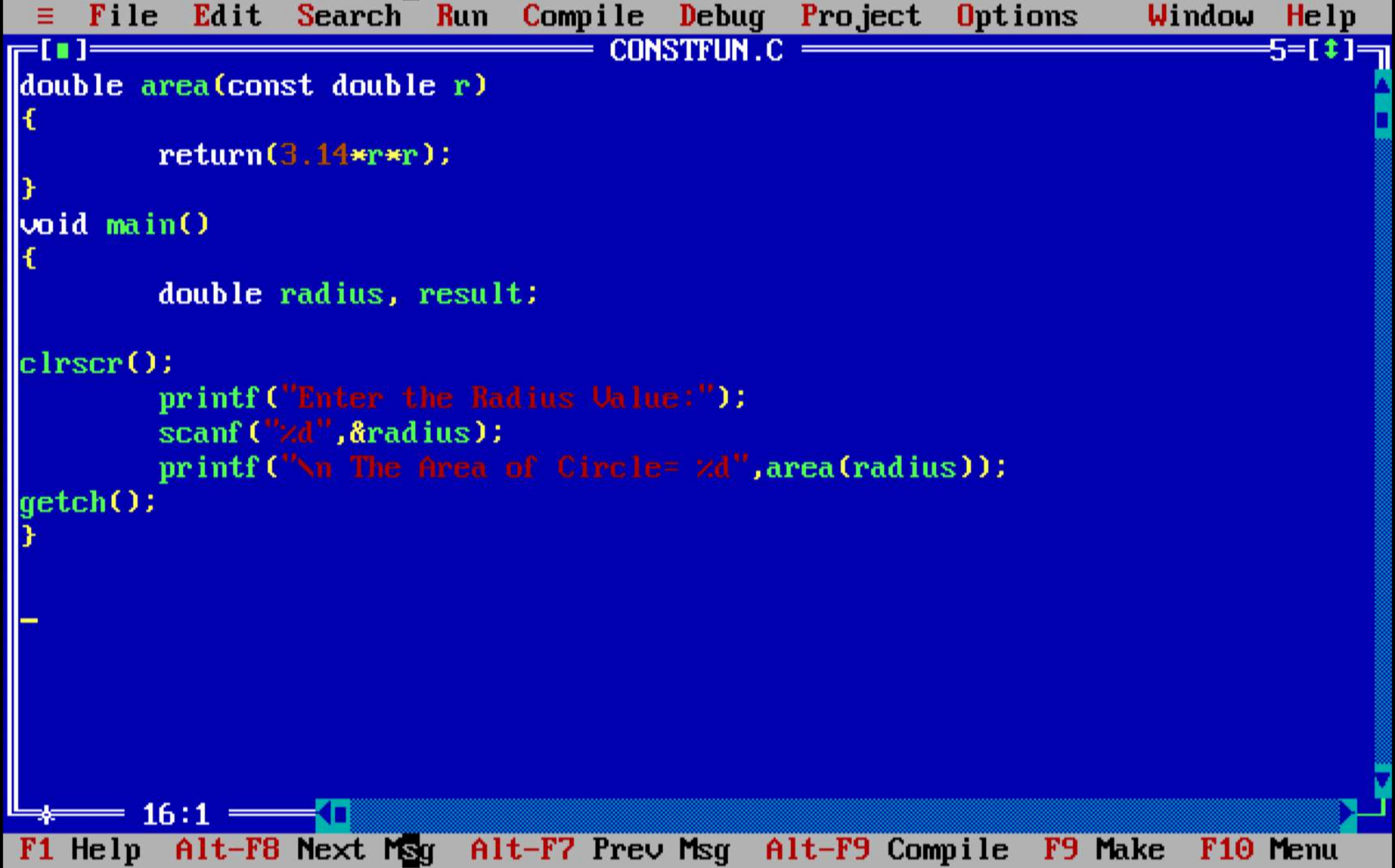
- The constant variable can be declared using const keyword.
- The const keyword makes variable value stable.
- The constant variable should be initialized while declaring.
- The const modifier enables to assign an initial value to a variable that cannot be changed later inside the body of the function.

