1402/11/22

Microcontrollers

Lecture 4:

AVR Programming in C

By: M.Razeghizadeh

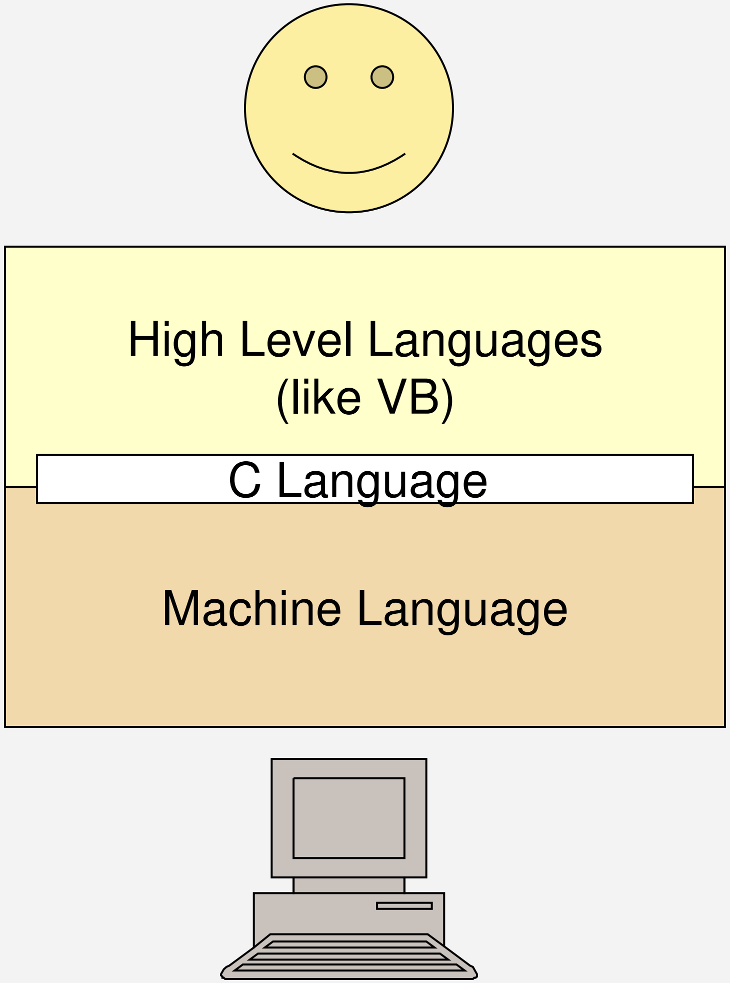
Start now!

Contents of This Slide

- Data Types
- Time Delays
- IO Programming in C
- Logic Operation in C
- Data serialization in C

Languages

- High Level Languages
  - Easy to develop and update
- C Language
  - Acceptable performance
  - Easy to develop and update
  - Portable
- Low Level Languages
  - High performance
  - Not portable

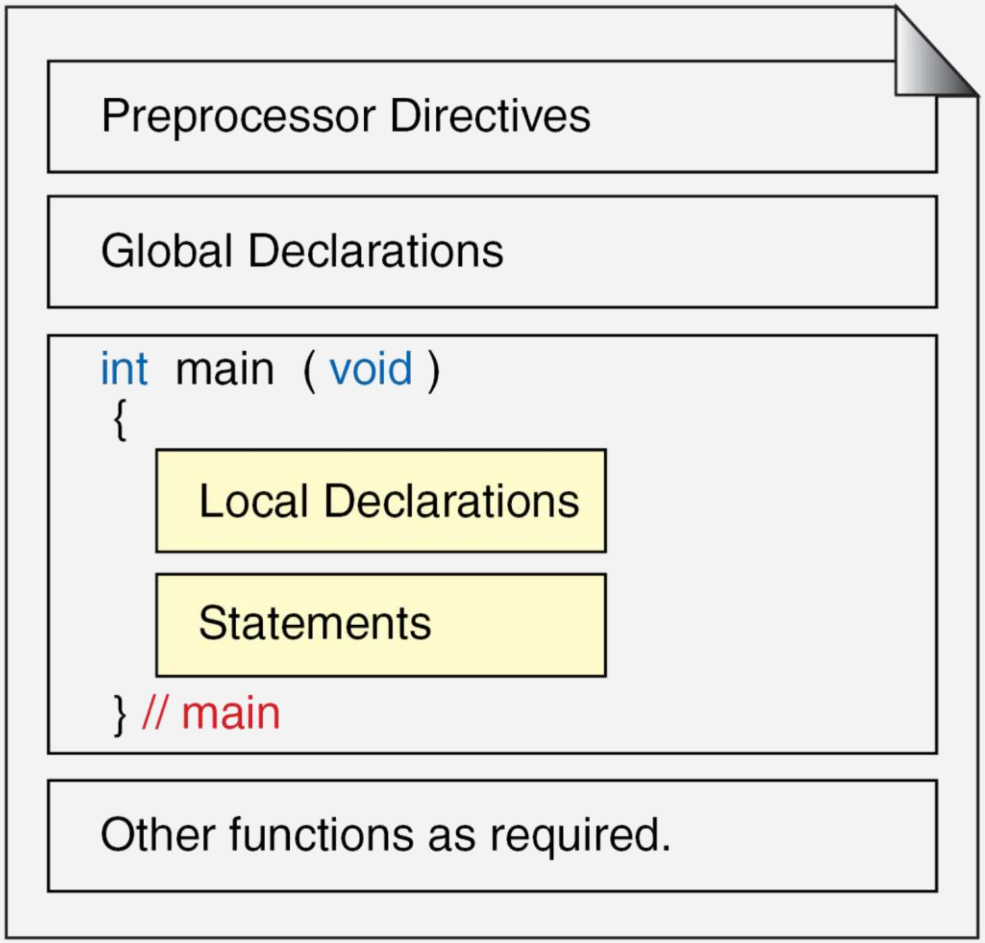


High Level Languages (like VB)

C Language

Machine Language

Structure of a C Program

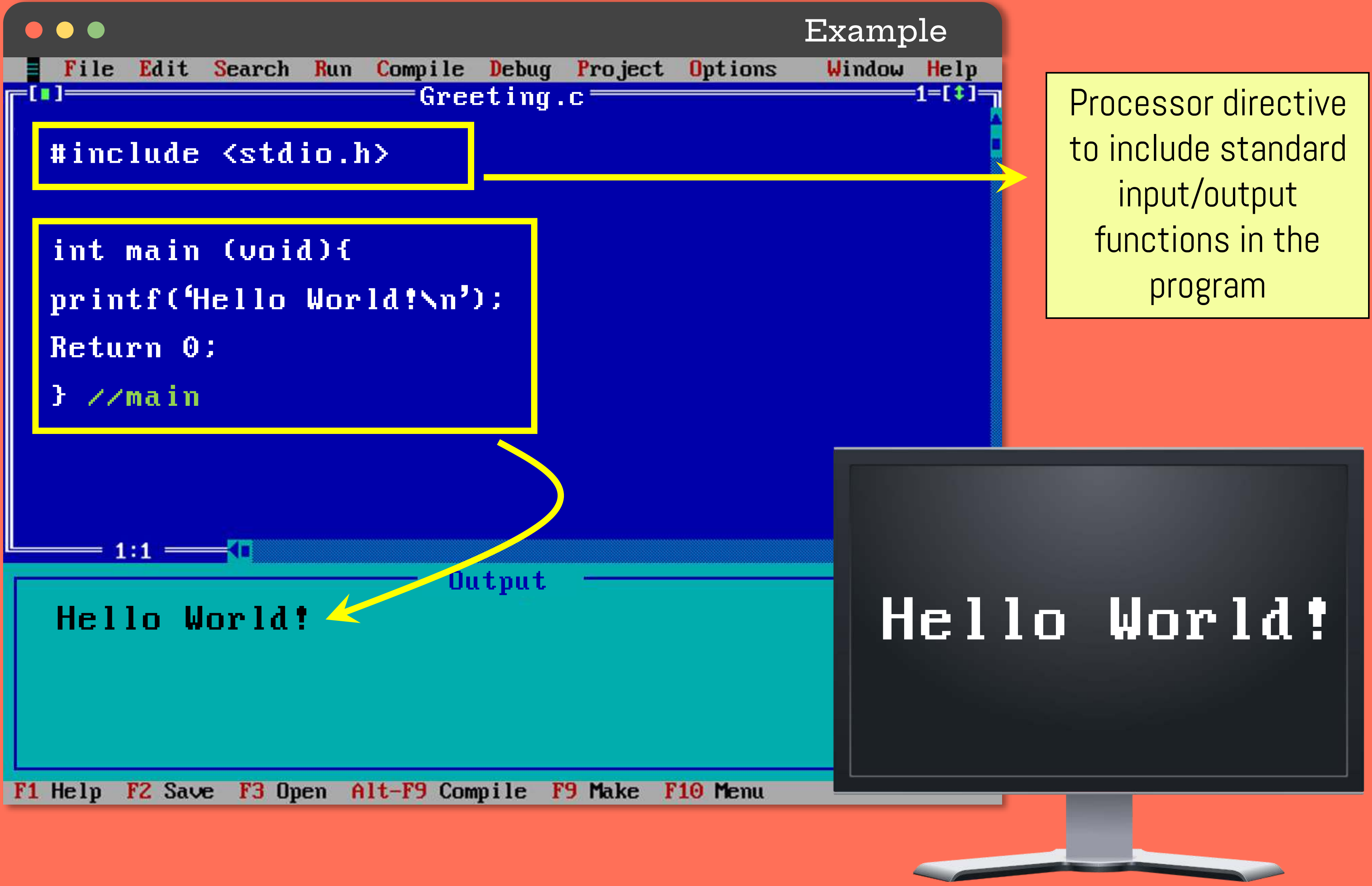


Preprocessor Directives

Global Declarations

```
int  main  ( void )
{
    Local Declarations
    Statements
} // main
```

Other functions as required.



Example
File Edit Search Run Compile Debug Project Options Window Help
Greeting.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>

int main (void){
printf(‘Hello World!\n’);
Return 0;
} //main
```

```
Hello World!
```

Processor directive to include standard input/output functions in the program

```
Hello World!
```

Keywords in C

- C language include 32 words.
- The lack of keywords in a language does not necessarily indicate its weakness.
- Basic language include 150 keywords.

Keywords

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

AVR Keywords in C

Keywords

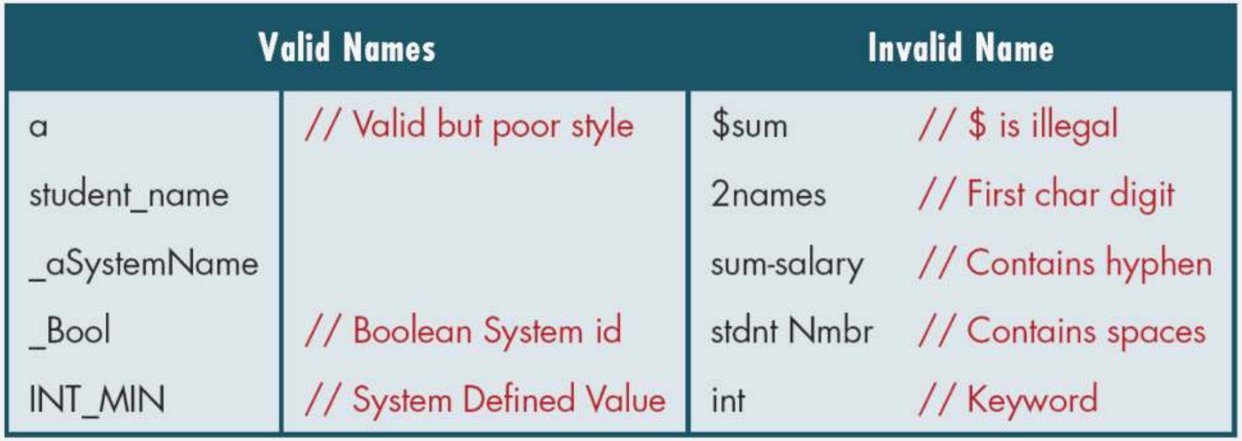
~~auto~~
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

+

interrupt
defined
Inline
bit
eeprom
flash
sfrw
sfrb

Features of C

- C language is case-sensitive. In this language, there is a difference between lowercase and uppercase letters, and all keywords must be written in lowercase.

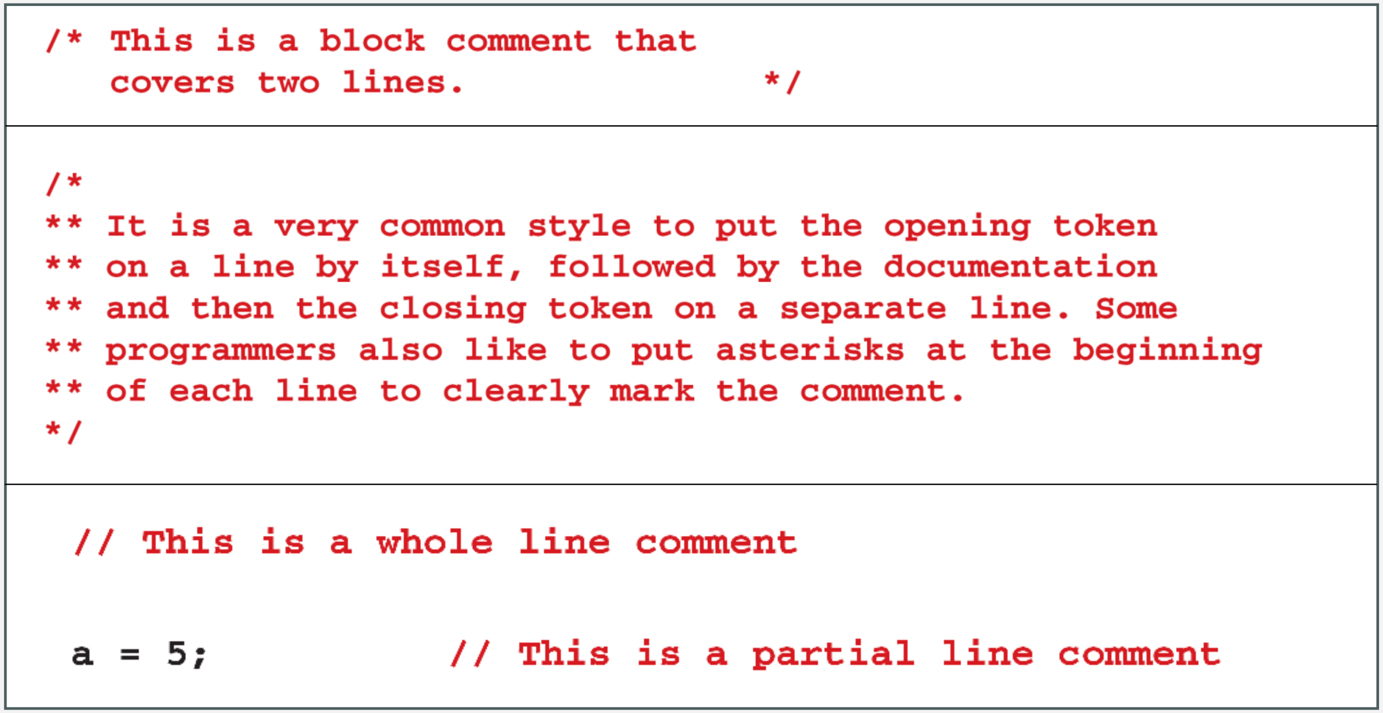


Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

- In the C programming language, each statement must be terminated with a semicolon (;).
- The maximum length of a command is 255 characters.
- Each instruction can be written on one or multiple lines.

Features of C

- Single-line comments start with // at the beginning of the line, and multi-line comments are enclosed between /* and */.



```
/* This is a block comment that
   covers two lines.              */
```

```
/*
** It is a very common style to put the opening token
** on a line by itself, followed by the documentation
** and then the closing token on a separate line. Some
** programmers also like to put asterisks at the beginning
** of each line to clearly mark the comment.
*/
```

```
 // This is a whole line comment

 a = 5;           // This is a partial line comment
```

Identifiers and Constants

- Identifier feature present in all computer languages.
- Identifiers allow us to name data, string, or any value (other objects) in the program. They can also be used to define function-like macros.
- Each identified object in the computer is stored at a unique address.
- This feature is similar to the EQU directive in assembly language.

```
#define name "Micro"
#define number 455.46
#define out5 PORTA.5

#define sum(x,y) x+y
i = sum(5,8);
```

Note:

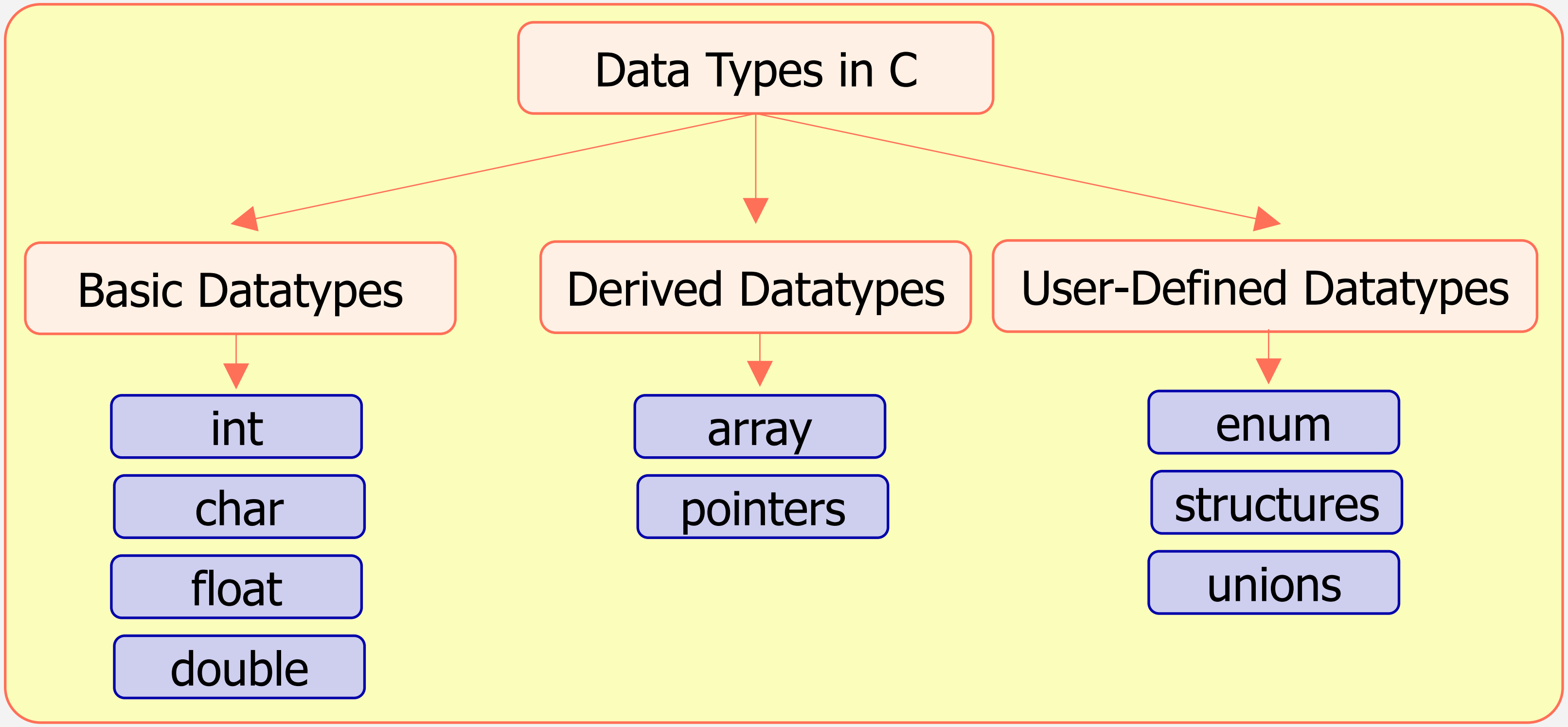
1. First character must be alphabetic character or underscore.
2. Must consist only of alphabetic characters, digits, or underscores.
3. First 63 characters of an identifier are significant.
4. Can not duplicate a keyword.

Data Types

- data types are used to notify the type of value assigned to a variable.
- four basic data types in c are: int, float, char and double.

general syntax is:

```
Datatype    var1, var2, ... , varn;
```



Basic Data Types in C

int

- Hold integer values.
- Occupy 2 bytes of space in memory.
- Range from -32768 to +32767.

Example:

```
int a, b, sum;
```

Initialization:

```
int a=2, b=23;
```

char

- Hold a single ascii character that contains numbers from 1 to 9, small and capital letters e to z and some special characters enclosed within two single quotes
- Occupy 1 bytes of space in memory.
- Range from -128 to +127.

Example:

```
char ch, arr;
```

Initialization:

```
char ch='c', arr='2';
```

Basic Data Types in C

float

- Hold the value with decimal points that is up to 6 decimal places
- Occupy 4 bytes of space in memory.
- Range from 3.4e-38 to 3.4e+38.

Example:

```
float x, age;
```

Initialization:

```
float pi = 3.14;
float age = 2.5;
```

double

- Hold values with decimal points
- Occupy 8 bytes of space in memory.
- Range from 1.7e-308 to 1.7e+308.

Example:

```
double a;
```

Initialization:

```
double a = 2.3214526;
```

Basic Data Types in C

Modifiers [Qualifiers] in C

signed

unsigned

short

long

general syntax of handling modifier:

```
<modifier> <basic data type> <variable-list>;
```

example:

```
unsigned int a = 345;
```

- signed modifiers are used to hold both negative and positive numbers at the same time
- unsigned is used to store only negative numbers
- short requires memory space half the size of the basic data type
- Long modifier is used to hold large numbers and it doubles the size of the data type.
- we cannot apply qualifiers to the float data type but we can increase the size of the double data type by using a long qualifier
- we cannot apply qualifiers long and short on the char data type
- there are also a wide variety of data types available in c programming language that comes in handy when you level up to the next stage with respect to your data

Basic Data Types in C

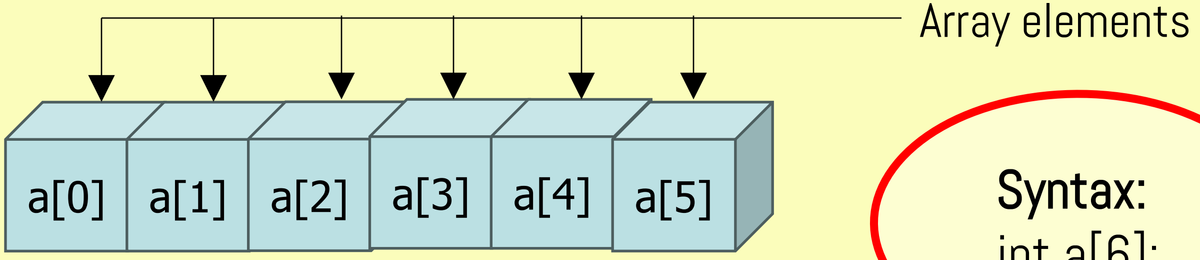
Size and Format Specifier of Datatypes

Datatype	Size in bytes	Format Specifier
char	1	%c
signed char	1	%c
unsigned char	1	%c
short int or int	2	%hd, %d
unsigned int	2	%u
long int	4	%ld
unsigned long int	4	%lu
float	4	%f
double	8	%lf
long double	10	%Lf

Derived Data Types in C

array

- array is a homogenous collection of elements of same datatypes [int, float, char and double].



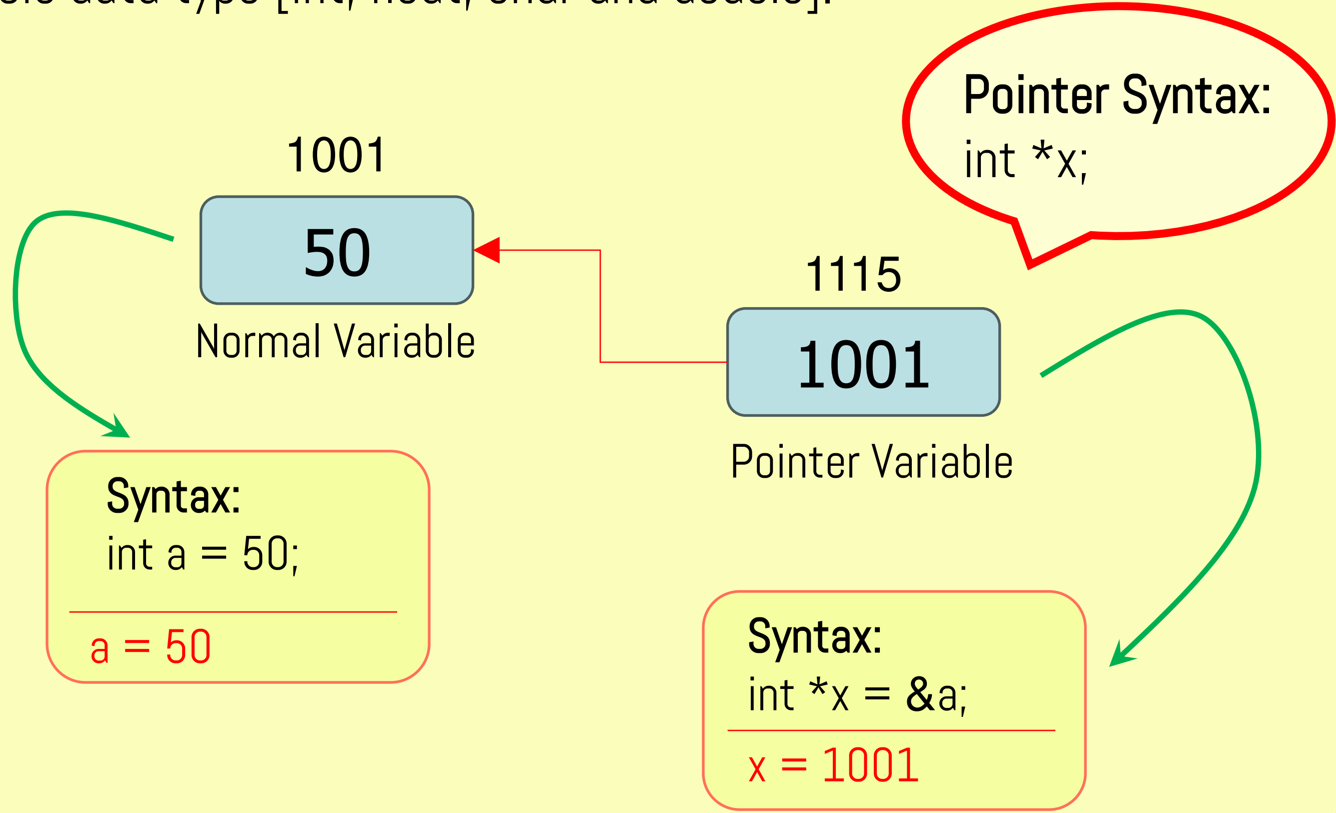
1-D Array with 6 elements

Syntax:
int a[6];

Derived Data Types in C

pointer

- Pointer holds the address of another variable declined under any basic data type [int, float, char and double].



Pointer Syntax:
int *x;

1001

50

Normal Variable

1115

1001

Pointer Variable

Syntax:
int a = 50;

a = 50

Syntax:
int *x = &a;

x = 1001

User-Defined Data Types in C

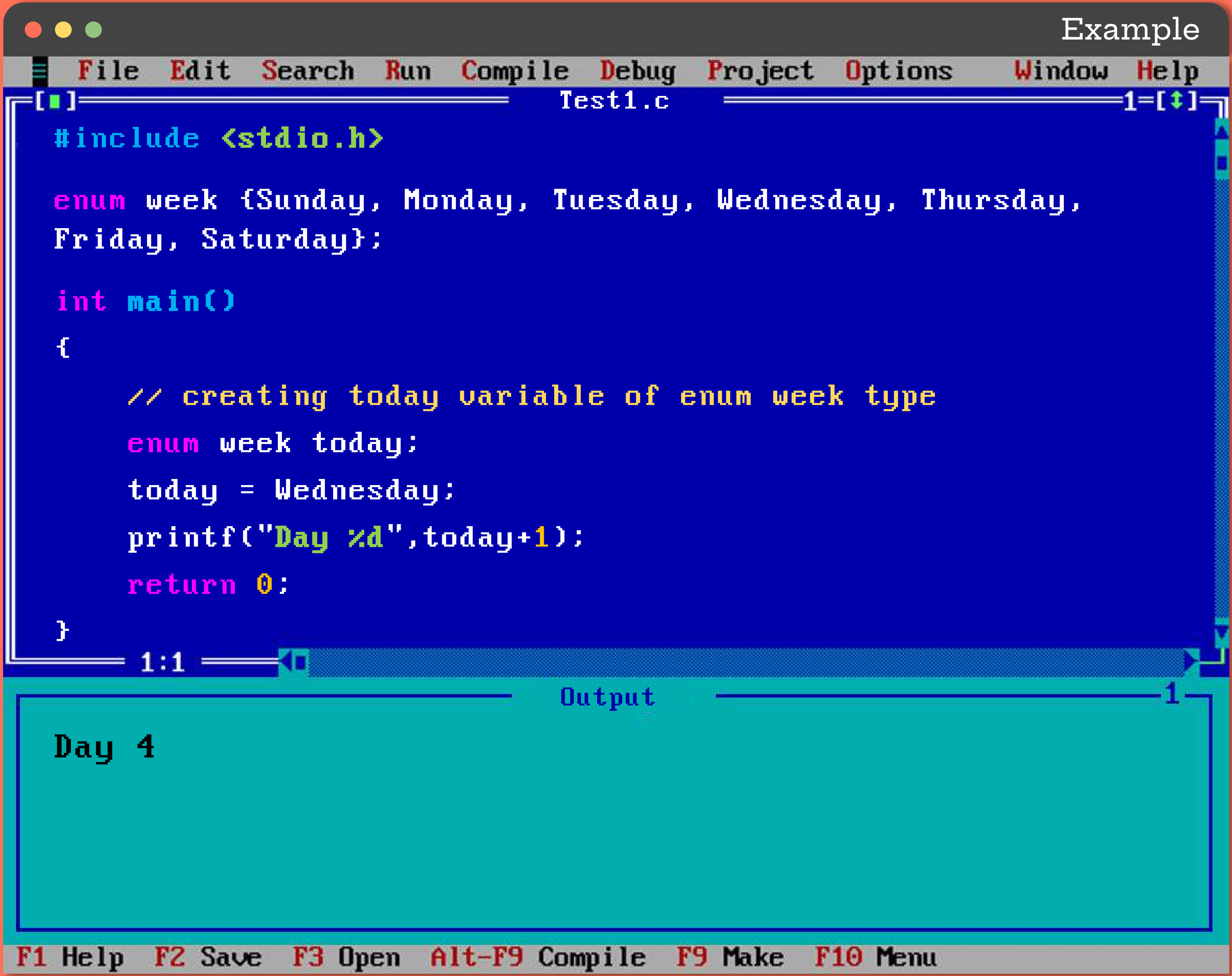
enum

- an enumeration type (enum) is a data type that consists of same integral constants. To define enums, the enum keyword is used.

```
enum  flag {const1, const2, ..., constN};
```

- By default, const1 is 0, const2 is 1 and so on.
- The default values of enum elements can be changed during declaration (if necessary).

```
// Changing default values of enum constants
enum suit {
   club = 0,
   diamonds = 10,
   hearts = 20,
   spades = 3,
};
```



Example
File Edit Search Run Compile Debug Project Options Window Help
Test1.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

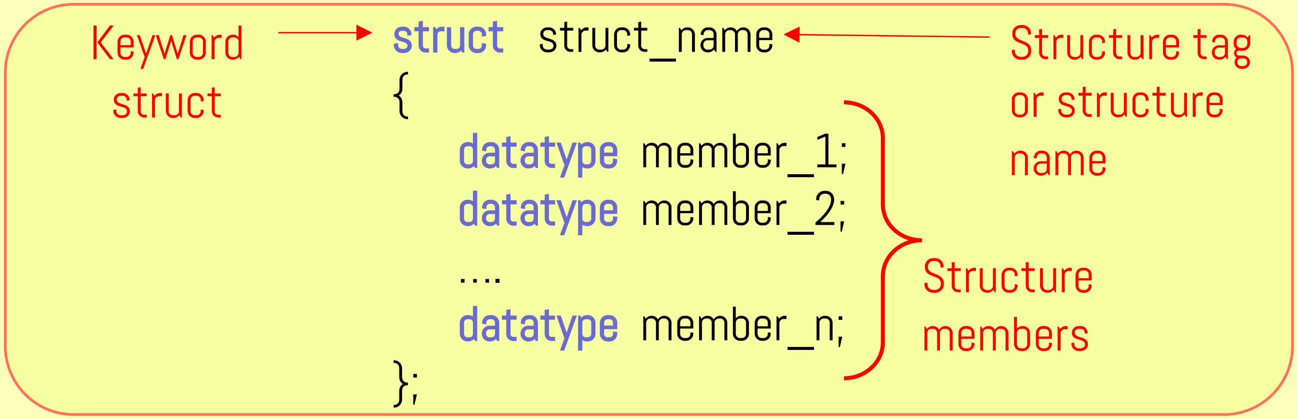
```c
#include <stdio.h>

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday};

int main()
{
    // creating today variable of enum week type
    enum week today;
    today = Wednesday;
    printf("Day %d",today+1);
    return 0;
}
```

```
Day 4
```

User-Defined Data Types in C

struct (structure)

- is a collection of elements having same or different data types



```
struct   struct_name
{
      datatype  member_1;
      datatype  member_2;
      ....
      datatype  member_n;
};
```

Keyword struct

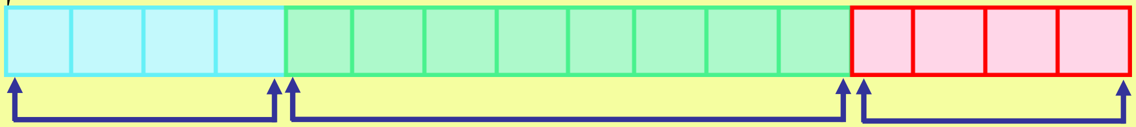
Structure tag or structure name

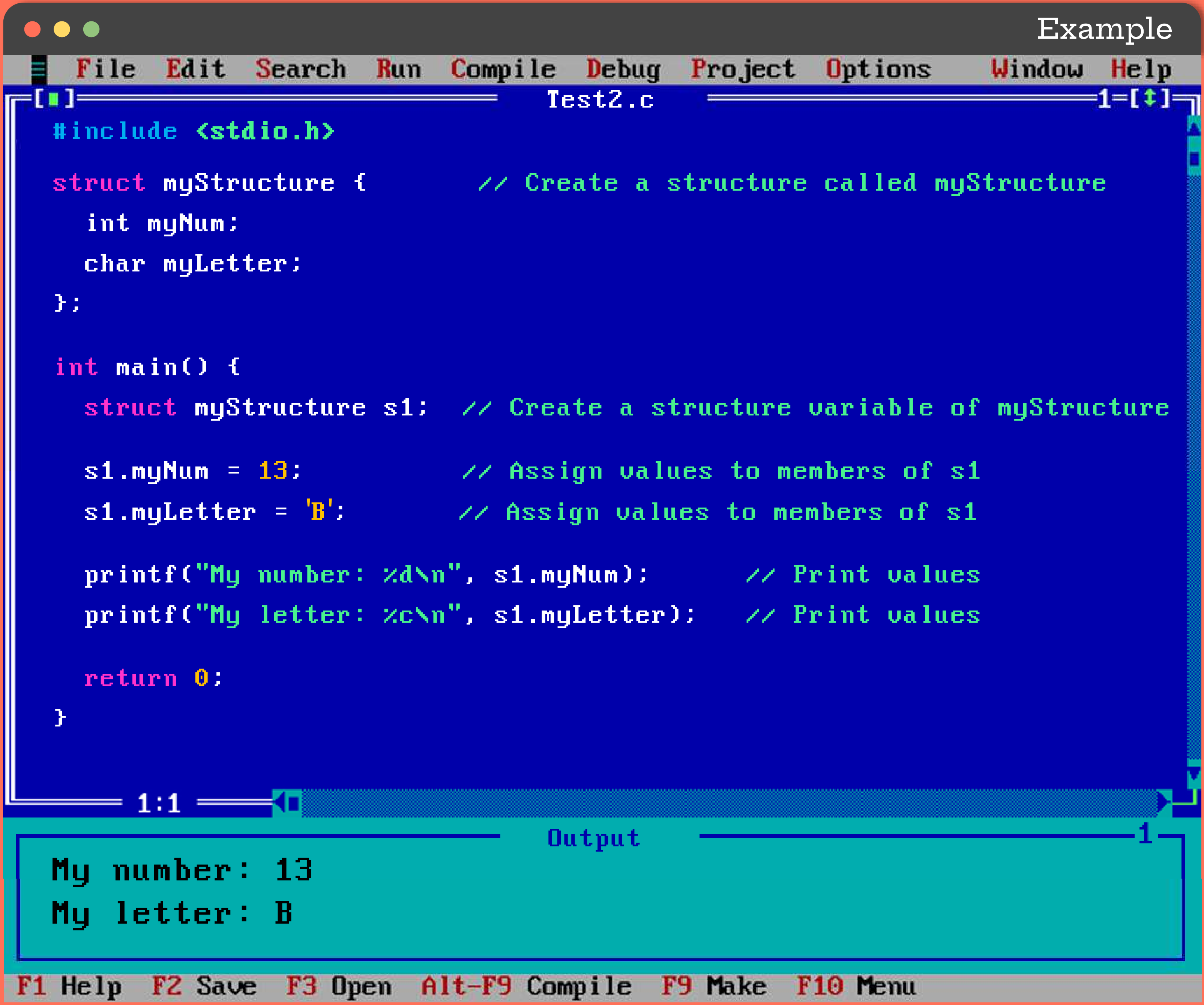
Structure members

example

```
struct  add
{
     int  var1;
     char var2[8];
     float  var3;
};
```

Memory représentation for struct_var





Example
File Edit Search Run Compile Debug Project Options Window Help
Test2.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <stdio.h>

struct myStructure {        // Create a structure called myStructure
  int myNum;
  char myLetter;
};

int main() {
  struct myStructure s1;  // Create a structure variable of myStructure

  s1.myNum = 13;          // Assign values to members of s1
  s1.myLetter = 'B';      // Assign values to members of s1

  printf("My number: %d\n", s1.myNum);     // Print values
  printf("My letter: %c\n", s1.myLetter);  // Print values

  return 0;
}
```

```
My number: 13
My letter: B
```

User-Defined Data Types in C

Union

- is a collection of elements having same or different data types but share memory space.

Keyword union →

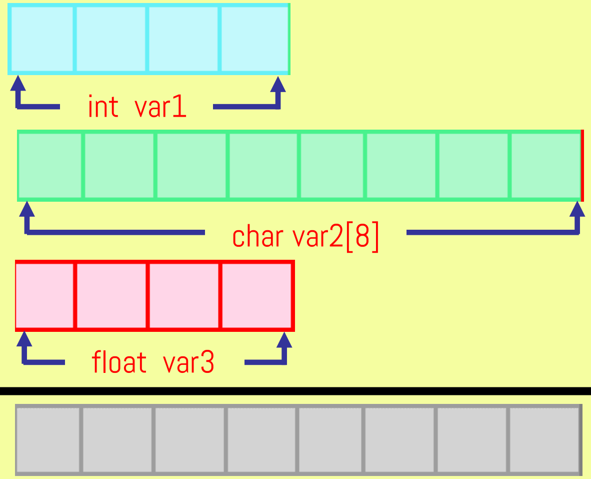
```
union union_name
{
    datatype member_1;
    datatype member_2;
    ....
    datatype member_n;
};
```

← union tag or union name

} Union members

example

```
union add
{
    int var1;
    char var2[8];
    float var3;
};
```

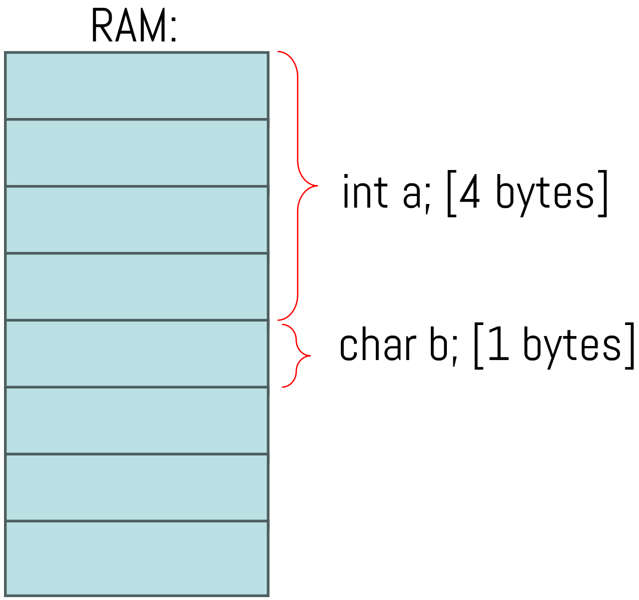


Memory representation for add:

User-Defined Data Types in C

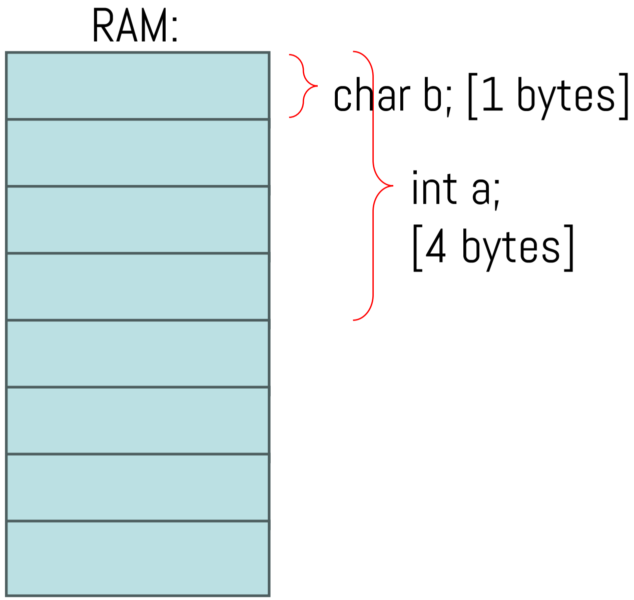
Union Vs Structure

```
struct add
{
      int a;
      char b;
};
```



add size: 5 byte

```
union add
{
      int a;
      char b;
};
```



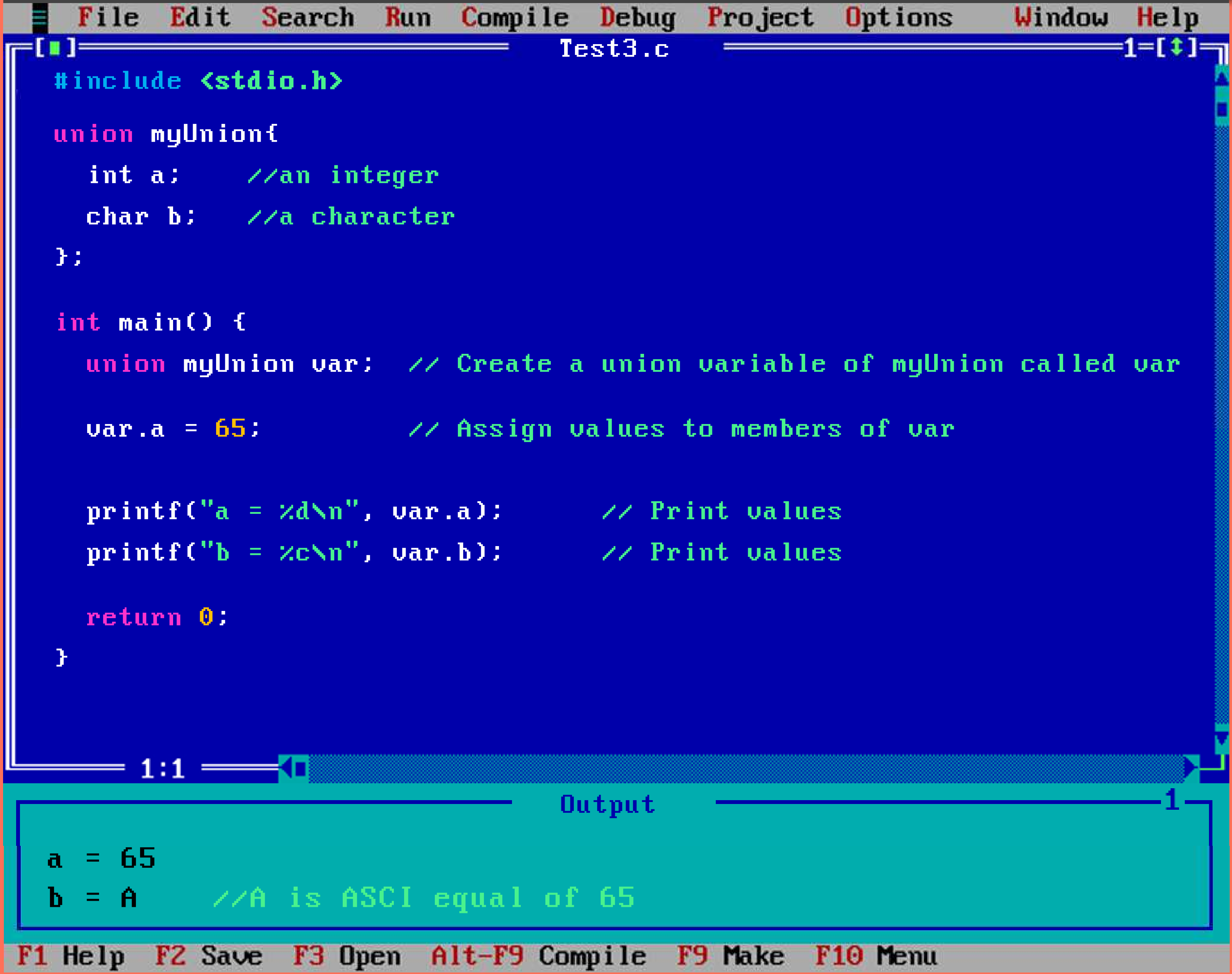
add size: 4 byte

User-Defined Data Types in C

Union Vs Structure

- In structure every member is assigned a unique memory location whereas in Union all the data members share a memory location.
- Change in the value of one data member does not affect other data members in the structure whereas Change in the value of one data member affects the value of other data members.
- In structure we can initialize multiple members at a time whereas in union we can initialize only one member at once.
- Size of a structure is the sum of the size of every data member whereas A union's total size is the size of the largest data member.
- Users can access or retrieve any member at a time in structure whereas can access or retrieve only one member at a time in unions.