Syntax :

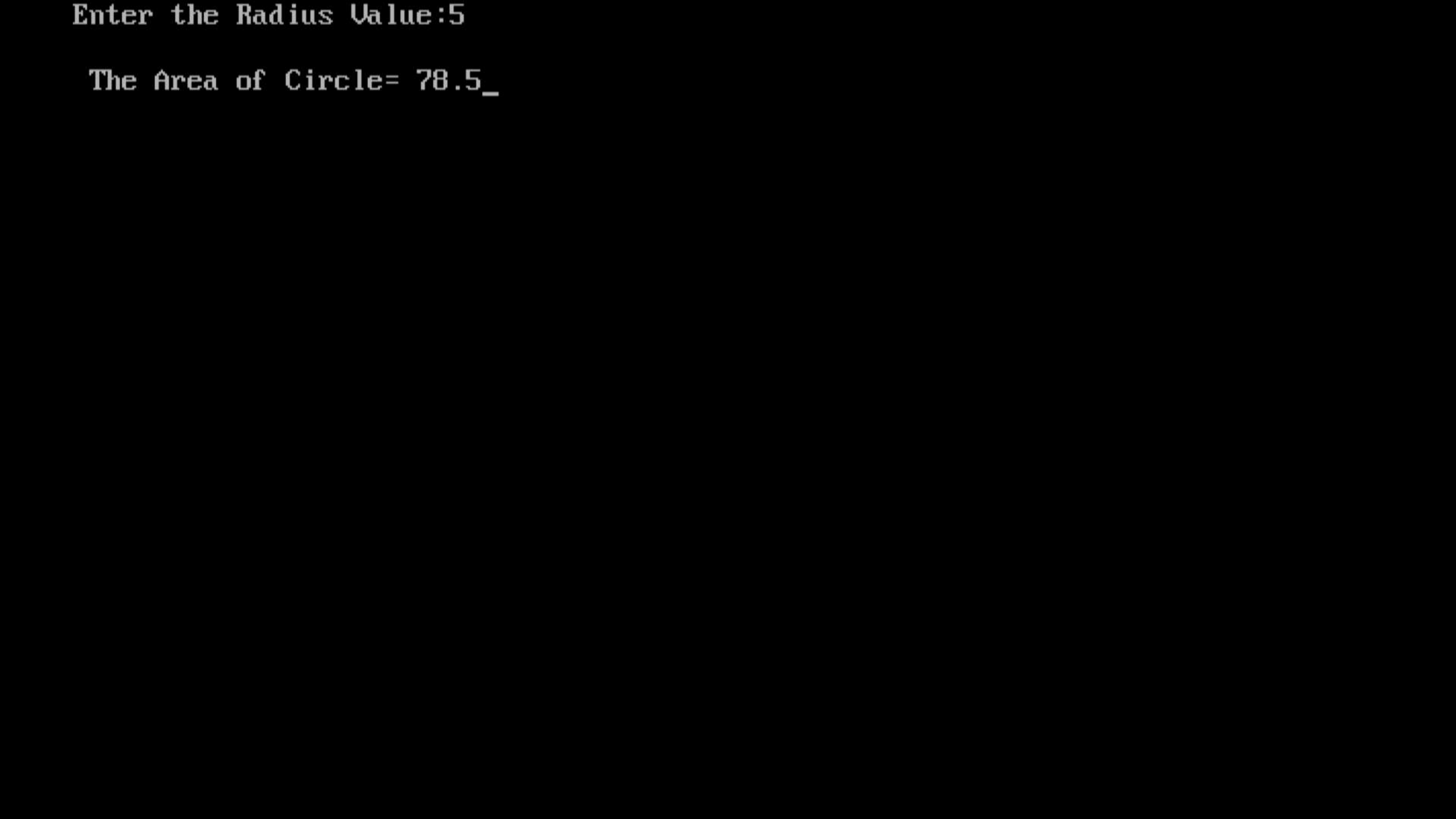
```
<returntype><functionname> (const <datatype variable=value>)
```

Example:

```
int minimum(const int a=10);
float area(const float pi=3.14, int r=5);
```


≡ File Edit Search Run Compile Debug Project Options Window Help
CONSTFUN.C
16:1
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
double area(const double r)
{
        return(3.14*r*r);
}
void main()
{
        double radius, result;

clrscr();
        printf("Enter the Radius Value:");
        scanf("%d",&radius);
        printf("\n The Area of Circle= %d",area(radius));
getch();
}
```


```
    Enter the Radius Value:5

     The Area of Circle= 78.5_
```

Thank you