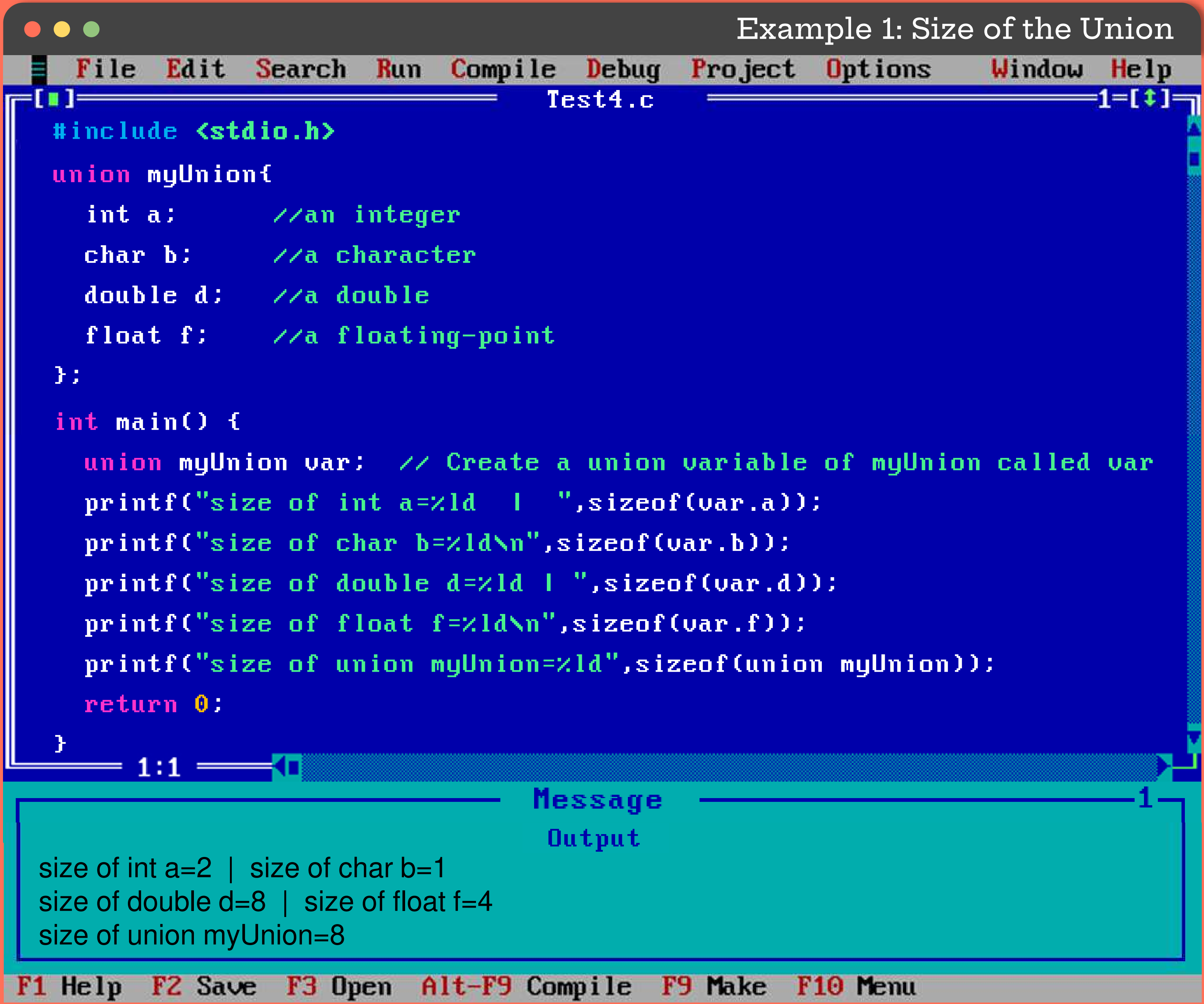
Example 1: Program to Declare and Define Union



```
#include <stdio.h>

union myUnion{
   int a;     //an integer
   char b;    //a character
};

int main() {
   union myUnion var;  // Create a union variable of myUnion called var

   var.a = 65;         // Assign values to members of var


   printf("a = %d\n", var.a);      // Print values
   printf("b = %c\n", var.b);      // Print values

   return 0;
}
```

```
a = 65
b = A    //A is ASCI equal of 65
```

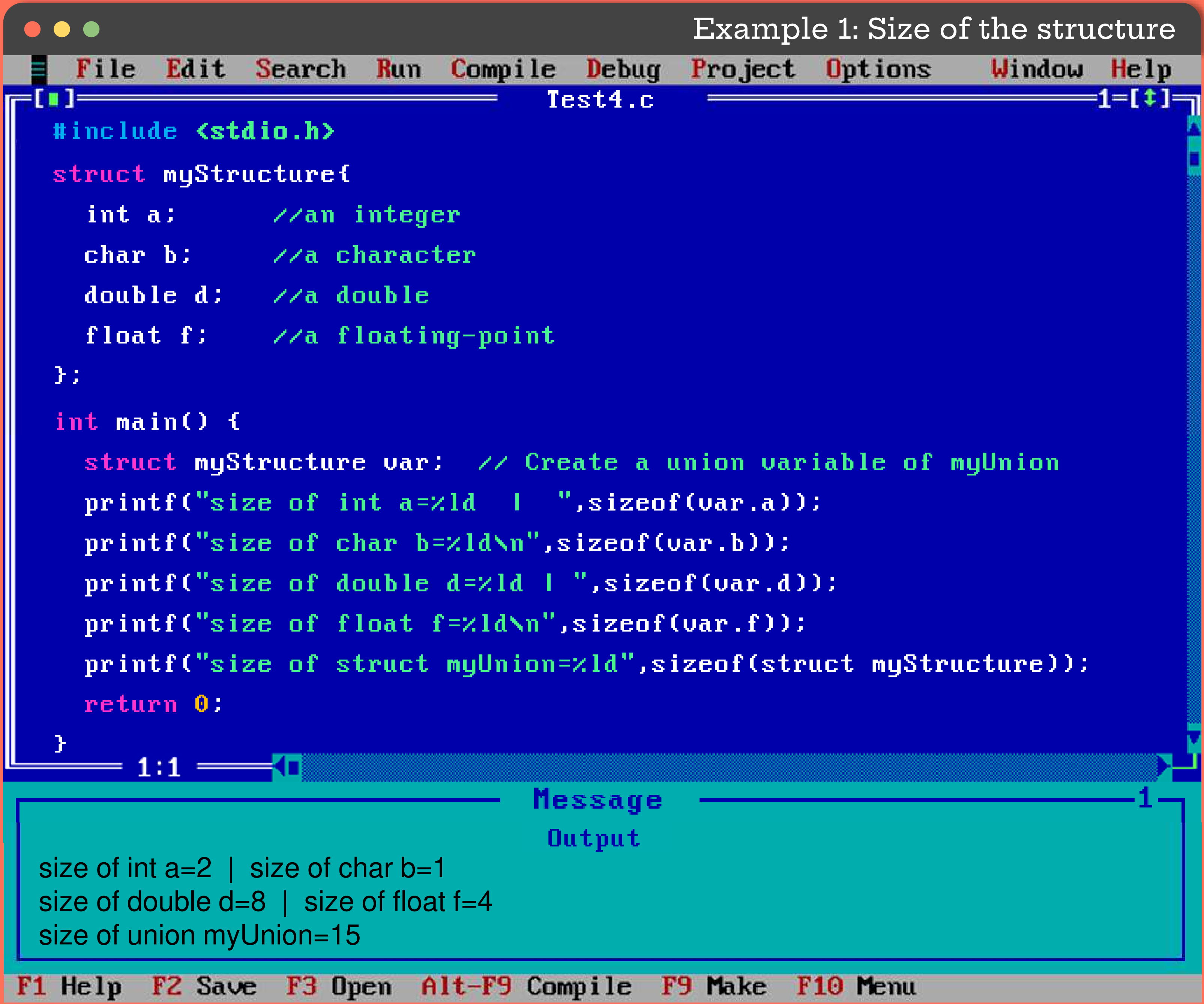
Example 1: Size of the Union



```c
#include <stdio.h>
union myUnion{
   int a;       //an integer
   char b;      //a character
   double d;    //a double
   float f;     //a floating-point
};
int main() {
   union myUnion var;  // Create a union variable of myUnion called var
   printf("size of int a=%ld  |  ",sizeof(var.a));
   printf("size of char b=%ld\n",sizeof(var.b));
   printf("size of double d=%ld | ",sizeof(var.d));
   printf("size of float f=%ld\n",sizeof(var.f));
   printf("size of union myUnion=%ld",sizeof(union myUnion));
   return 0;
}
```

Output

```
size of int a=2  |  size of char b=1
size of double d=8  |  size of float f=4
size of union myUnion=8
```



Example 1: Size of the structure

File Edit Search Run Compile Debug Project Options Window Help
Test4.c
1
1:1
Message
1

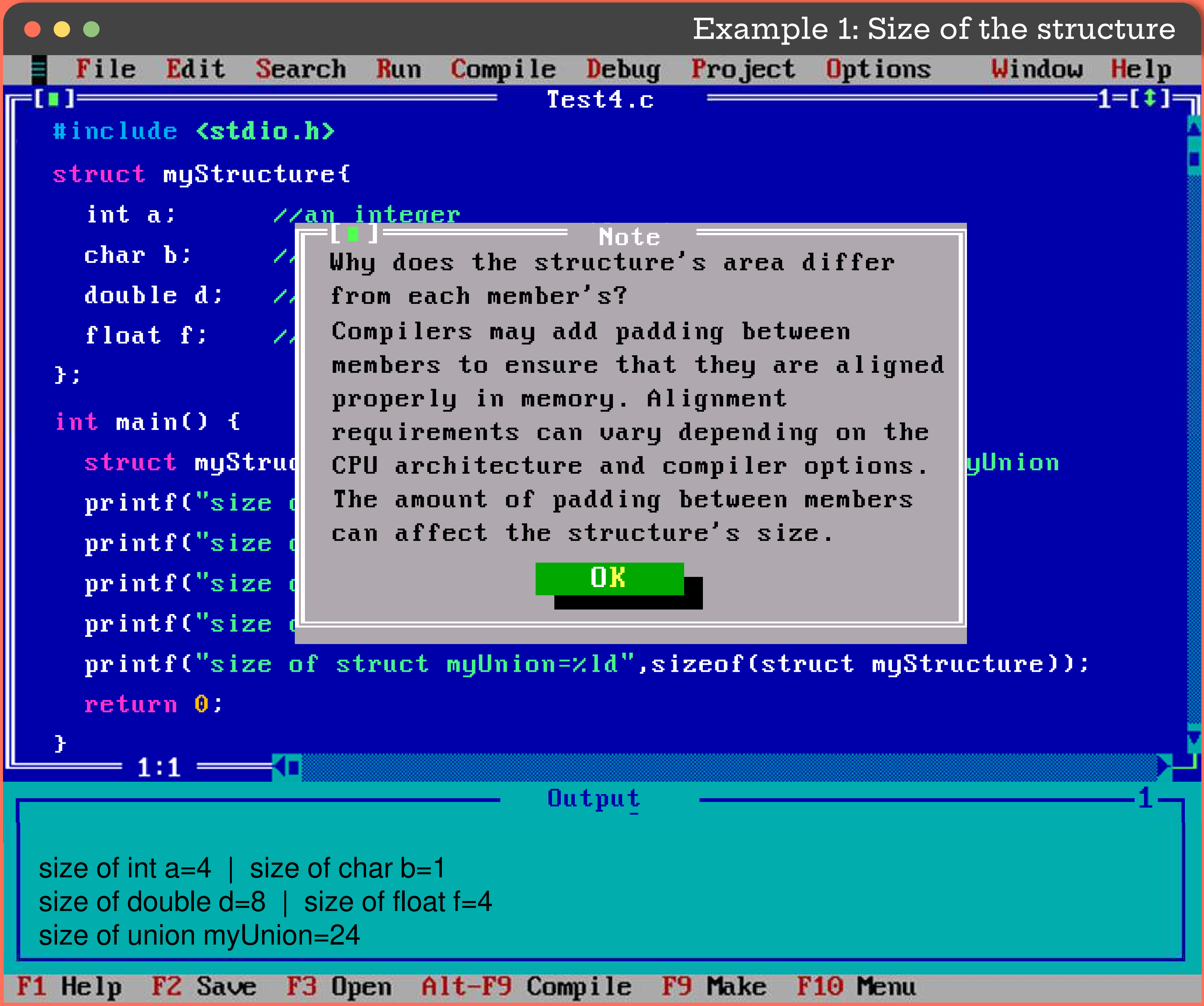
```c
#include <stdio.h>
struct myStructure{
   int a;       //an integer
   char b;      //a character
   double d;    //a double
   float f;     //a floating-point
};
int main() {
   struct myStructure var;  // Create a union variable of myUnion
   printf("size of int a=%ld  |  ",sizeof(var.a));
   printf("size of char b=%ld\n",sizeof(var.b));
   printf("size of double d=%ld | ",sizeof(var.d));
   printf("size of float f=%ld\n",sizeof(var.f));
   printf("size of struct myUnion=%ld",sizeof(struct myStructure));
   return 0;
}
```

Output

```
size of int a=2  |  size of char b=1
size of double d=8  |  size of float f=4
size of union myUnion=15
```

F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Example 1: Size of the structure



```
#include <stdio.h>
struct myStructure{
   int a;        //an integer
   char b;       //
   double d;     //
   float f;      //
};
int main() {
   struct myStruc                                    yUnion
   printf("size o
   printf("size o
   printf("size o
   printf("size o
   printf("size of struct myUnion=%ld",sizeof(struct myStructure));
   return 0;
}
```

Note

Why does the structure's area differ from each member's?

Compilers may add padding between members to ensure that they are aligned properly in memory. Alignment requirements can vary depending on the CPU architecture and compiler options. The amount of padding between members can affect the structure's size.

OK

```
size of int a=4  |  size of char b=1
size of double d=8  |  size of float f=4
size of union myUnion=24
```

Variables in C [AVR]

- A variable is a region of memory space that is identified by a unique name.
- A variable can hold a numerical value, depending on its type.
- A variable can participate in calculations or store the result of calculations.

Types of Variables

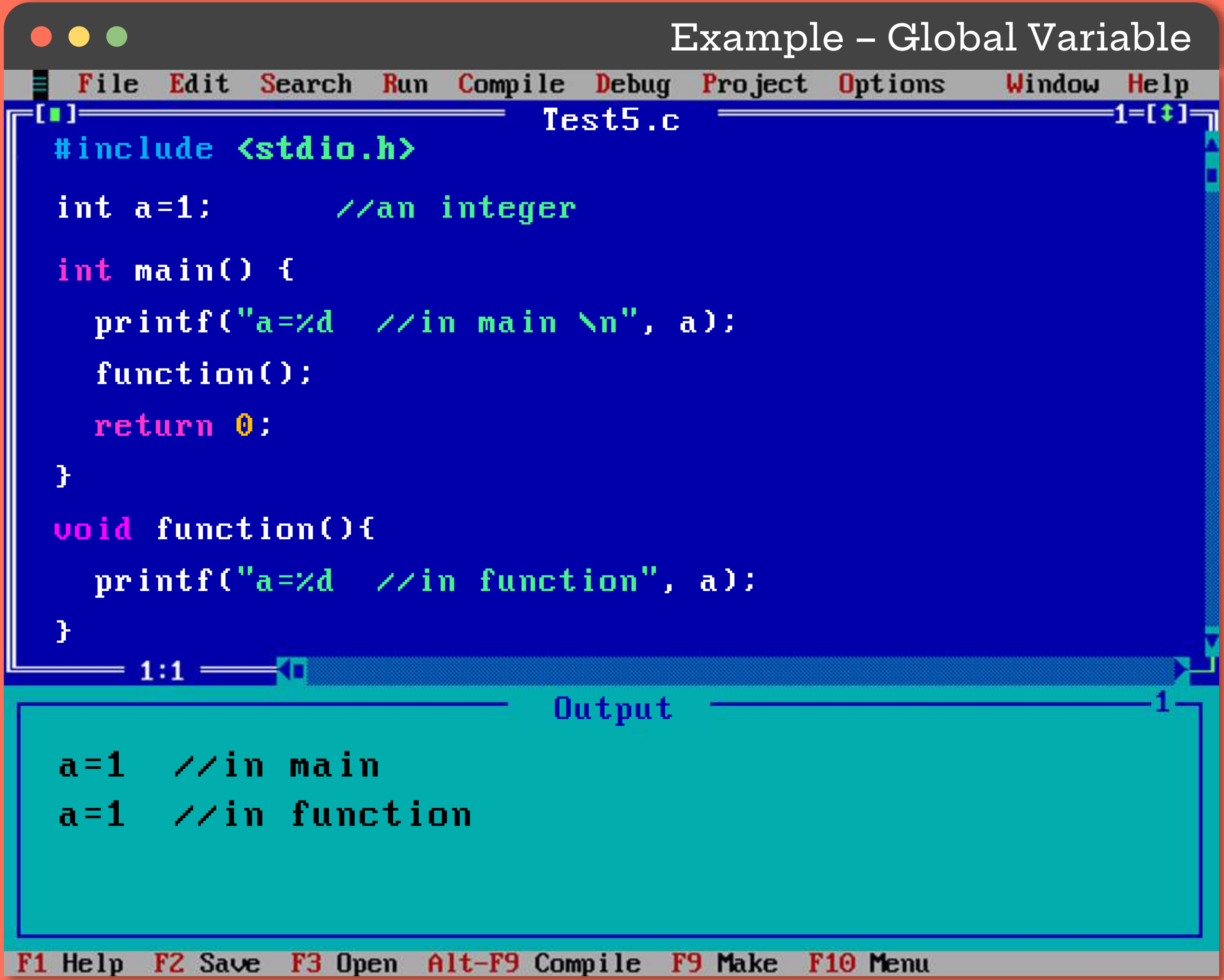
- Global Variables

  These variables are declared after the preprocessors and identifiers at the beginning of the program and outside of the function bodies. Their values are accessible in all functions and they retain their values throughout all functions.

- Local Variables

  Variables declared within the body of a function are local to that function and their values are lost (i.e., become zero) when exit the function body.

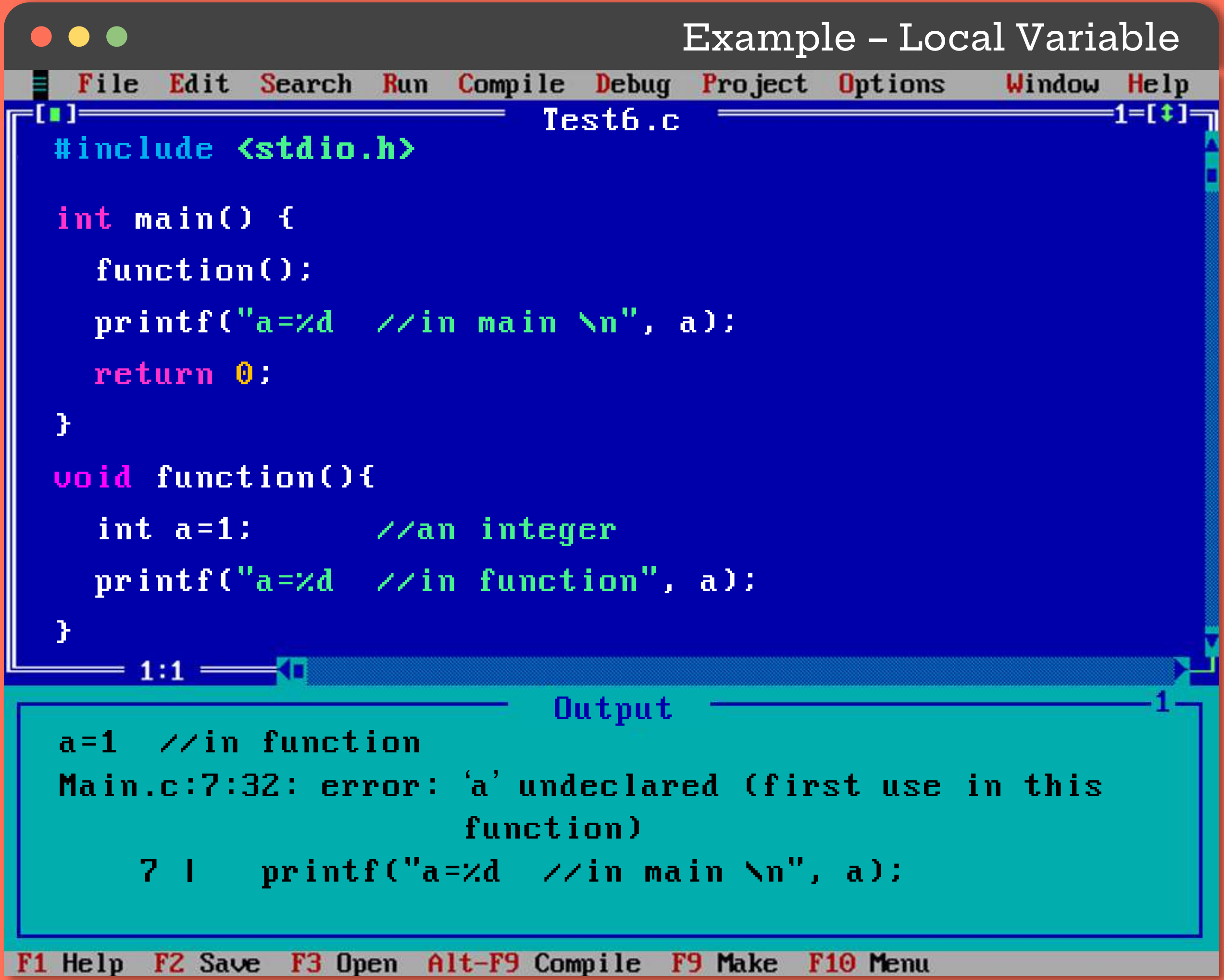
Example – Global Variable


File Edit Search Run Compile Debug Project Options Window Help
Test5.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>
int a=1;       //an integer
int main() {
  printf("a=%d  //in main \n", a);
  function();
  return 0;
}
void function(){
  printf("a=%d  //in function", a);
}
```

```
a=1  //in main
a=1  //in function
```

Example – Local Variable



```
#include <stdio.h>

int main() {
  function();
  printf("a=%d  //in main \n", a);
  return 0;
}
void function(){
  int a=1;      //an integer
  printf("a=%d  //in function", a);
}
```

```
a=1  //in function
Main.c:7:32: error: ‘a’ undeclared (first use in this
                    function)
    7 |   printf("a=%d  //in main \n", a);
```

Variables in C [AVR]

Name of Variables

- The first character of a variable name cannot be a number.
- A variable name cannot be longer than 31 characters.
- A variable name can only be a combination of the letters a to z and A to Z, numbers, and the _ character.

Assign variables to different memory spaces for AVRs

- Generally when a variable is declared, it is stored in the first available location in SRAM memory.
- To define a variable in EEPROM memory, the **eeprom** keyword is used, and to define a variable in FLASH memory, the **flash** keyword is used before the variable declaration.

```
Syntax:     <type of memory space> <variable>
Example:     eeprom   char  b;
             flash    float  c;
```

Operators in C [AVR]

Definition

- An operator is a symbol that operates on a value or a variable. For example + is an operator to perform addition.
- Operators cannot be called as Instructions because they only perform a specific operation.

Arithmetic Operators

Priority	Operator	Definition	Example
First	++	One unit increment	++x or x++
	--	One unit decrement	--x or x--
Second	-	unary minus	-x
Third	*	Multiplication	x * y
	/	Division (Quotient)	x / y
	%	Division (remainder)	x % y
Fourth	-	Subtraction	x – y
	+	Addition	X + y

Operators in C [AVR]

D

- Ar

example + is an operator to perform addition.

- Operators (

specific op

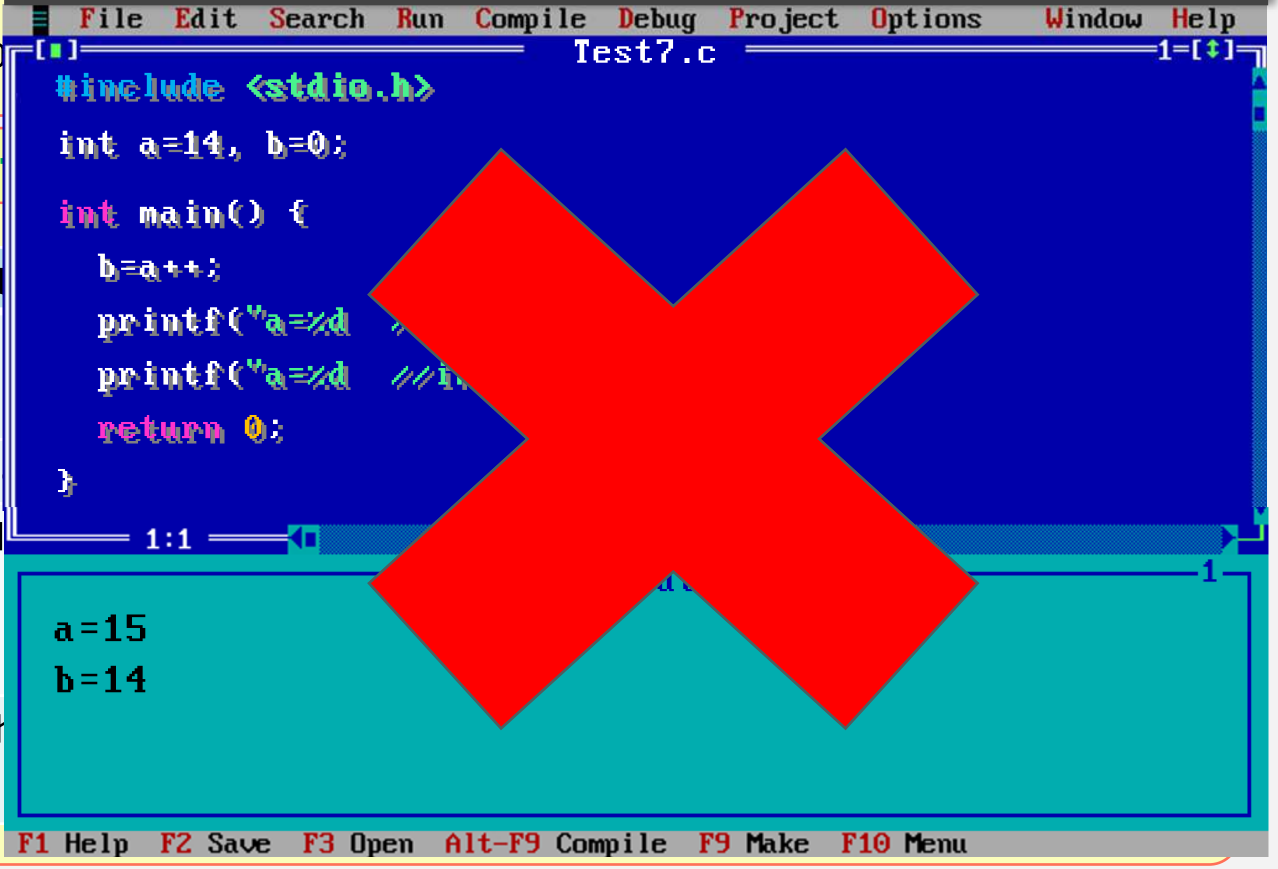
Arithme

Priorit
First
Secon
Third
Fourth

Example

Using ++ operator increase the value of a=14 by one unit, then store it in b variable?

Solution


File Edit Search Run Compile Debug Project Options Window Help
Test7.c
1:1
1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>
int a=14, b=0;
int main() {
  b=a++;
  printf("a=%d  /
  printf("a=%d  //i
  return 0;
}
```

```
a=15
b=14
```

Operators in C [AVR]

D

- Ar
  example + is an operator to perform addition.
- Operators
  specific op

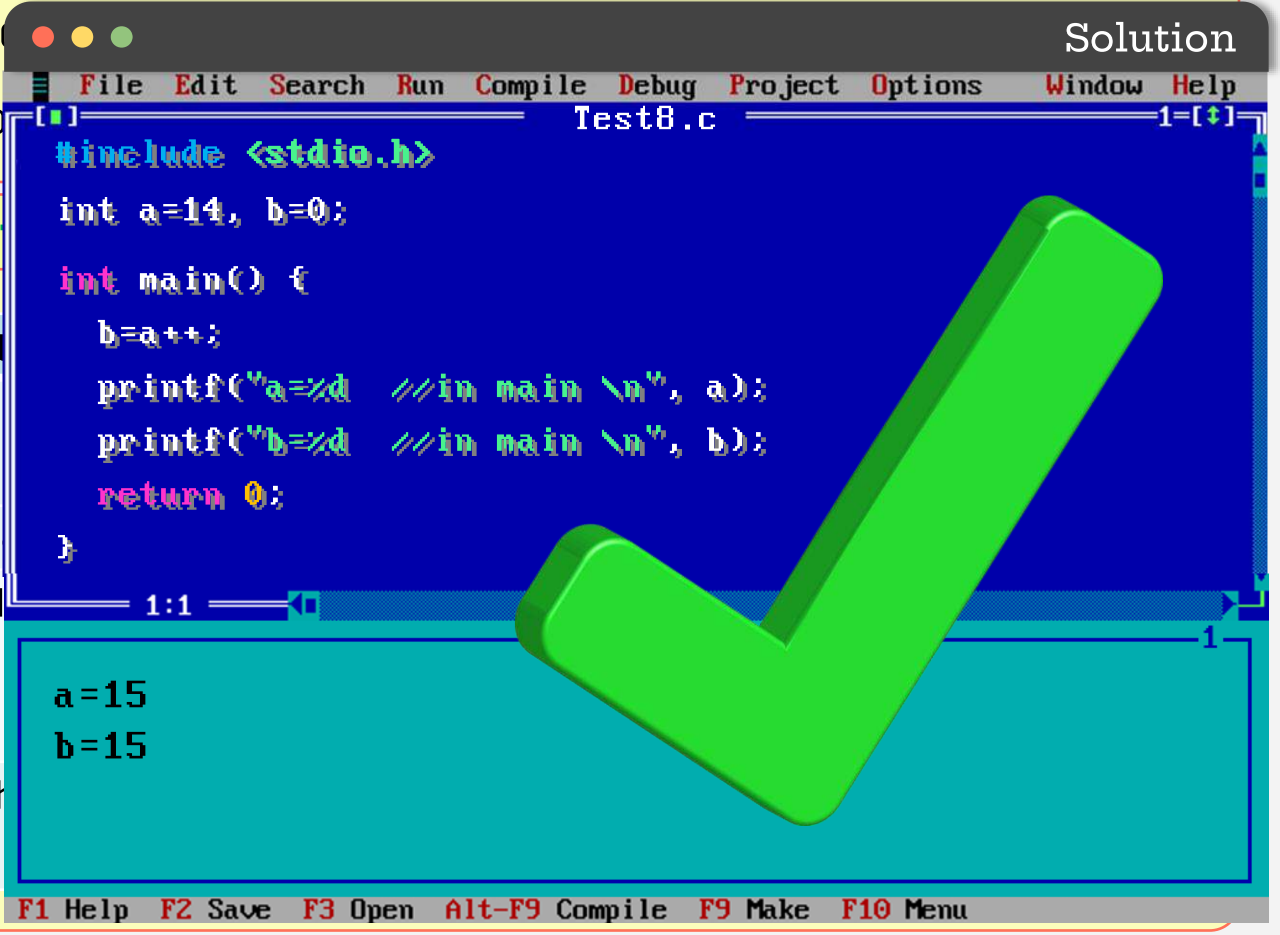
Arithme

Priorit
First
Secon
Third
Fourth

Example

Using ++ operator increase the value of a=14 by one unit, then store it in b variable?

Solution


File Edit Search Run Compile Debug Project Options Window Help
Test8.c
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

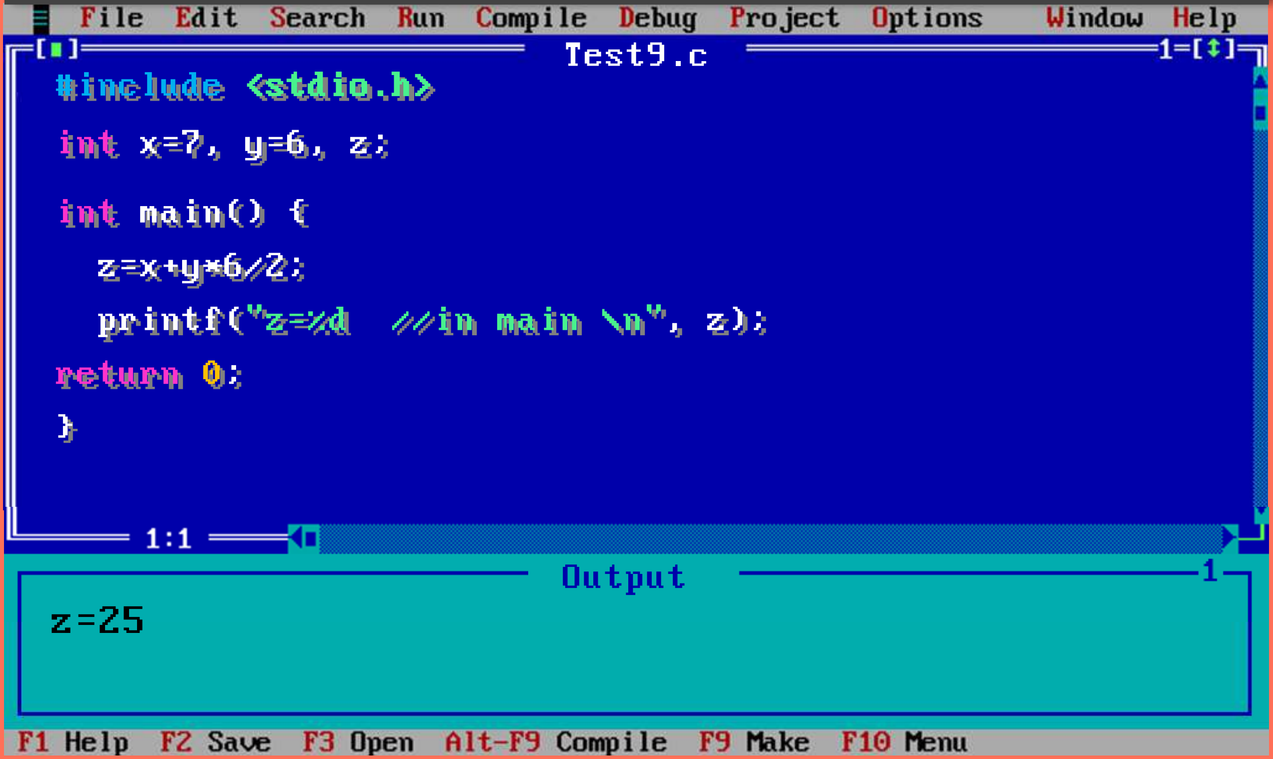
```
#include <stdio.h>
int a=14, b=0;
int main() {
  b=a++;
  printf("a=%d  //in main \n", a);
  printf("b=%d  //in main \n", b);
  return 0;
}
```

```
a=15
b=15
```

Example

What will be the z variable value after compiling this code?

Solution


File Edit Search Run Compile Debug Project Options Window Help
Test9.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>
int x=7, y=6, z;
int main() {
  z=x+y*6/2;
  printf("z=%d  //in main \n", z);
return 0;
}
```

```
z=25
```

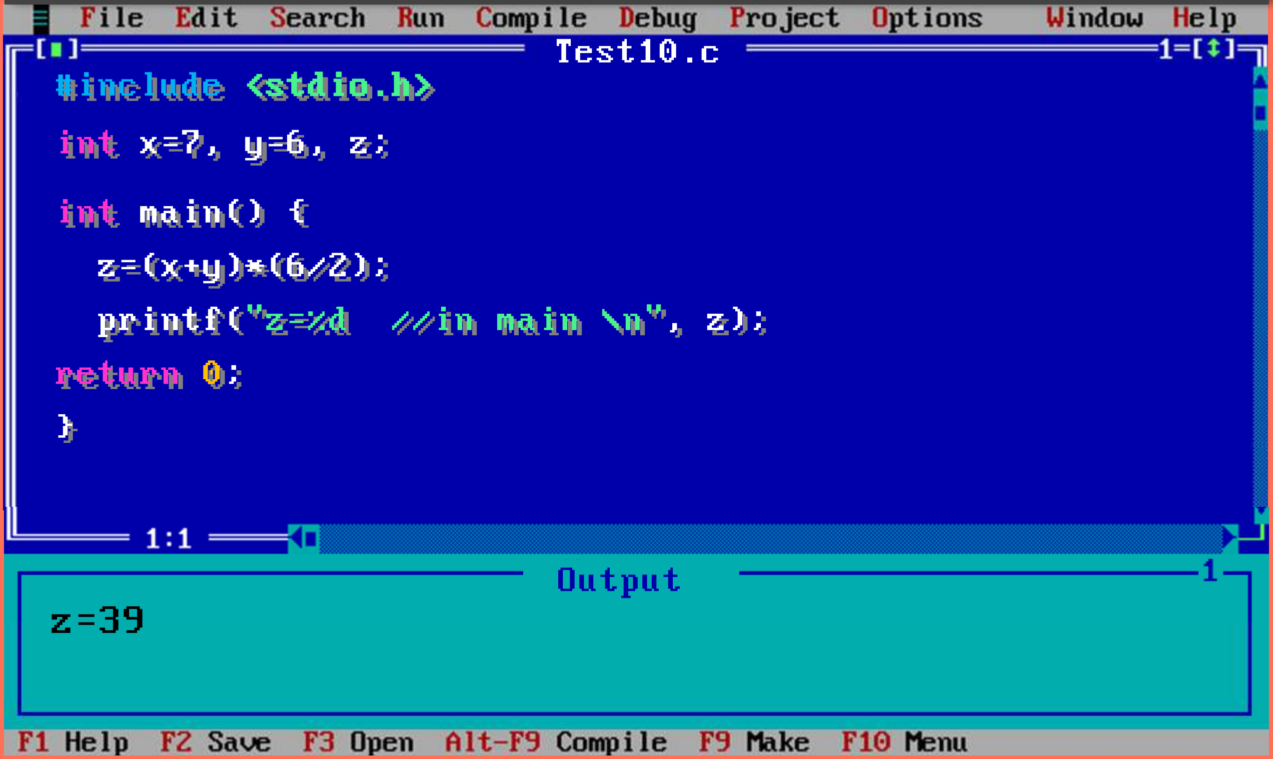
Note

When two operators with the same priority are used, the operator that comes first has priority.

Example

What will be the z variable value after compiling this code?

Solution


File Edit Search Run Compile Debug Project Options Window Help
Test10.c
1
1:1
Output
1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <stdio.h>
int x=7, y=6, z;
int main() {
  z=(x+y)*(6/2);
  printf("z=%d  //in main \n", z);
return 0;
}
```

```
z=39
```

Note

Whenever an expression is enclosed in parentheses, it has higher priority.

Operators in C [AVR]

Relational Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.
- Relational operators are used in **decision making** and **loops**.

Priority	Operator	Definition	Example
First	>	Greater than	a > b
	>=	Greater than or equal to	a >= b
	<	Less than	a < b
	<=	Less than or equal to	a <= b
Second	==	Equal to	a == b
	!=	Not equal to	a != b

Operators in C [AVR]

Logical Operators

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false.
- Logical operators are commonly used in decision making in C programming.

Priority	Operator	Definition	Example
First	!	Logical NOT	!x
Second	&&	Logical AND	a && b
Third	||	Logical OR	a || b

Operators in C [AVR]

Bitwise Operators

- During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.
- Bitwise operators are used in C programming to perform bit-level operations.

Priority	Operator	Definition	Example	Result
First	~	Bitwise complement	~(0x66)	0x99
Second	>>	Shift left	0b11001011>>4	10110000
	<<	Shift Right	0b11001011<<4	00001100
Third	&	Bitwise AND	0x66 & 0xff	0x66
Fourth	^	Bitwise XOR	0x12 ^ 0x12	0
Fifth	|	Bitwise OR	0x05 | 0x30	0x35

Operators in C [AVR]

Assignment Operators

- An assignment operator is used for assigning a value to a variable.
- The most common assignment operator is =.

Operator	Definition	Example	Same as
=	equal to	x=12	
*=	Multiplication assignment	x*=y	x=x*y
/=	Division assignment	x/=y	x=x/y
%=	Division (remainder) assignment	x%=y	x=x%y
+=	Addition assignment	x+=y	x=x+y
-=	Subtraction assignment	x-=y	x=x-y
&=	Logical AND assignment	x&=y	x=x&y
^=	Logical XOR assignment	x^=y	x=x^y
|=	Logical OR assignment	x|=y	x=x|y
<<=	Left Shift assignment	x<<=5	x=x<<5
>>=	Right Shift assignment	x>>=3	x=x<<3

Operators in C [AVR]

Conditional Operators

Syntax: <phrase 1> ? <phrase 2> : <phrase 3>

In this conditional statement, phrase 1 is evaluated. If it is true, phrase 2 is assigned to the variable. Otherwise, phrase 3 is assigned to the variable.

Example: `y = (x>z) ? x:z`

& , * Operators

The & operator allows to access the address of a variable, and the * operator allows to access the content at an address.

Note: When using the * operator, make sure that the data type of the pointer matches the data type of the array or data working with.

Example:

```
unsigned char y;
unsigned char *x, i;
y = 15;
x = &y;
i = *x;
```

Operators in C [AVR]

Comma (,) Operator

This operator is used to separate multiple variables, expressions, and even instruction statements.

Example:

```
char a, b;
b = (a=10 , a/2);
```

sizeof() Operator

This operator returns the length of a variable or data in bytes.

Example:

```
int x, y;
x = sizeof(y);
```

Parentheses Operator

Parentheses are operators that increase the precedence (priority) of the operators within them and are also used in function definitions.

Operators in C [AVR]

Priority of Operator

Operator	Priority	Operator	Priority
( )	1	&	8
! ~ ++ -- sizeof	2	^	9
* / %	3	|	10
+ -	4	&&	11
<< >>	5	||	12
< <= > >=	6	?	13
== !=	7	= += -= *= /= %=	14

instructions in C [AVR]

Instructions in the C language come within the body of a function and perform a specific operation on global and local variables.

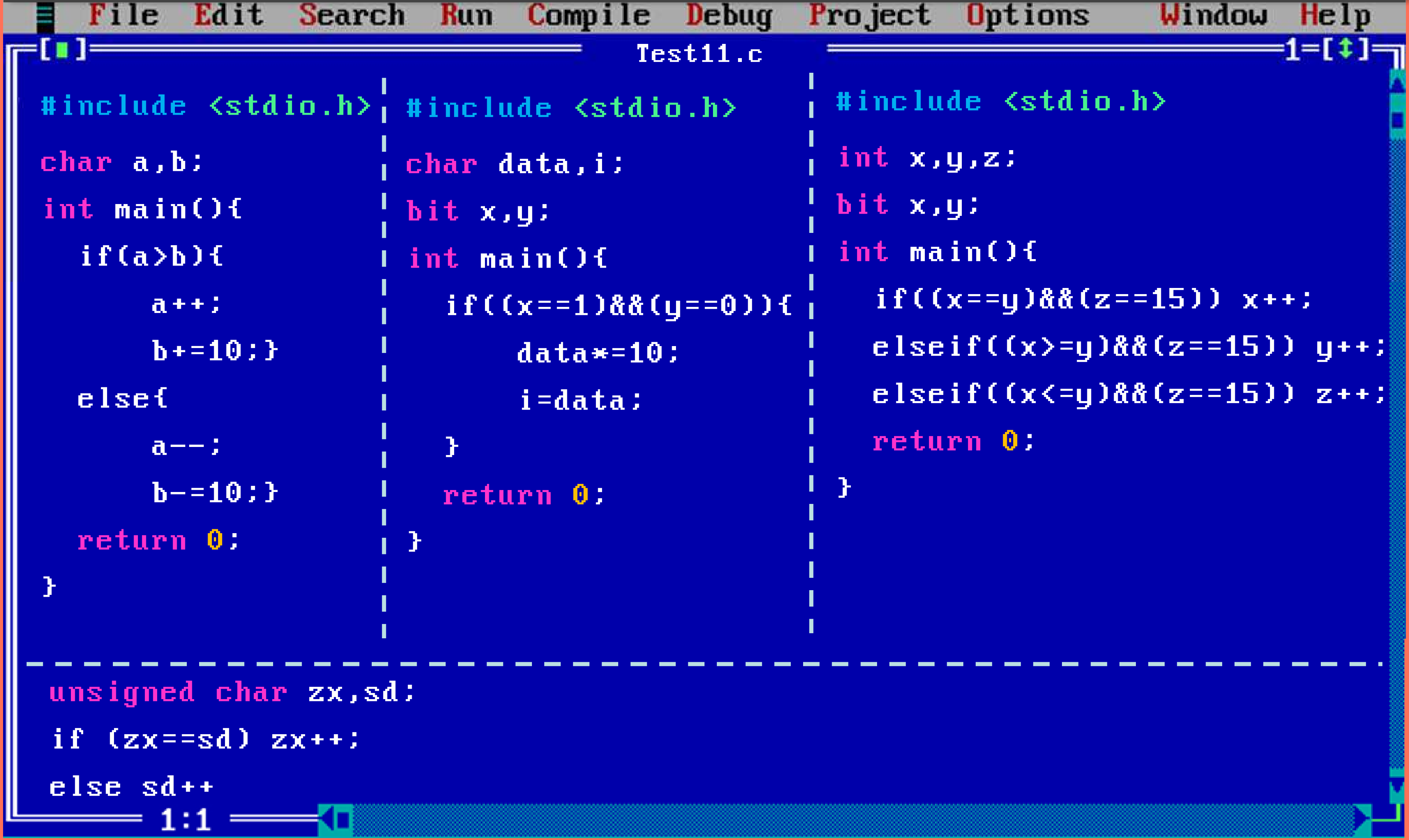
if-else Conditional Instruction

This statement is used to evaluate a conditional statement in a program.

Syntax:

```
If(condition){
    Instruction set 1;
}
else{
    Instruction set 2;
}
```

Example



```
#include <stdio.h>
char a,b;
int main(){
  if(a>b){
      a++;
      b+=10;}
  else{
      a--;
      b-=10;}
  return 0;
}
```

```
#include <stdio.h>
char data,i;
bit x,y;
int main(){
  if((x==1)&&(y==0)){
      data*=10;
      i=data;
  }
  return 0;
}
```

```
#include <stdio.h>
int x,y,z;
bit x,y;
int main(){
  if((x==y)&&(z==15)) x++;
  elseif((x>=y)&&(z==15)) y++;
  elseif((x<=y)&&(z==15)) z++;
  return 0;
}
```

```
unsigned char zx,sd;
if (zx==sd) zx++;
else sd++
```

instructions in C [AVR]

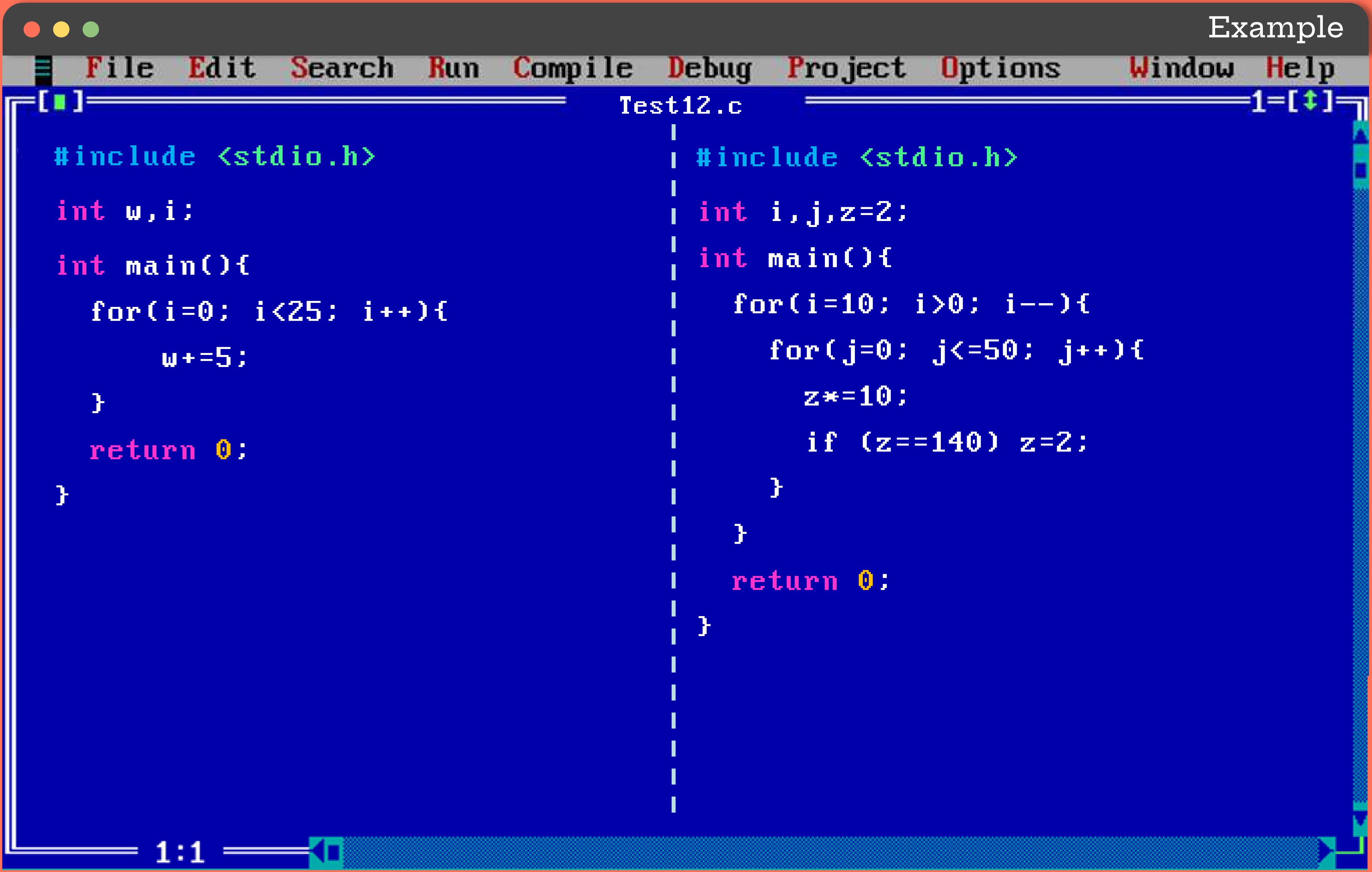
for Instruction

This statement is used for conditional loop structures in a program when a loop counter is needed.

Syntax:

```
for (Initialization Statement; Conditional Statement; Counter Statement){

    // statements inside the body of loop

}
```

Example



```
#include <stdio.h>

int w,i;

int main(){
  for(i=0; i<25; i++){
      w+=5;
  }
  return 0;
}
```

```
#include <stdio.h>

int i,j,z=2;
int main(){
  for(i=10; i>0; i--){
    for(j=0; j<=50; j++){
      z*=10;
      if (z==140) z=2;
    }
  }
  return 0;
}
```

while Conditional Loop Instruction

This statement is also used to execute statements in a program loop, but it does not have a loop counter. This statement first tests the loop condition, and if the condition is true, the program line enters the loop, otherwise the loop is never executed.

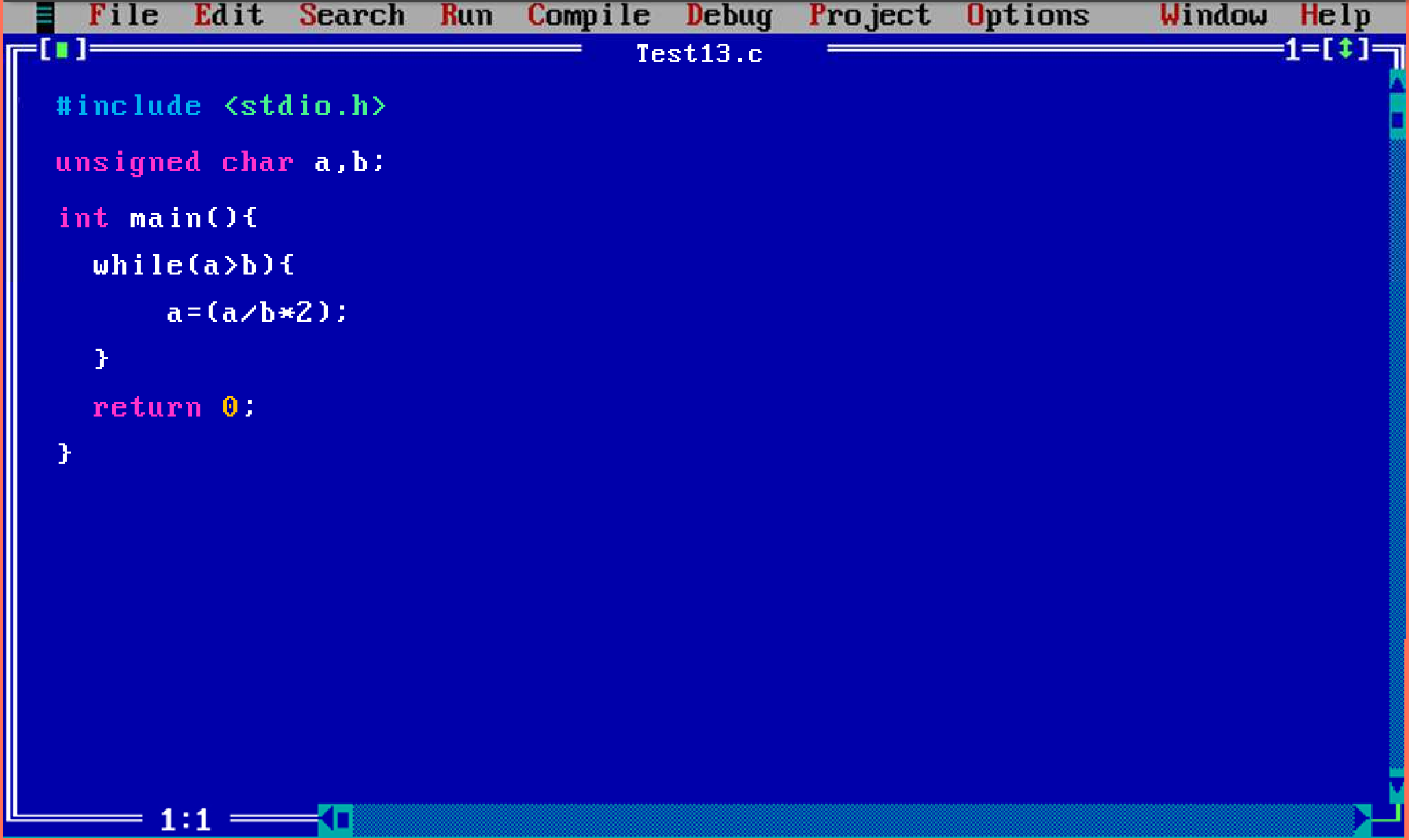
Note: In other words, this statement checks the condition after each iteration of the loop. If the result of the loop condition is true (non-zero), the loop continues, otherwise it exits the loop.

```
Syntax:
while (Conditional Statement){
    // statements inside the body of loop
}
```

Infinite loop using the while statement:

```
while (1){
    // statements inside the body of loop
}
```

Example



```
#include <stdio.h>
unsigned char a,b;
int main(){
  while(a>b){
      a=(a/b*2);
  }
  return 0;
}
```

do-while Conditional Loop Instruction

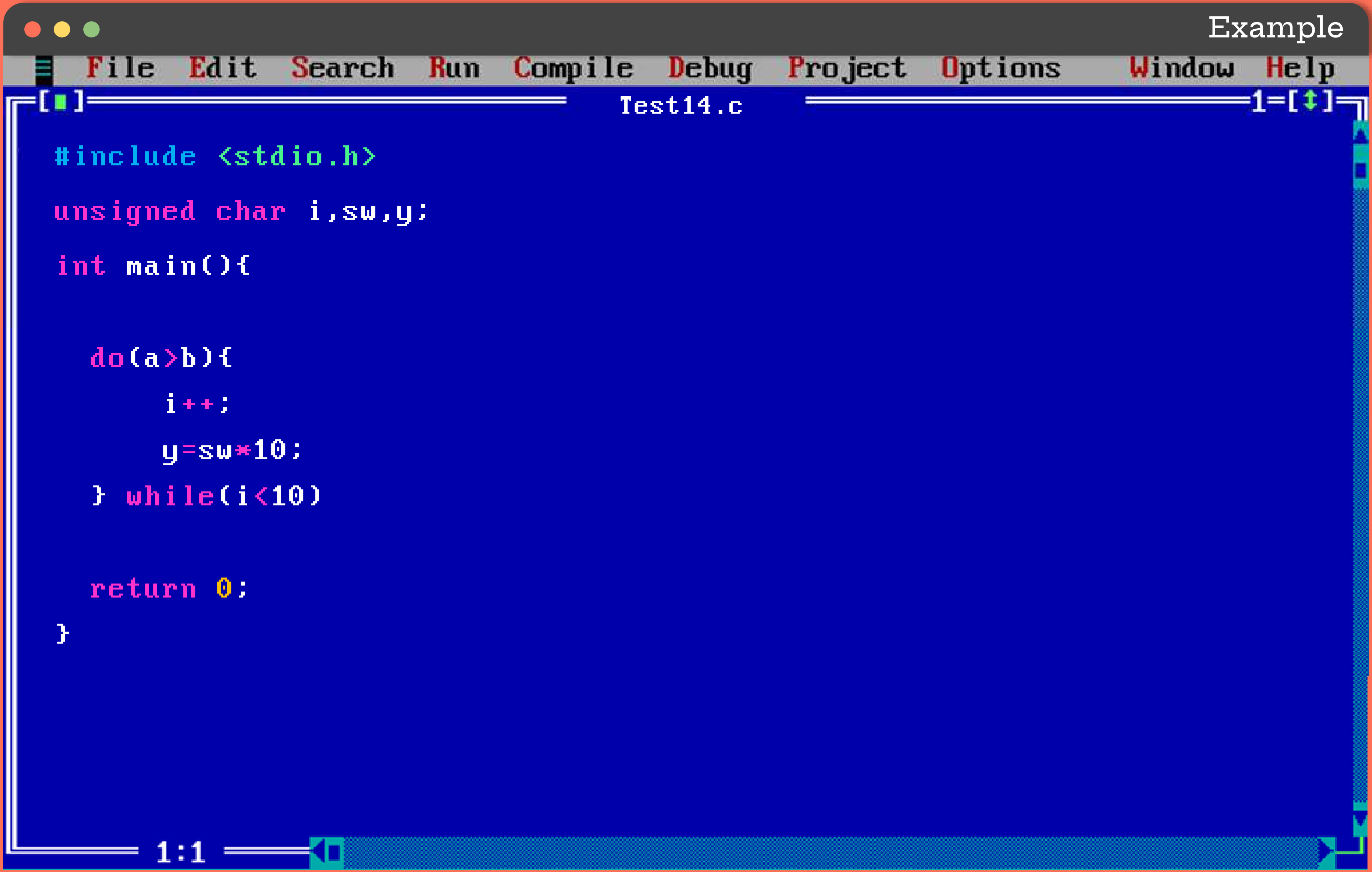
This statement is similar to the while loop, but it allows the loop to be executed once before checking the condition at the end of the loop.

Syntax:

```
do {
    // statements inside the body of loop
} while (Conditional Statement)
```

Infinite loop using the while statement:

```
do {
    // statements inside the body of loop
} while (1)
```

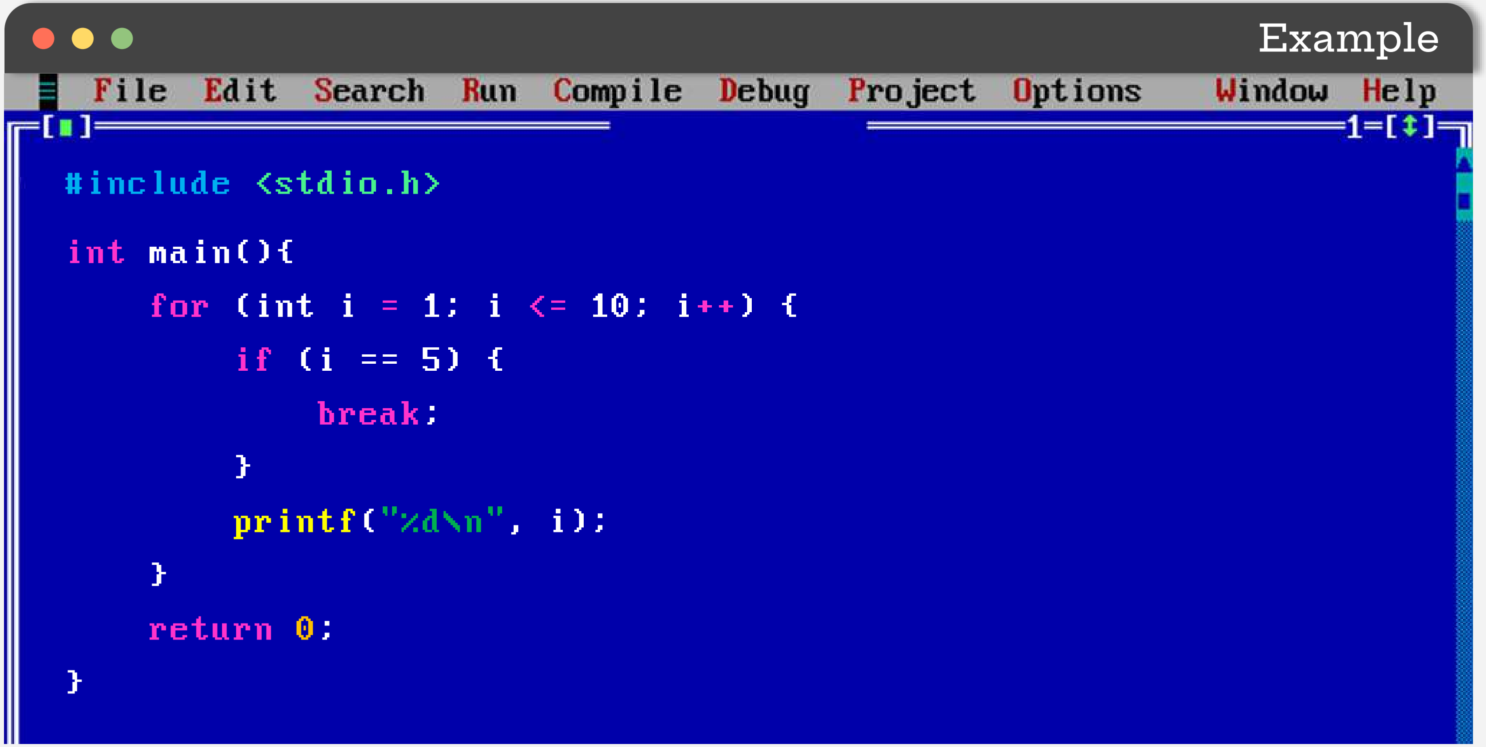


```
#include <stdio.h>
unsigned char i,sw,y;
int main(){

  do(a>b){
      i++;
      y=sw*10;
  } while(i<10)

  return 0;
}
```

instructions in C [AVR]

break Instruction

This statement is used to break out of a loop unconditionally. Whenever the program reaches this statement, it will exit the loop structure (i.e., the {}) and continue with the first instruction after the loop structure.


Example
File Edit Search Run Compile Debug Project Options Window Help
[■]
1=[↕]

```
#include <stdio.h>

int main(){
    for (int i = 1; i <= 10; i++) {
        if (i == 5) {
            break;
        }
        printf("%d\n", i);
    }
    return 0;
}
```

instructions in C [AVR]

switch Instruction

- Switch statement in C tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.

- Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found.

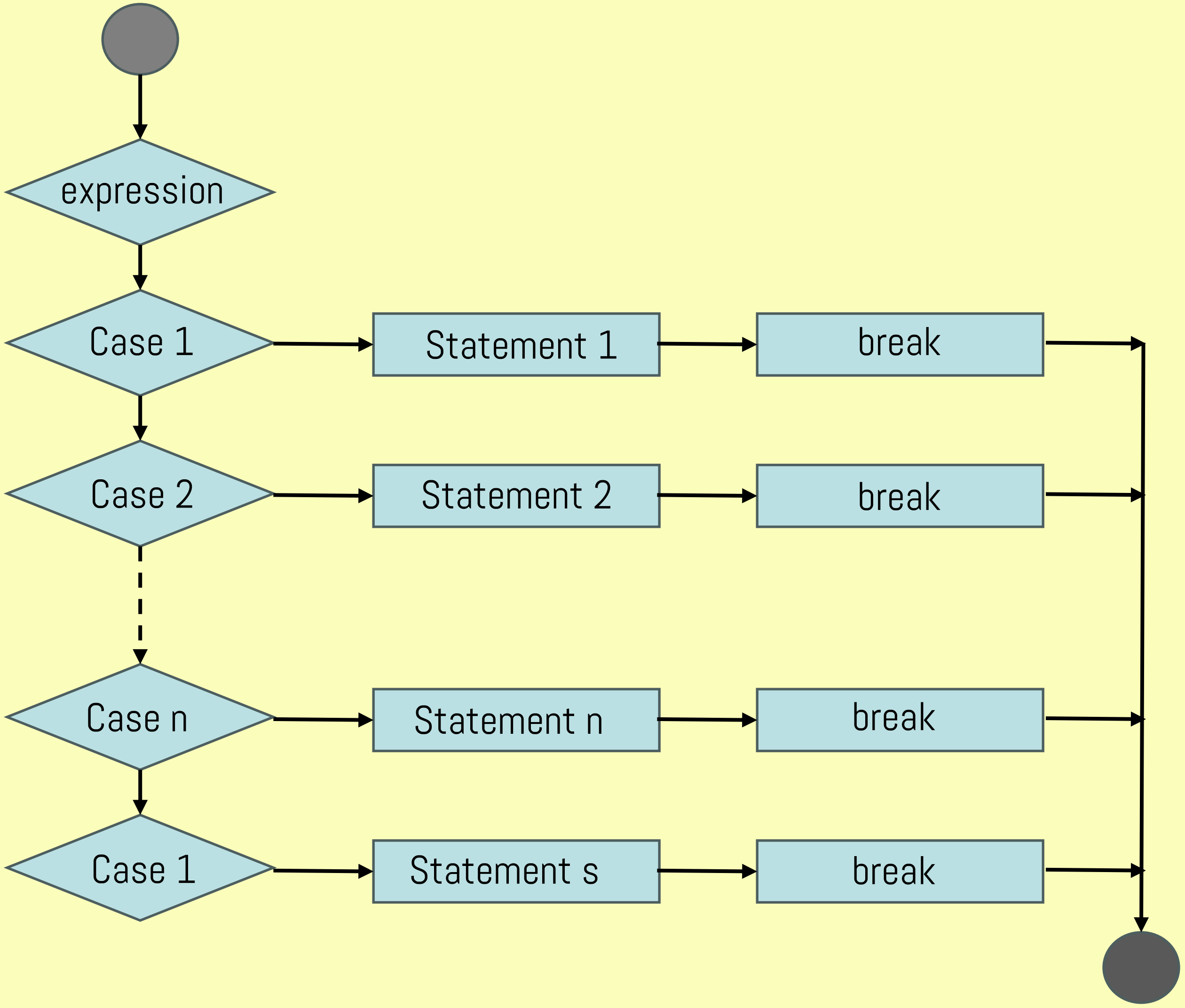
- If a case match is NOT found, then the default statement is executed, and the control goes out of the switch block.

Syntax:

```
switch (expression)
{
   case constant1:
     // statements
     break;

   case constant2:
     // statements
     break;
   .
   .
   .
   default:
     // default statements
}
```

instructions in C [AVR]

switch Instruction



expression

Case 1

Statement 1

break

Case 2

Statement 2

break

Case n

Statement n

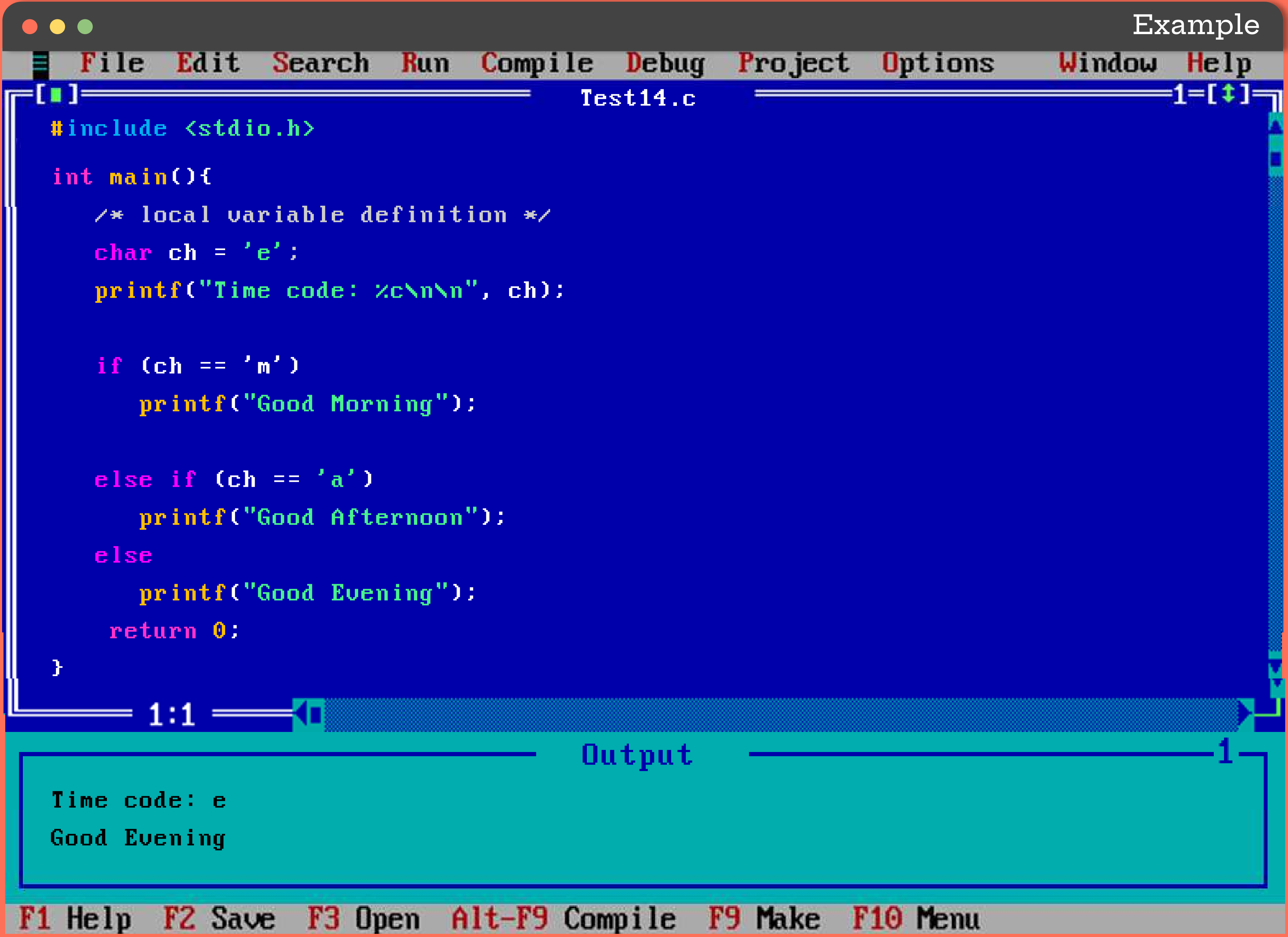
break

Case 1

Statement s

break

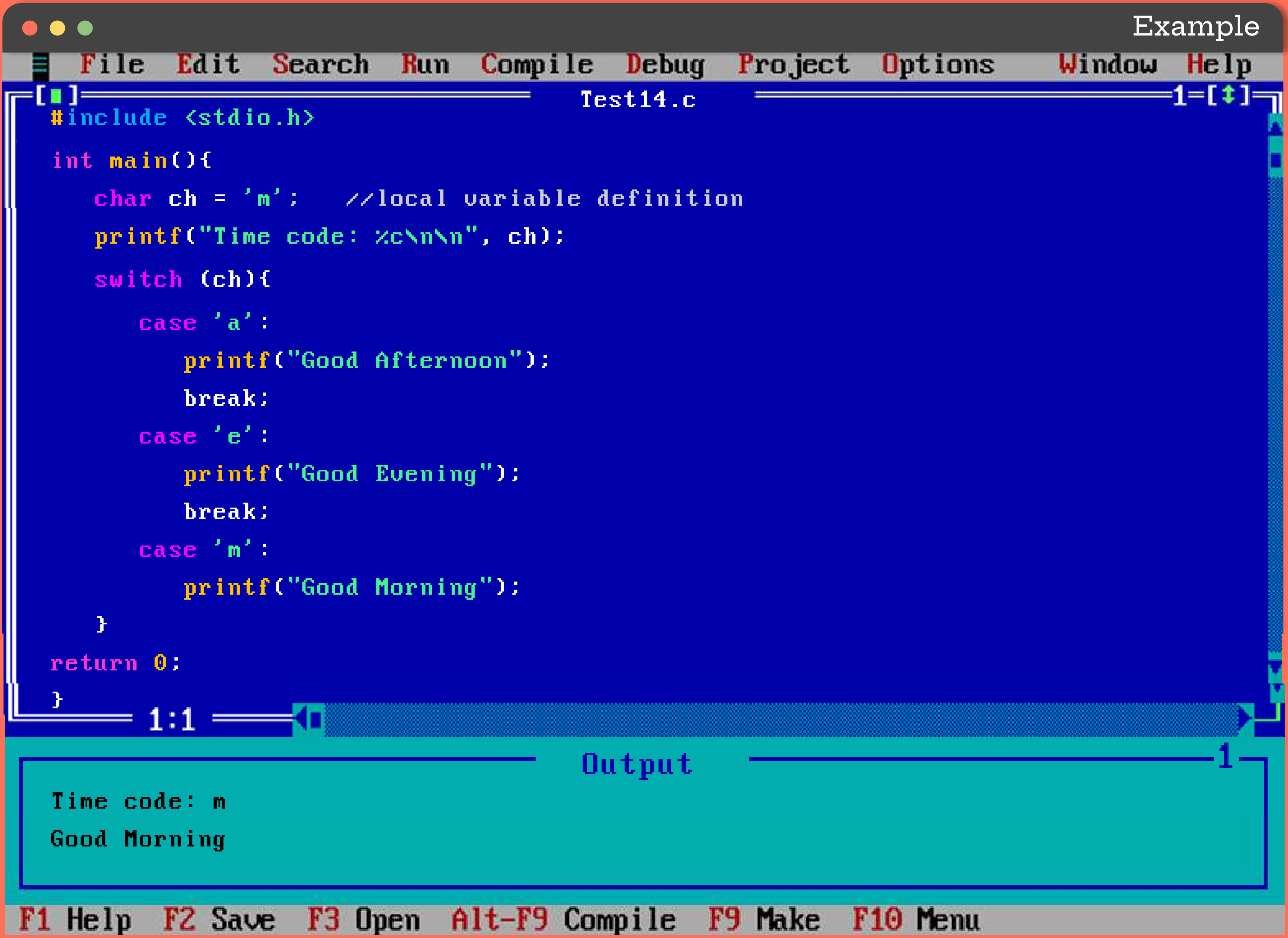
Example


File Edit Search Run Compile Debug Project Options Window Help
[■]
Test14.c
1=[↕]
1:1
Output
1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <stdio.h>
int main(){
   /* local variable definition */
   char ch = 'e';
   printf("Time code: %c\n\n", ch);

   if (ch == 'm')
      printf("Good Morning");

   else if (ch == 'a')
      printf("Good Afternoon");
   else
      printf("Good Evening");
    return 0;
}
```

```
Time code: e
Good Evening
```

Example


File Edit Search Run Compile Debug Project Options Window Help
Test14.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>

int main(){
   char ch = 'm';    //local variable definition
   printf("Time code: %c\n\n", ch);
   switch (ch){
      case 'a':
         printf("Good Afternoon");
         break;
      case 'e':
         printf("Good Evening");
         break;
      case 'm':
         printf("Good Morning");
   }
return 0;
}
```

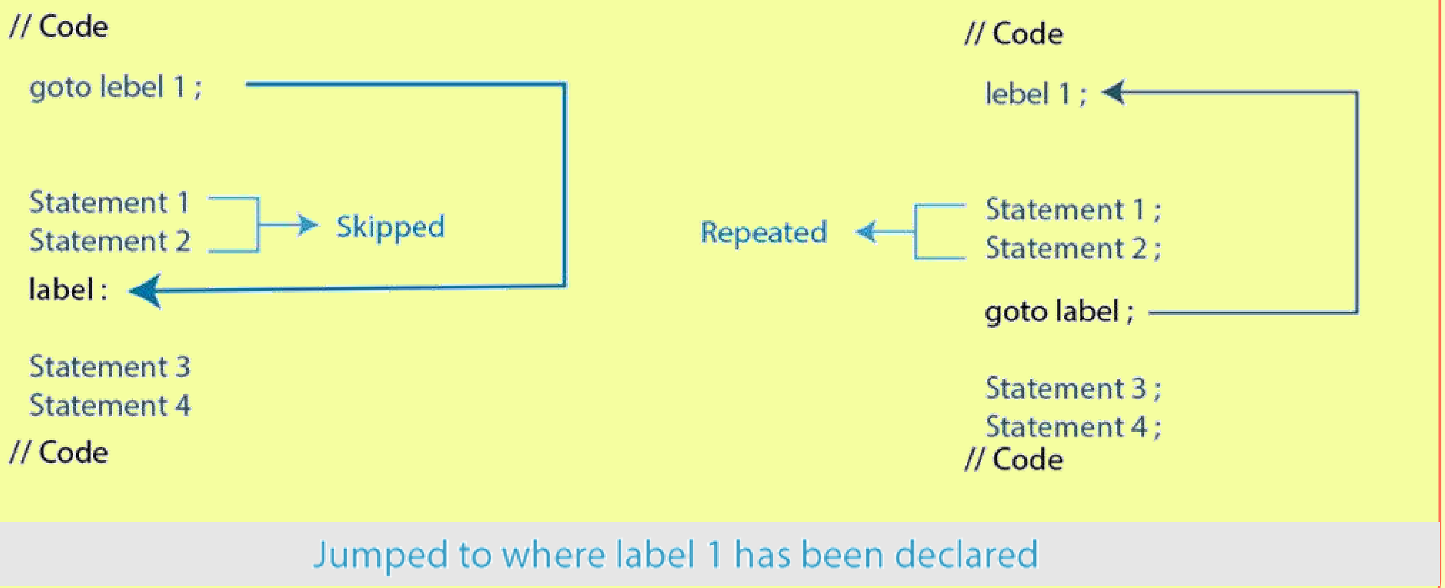
```
Time code: m
Good Morning
```

instructions in C [AVR]

goto Instruction

The C goto statement is a jump statement which is sometimes also referred to as an unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:



```
// Code
goto lebel 1 ;

Statement 1
Statement 2
label :

Statement 3
Statement 4
// Code
```

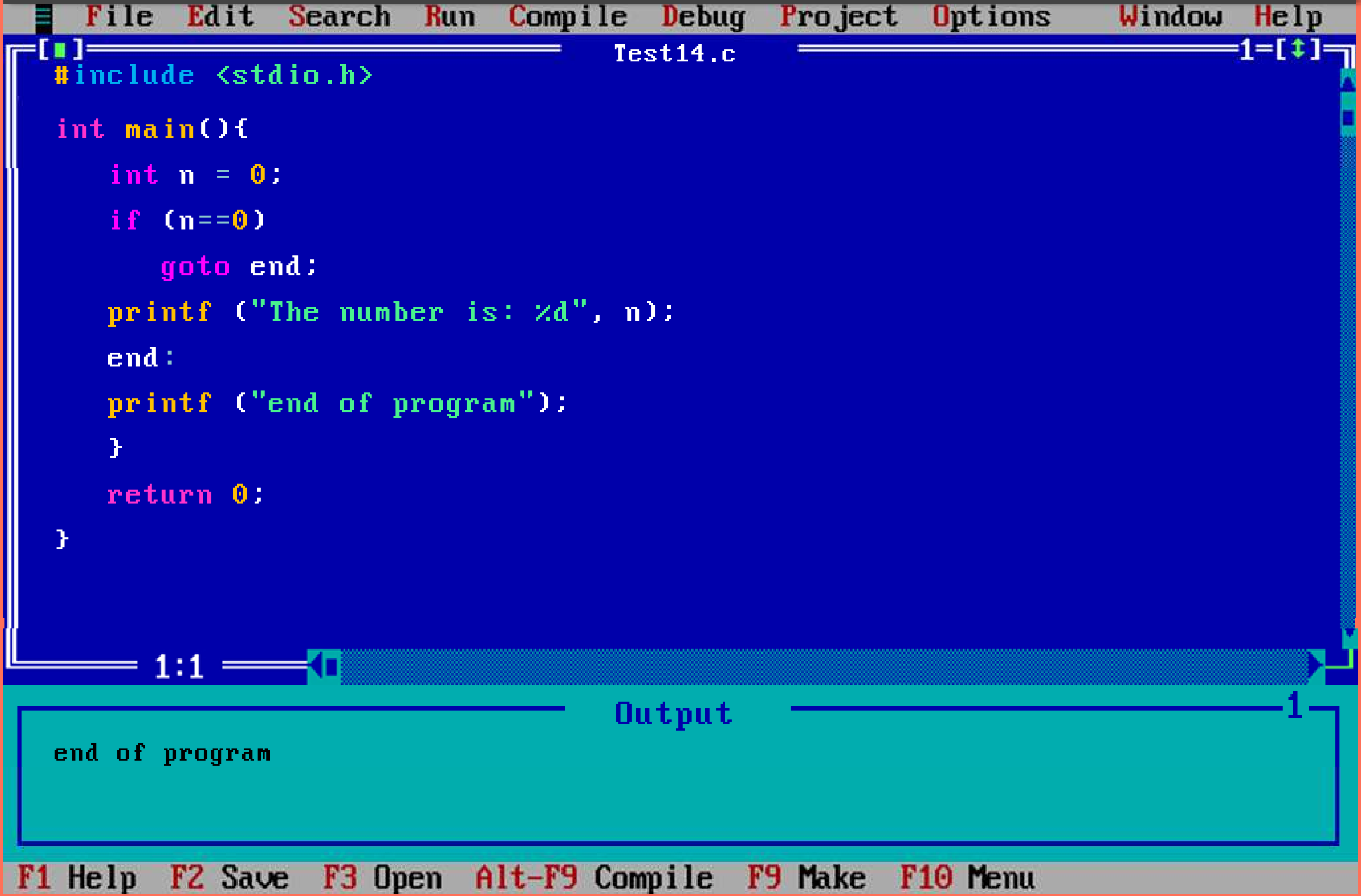
Skipped

```
// Code
lebel 1 ;

Statement 1 ;
Statement 2 ;

goto label ;

Statement 3 ;
Statement 4 ;
// Code
```

Repeated

Jumped to where label 1 has been declared

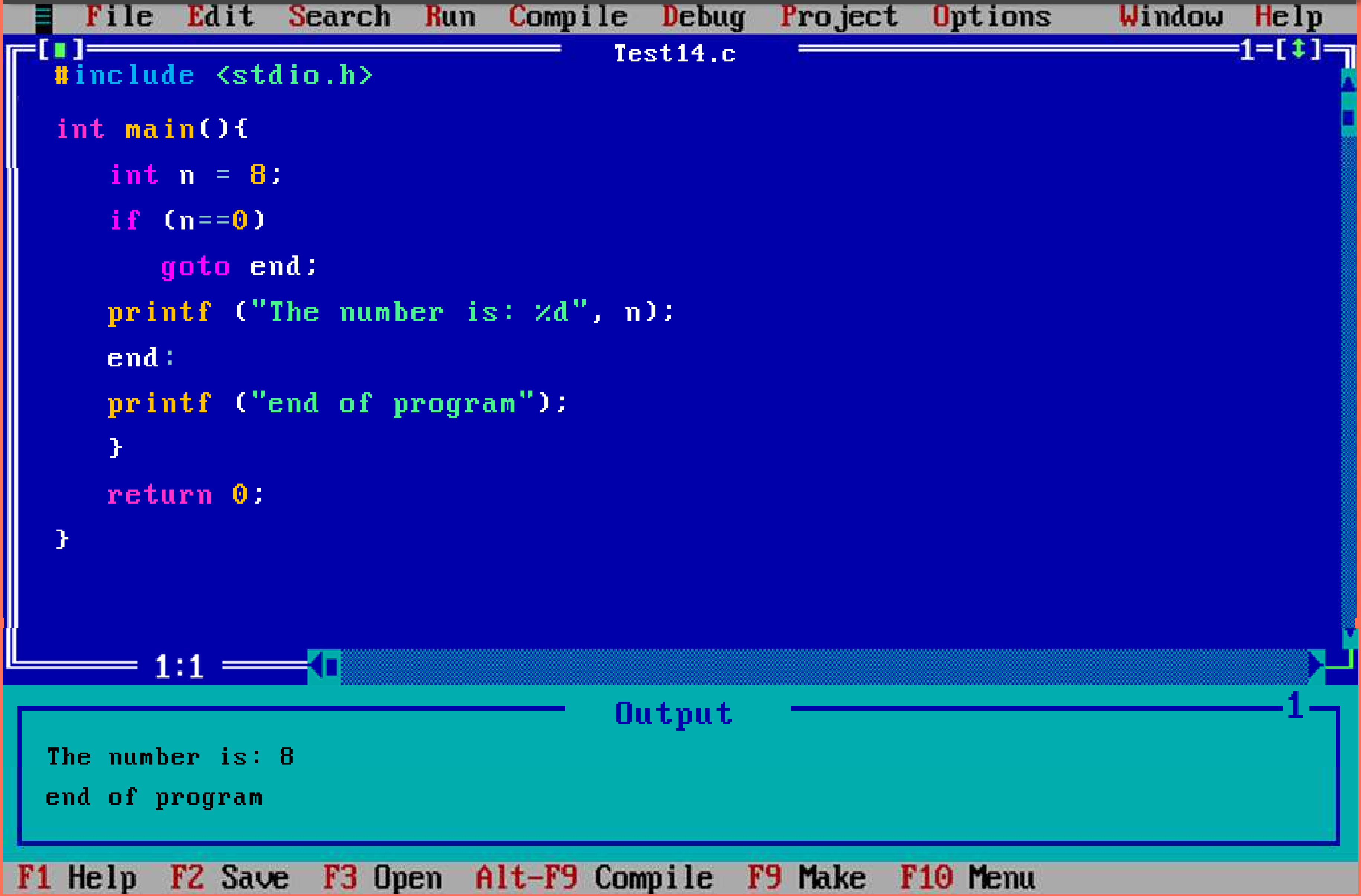
Example


File Edit Search Run Compile Debug Project Options Window Help
Test14.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <stdio.h>

int main(){
   int n = 0;
   if (n==0)
      goto end;
   printf ("The number is: %d", n);
   end:
   printf ("end of program");
   }
   return 0;
}
```

```
end of program
```

Example


≡ File Edit Search Run Compile Debug Project Options Window Help
[■]
Test14.c
1=[↕]
1:1
Output
1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

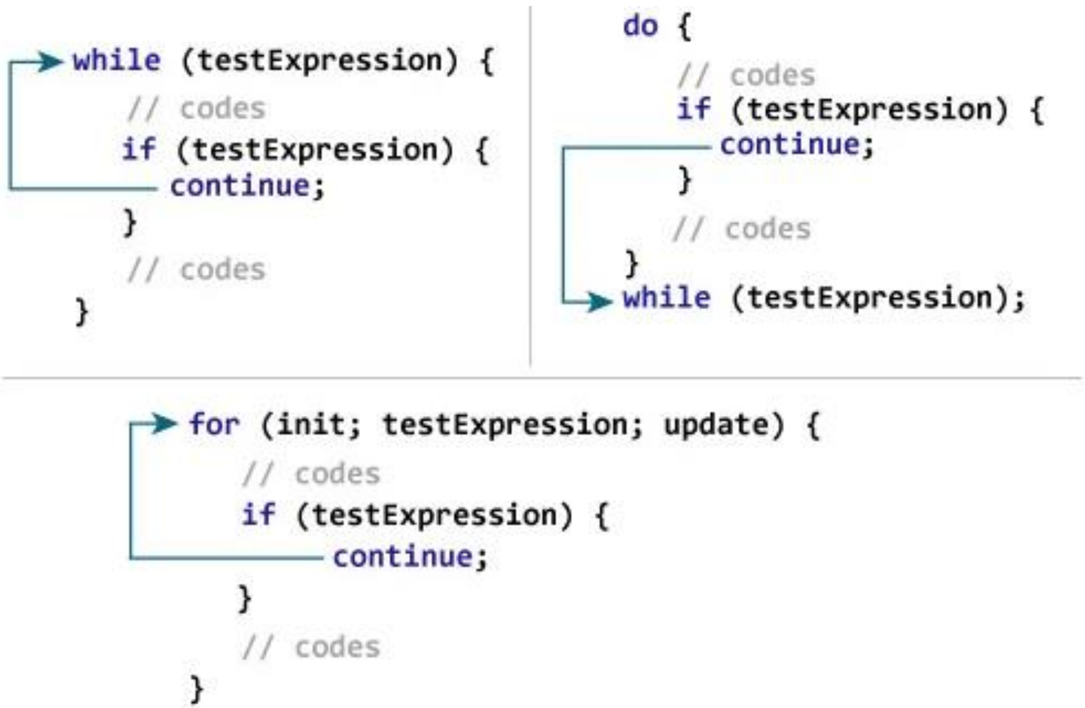
```
#include <stdio.h>

int main(){
   int n = 8;
   if (n==0)
      goto end;
   printf ("The number is: %d", n);
   end:
   printf ("end of program");
   }
   return 0;
}
```

```
The number is: 8
end of program
```

continue Instruction

The **continue** statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Syntax:

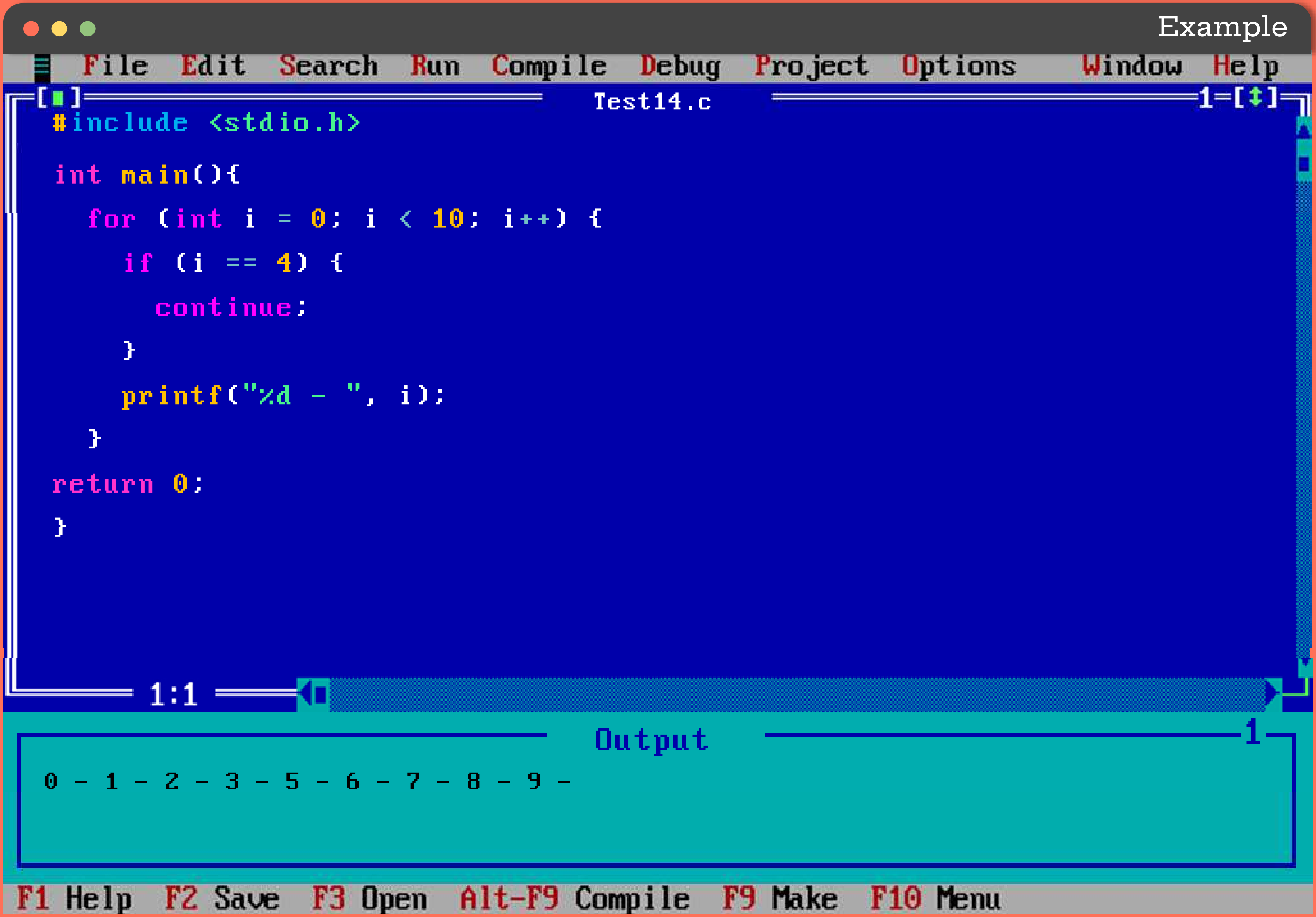


```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

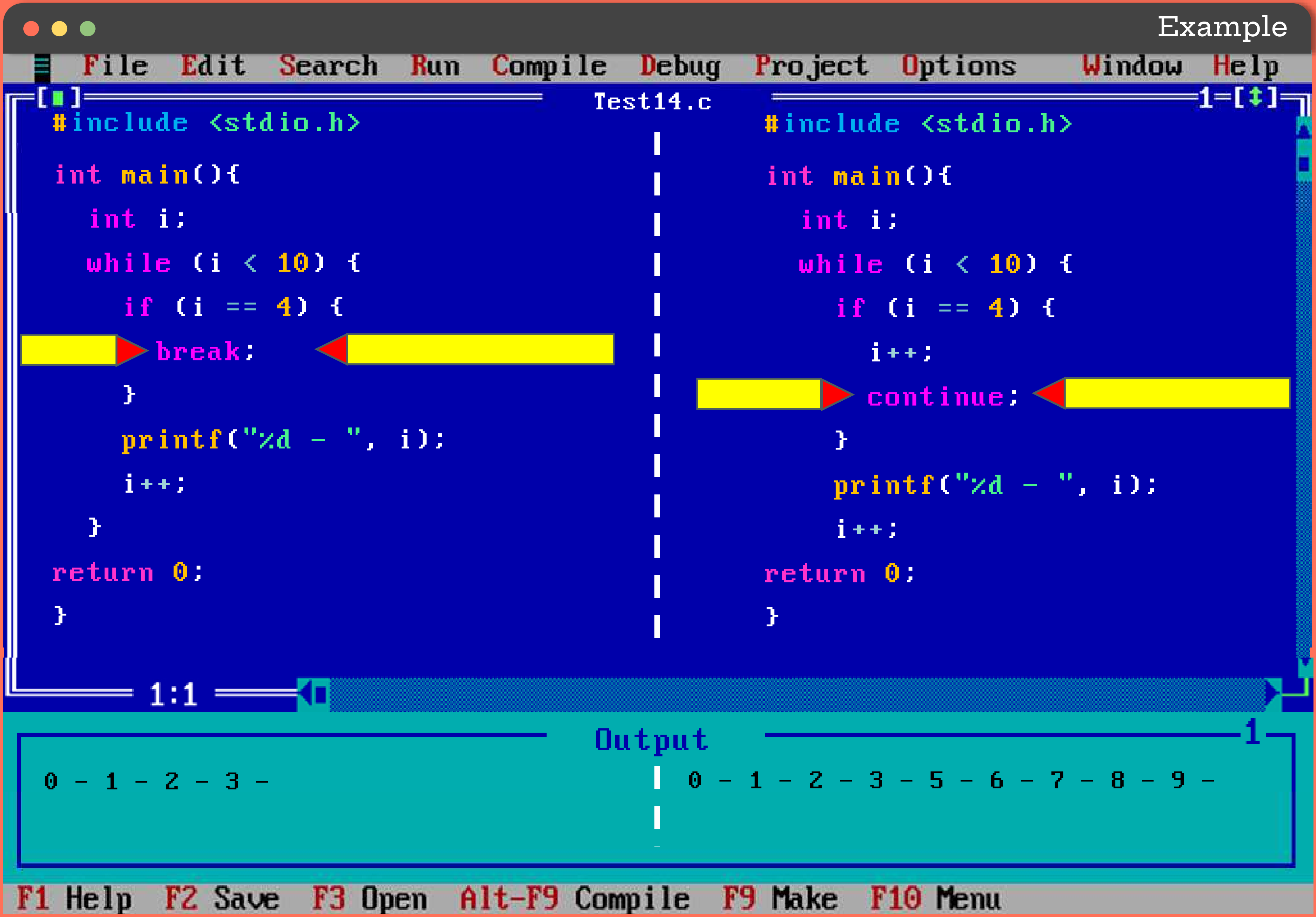
```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Example



```c
#include <stdio.h>

int main(){
 for (int i = 0; i < 10; i++) {
  if (i == 4) {
   continue;
  }
  printf("%d - ", i);
 }
return 0;
}
```

```
0 - 1 - 2 - 3 - 5 - 6 - 7 - 8 - 9 -
```


Example
File Edit Search Run Compile Debug Project Options Window Help
Test14.c
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <stdio.h>

int main(){
  int i;
  while (i < 10) {
    if (i == 4) {
      break;
    }
    printf("%d - ", i);
    i++;
  }
return 0;
}
```

```c
#include <stdio.h>

int main(){
  int i;
  while (i < 10) {
    if (i == 4) {
      i++;
      continue;
    }
    printf("%d - ", i);
    i++;
return 0;
}
```

Output

```
0 - 1 - 2 - 3 -
```

```
0 - 1 - 2 - 3 - 5 - 6 - 7 - 8 - 9 -
```

instructions in C [AVR]

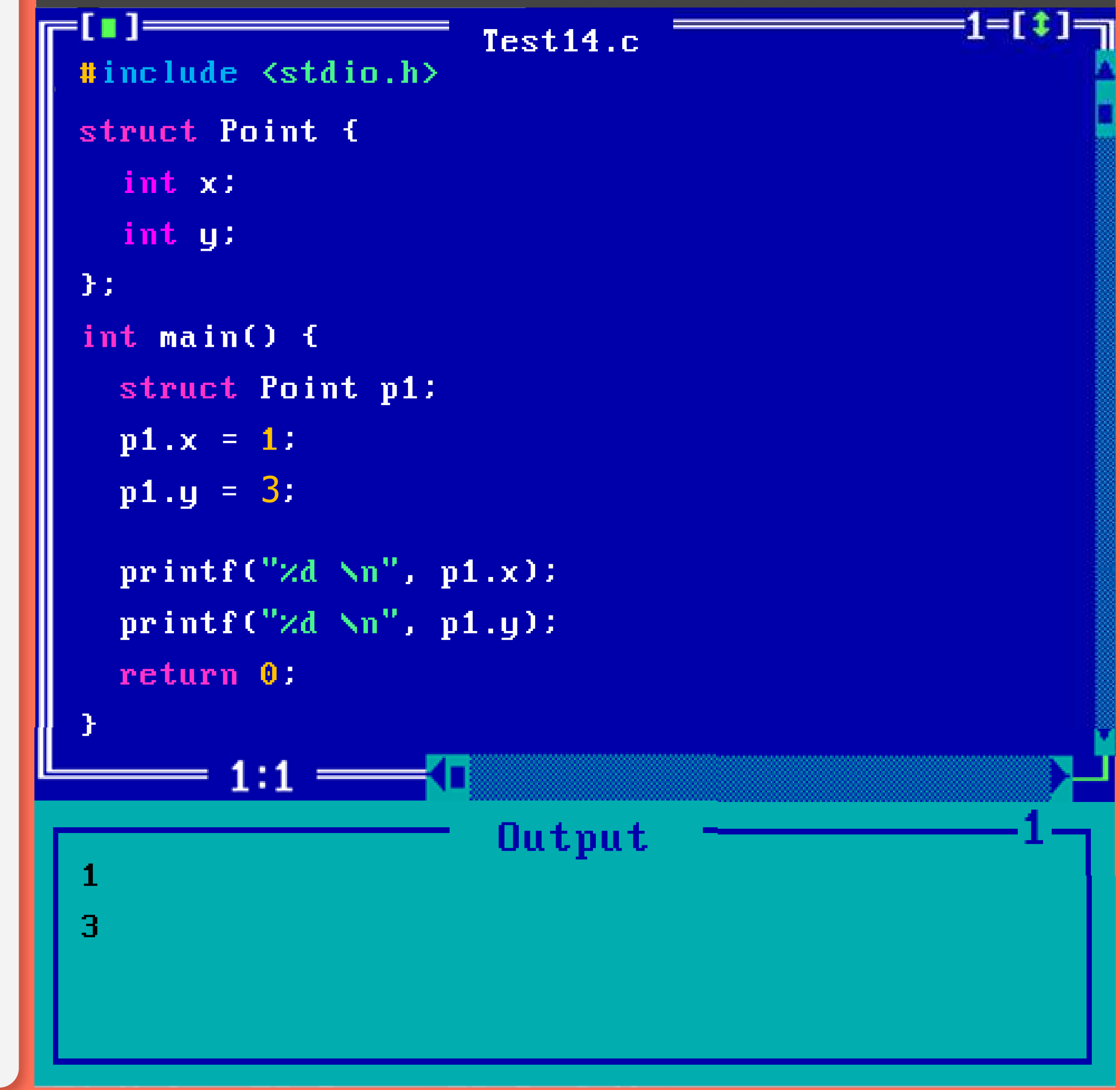
typedef

typedef keyword allow users to provide alternative names for the primitive (e.g., int) and user-defined (e.g struct) data types.

- ➢ Remember, this keyword adds a new name for some existing data type but does not create a new type.

Syntax:
typedef <existing_type> <alias>

Example



```
#include <stdio.h>
struct Point {
   int x;
   int y;
};
int main() {
   struct Point p1;
   p1.x = 1;
   p1.y = 3;

   printf("%d \n", p1.x);
   printf("%d \n", p1.y);
   return 0;
}
```

```
1
3
```

instructions in C [AVR]

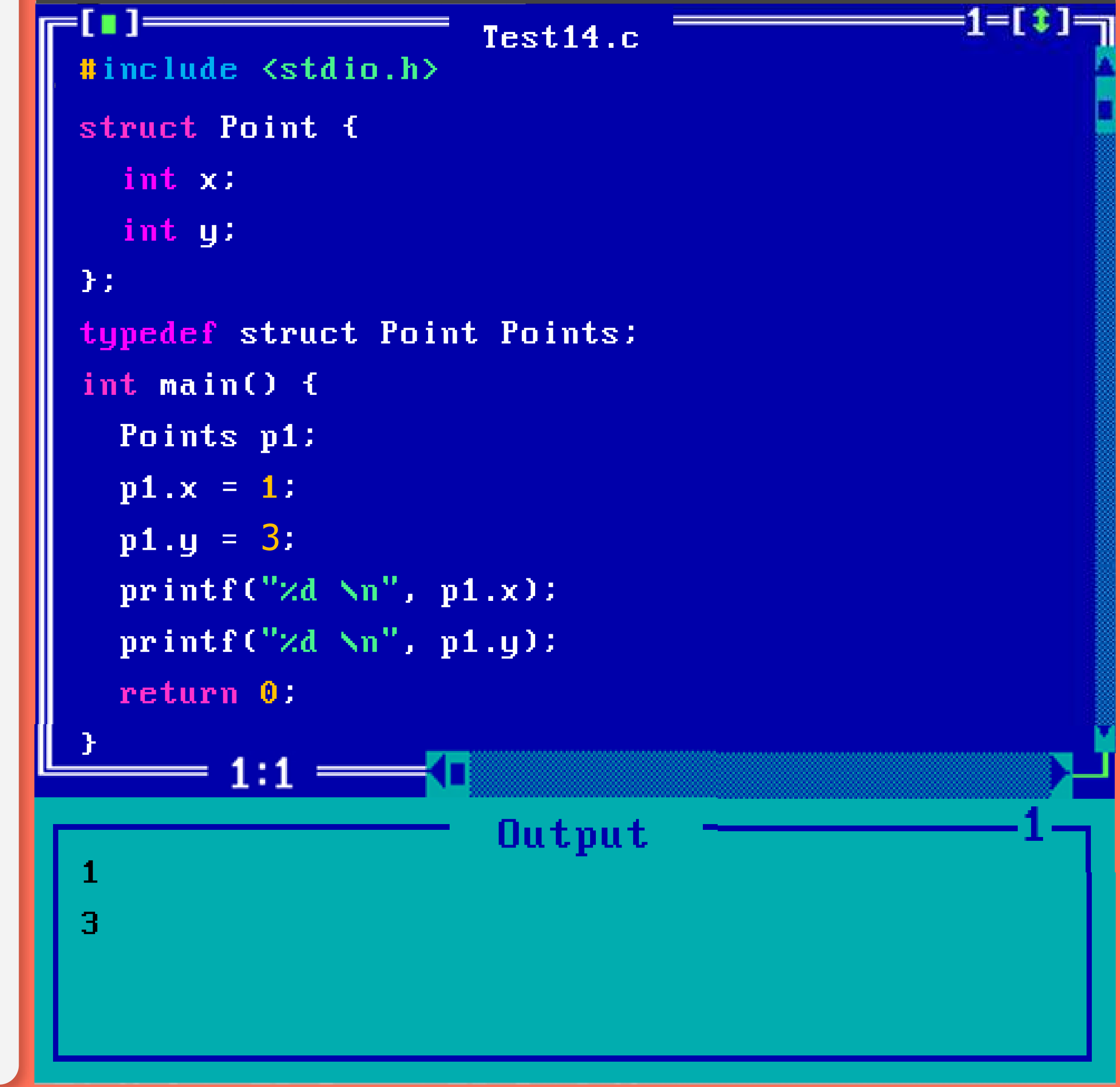
typedef

typedef keyword allow users to provide alternative names for the primitive (e.g., int) and user-defined (e.g struct) data types.

➢ Remember, this keyword adds a new name for some existing data type but does not create a new type.

Syntax:
typedef <existing_type> <alias>

Example – typedef [Methode 1]



```
#include <stdio.h>
struct Point {
   int x;
   int y;
};
typedef struct Point Points;
int main() {
   Points p1;
   p1.x = 1;
   p1.y = 3;
   printf("%d \n", p1.x);
   printf("%d \n", p1.y);
   return 0;
}
```

```
1
3
```

instructions in C [AVR]

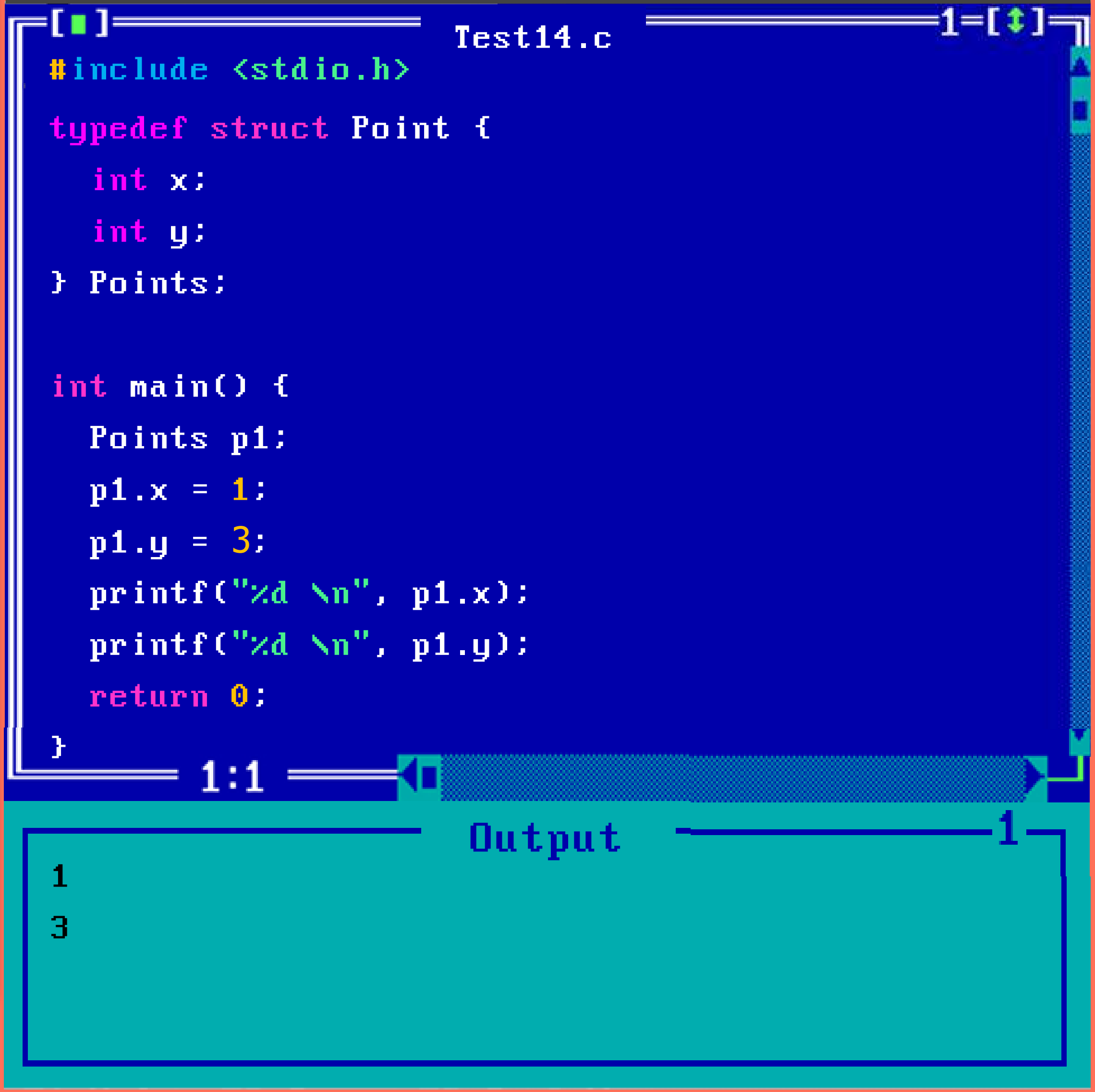
typedef

typedef keyword allow users to provide alternative names for the primitive (e.g., int) and user-defined (e.g struct) data types.

- ➢ Remember, this keyword adds a new name for some existing data type but does not create a new type.

Syntax:
typedef <existing_type> <alias>

Example – typedef [Methode 2]



```
#include <stdio.h>
typedef struct Point {
   int x;
   int y;
} Points;

int main() {
  Points p1;
  p1.x = 1;
  p1.y = 3;
  printf("%d \n", p1.x);
  printf("%d \n", p1.y);
  return 0;
}
```

```
1
3
```

Functions in C [AVR]

- A function in C is a set of statements that when called, perform some specific task.
- The programming statements of a function are enclosed within { } braces, having certain meanings and performing certain operations.

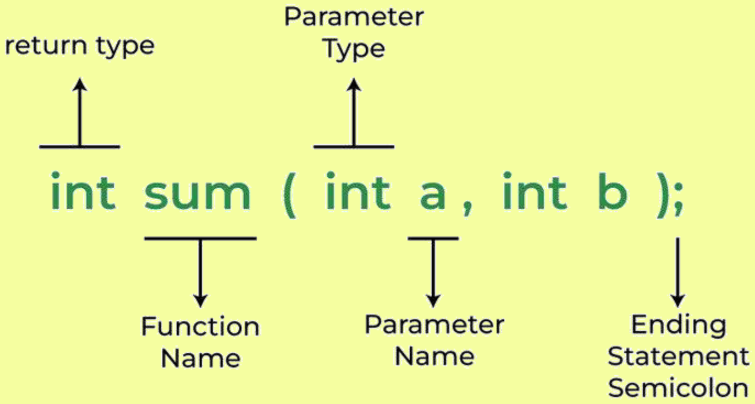
Syntax of Functions in C [Declaration]

A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.

```
return_type   name_of_the_function (parameter_1, parameter_2);
```

Example

```
int sum(int a, int b);
int sum(int , int);
```



Functions in C [AVR]

- The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

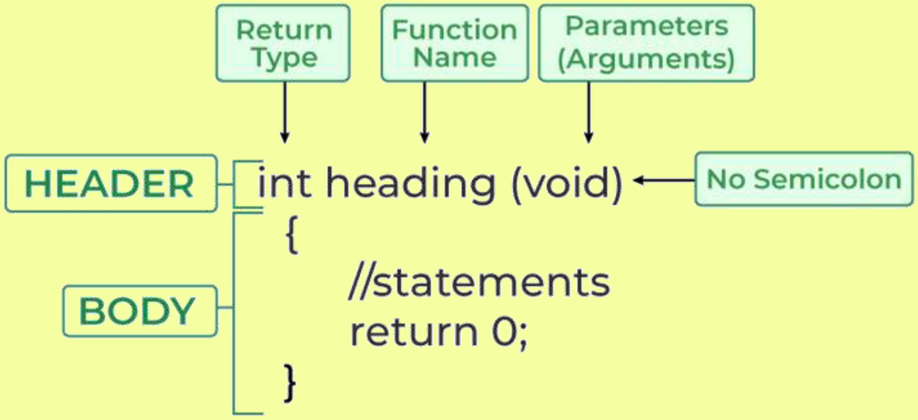
Syntax of Functions in C [Definition]

A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.

```
return_type  function_name (para1_type para1_name, para2_type para2_name)
{
   // body of the function
}
```

Example

```
int sum(int a, int b);
int sum(int , int);
```



```
int heading (void)
 {
       //statements
       return 0;
 }
```

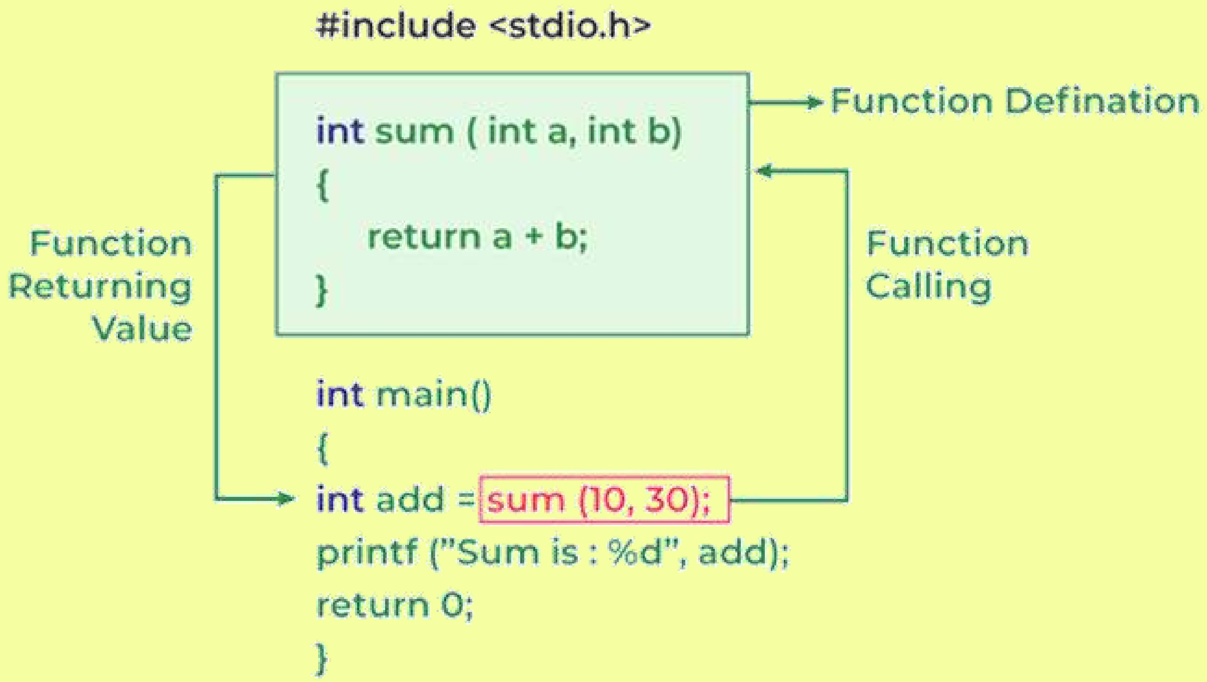
Functions in C [AVR]

- The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

Syntax of Functions in C [call]

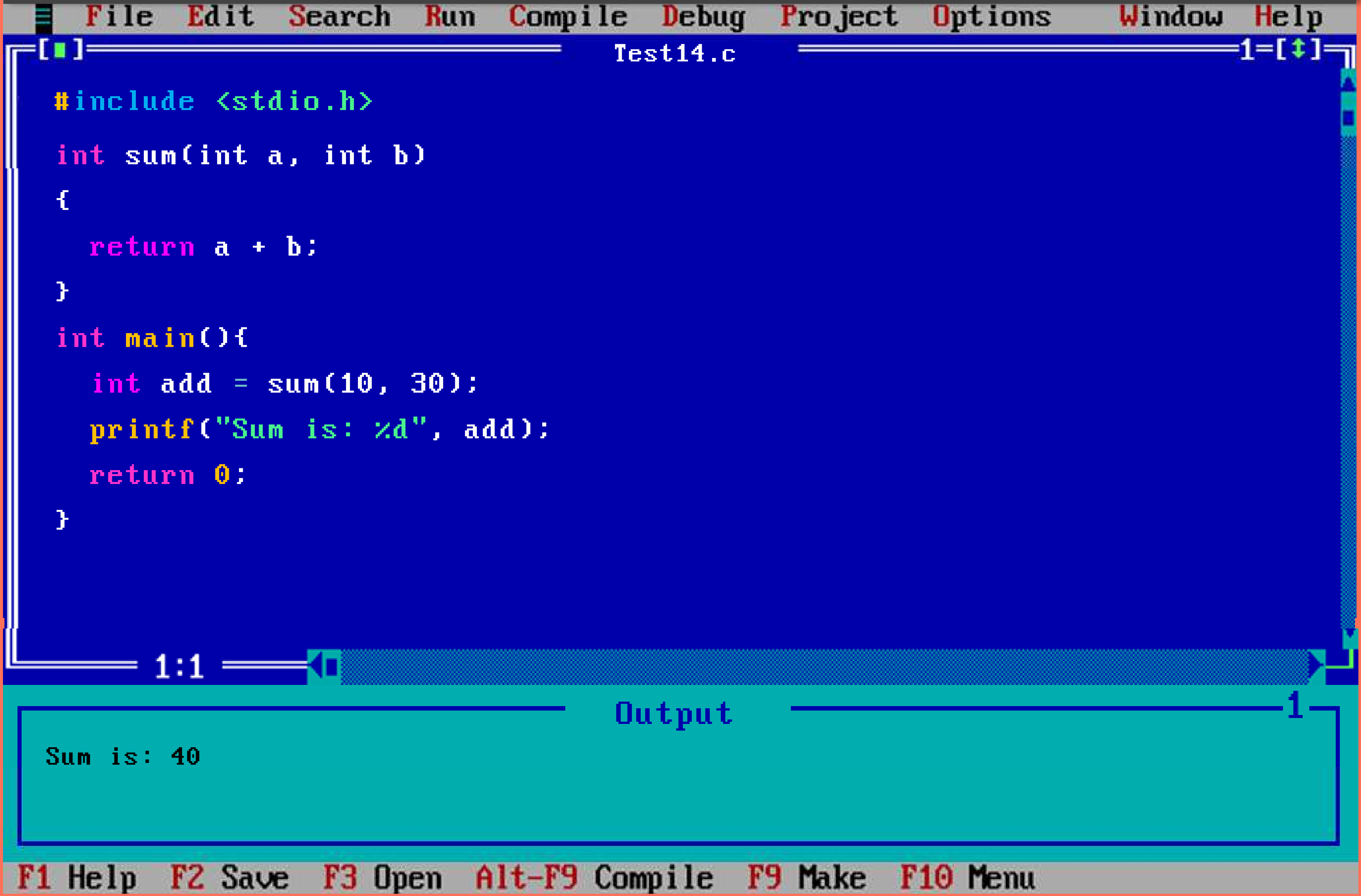
A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.

Example



```
#include <stdio.h>

int sum ( int a, int b)
{
    return a + b;
}

int main()
{
int add = sum (10, 30);
printf ("Sum is : %d", add);
return 0;
}
```

Example



```c
#include <stdio.h>
int sum(int a, int b)
{
  return a + b;
}
int main(){
  int add = sum(10, 30);
  printf("Sum is: %d", add);
  return 0;
}
```

```
Sum is: 40
```

Functions in C [AVR]

Function Return Type

Function return type tells what type of value is returned after all function is executed. When we don't want to return a value, we can use the void data type.

Example

```
Int  func(parameter_1,parameter_2);
```

The above function will return an integer value after running statements inside the function.

Note: Only one value can be returned from a C function. To return multiple values, we have to use pointers or structures.

Function Arguments

Function Arguments (also known as Function Parameters) are the data that is passed to a function.

Example

```
Int  function_name(int var1, int var2);
```

Functions in C [AVR]

Types of Function

There are two types of functions in C:

1.Library Functions
2.User Defined Functions

Library Functions:

- ✓ A library function is also referred to as a “built-in function”.
- ✓ A compiler package already contains these functions, each of which has a specific meaning and is included in the package.
- ✓ Built-in functions have the advantage of being directly usable without being defined, whereas user-defined functions must be declared and defined before being used.

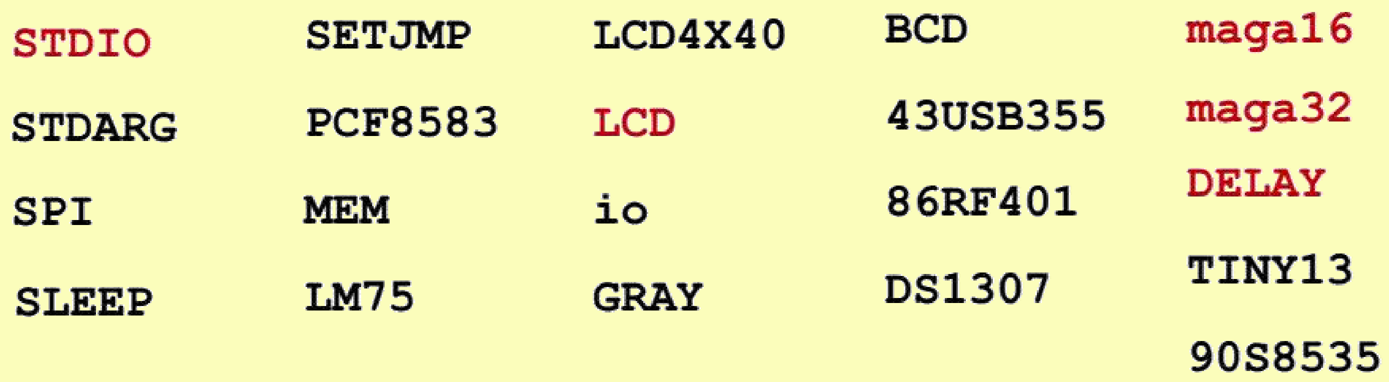
Example

pow(), sqrt(), strcmp(), strcpy(), etc.

Functions in C [AVR]

Types of Function

Library Functions:



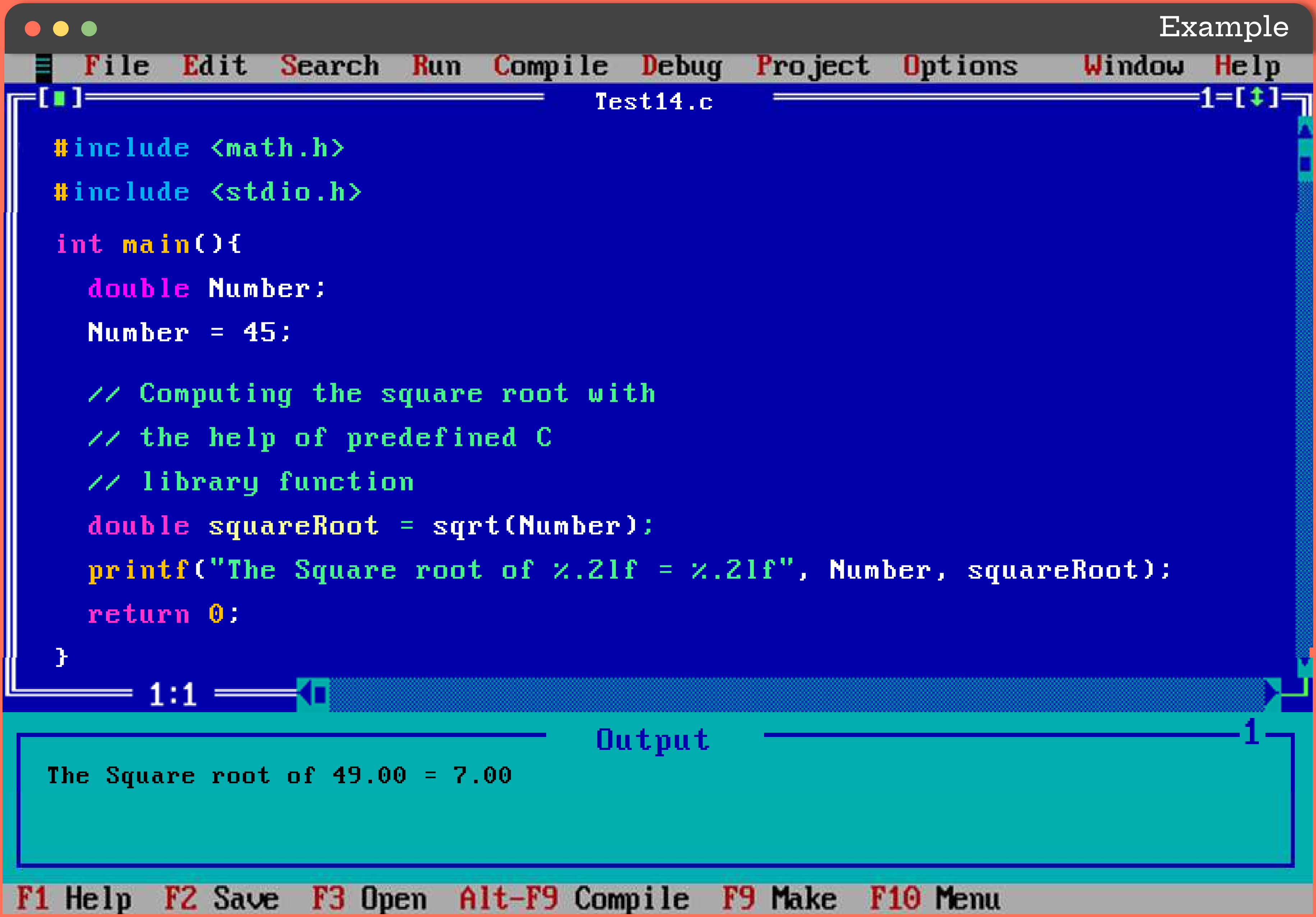
STDIO
STDARG
SPI
SLEEP
SETJMP
PCF8583
MEM
LM75
LCD4X40
LCD
io
GRAY
BCD
43USB355
86RF401
DS1307
maga16
maga32
DELAY
TINY13
90S8535

- ➢ The libraries **string**, **stdlib**, **ctype**, and **math** are part of the standard libraries in the C language, and the CodeVisionAVR compiler also includes these libraries.
- ➢ Library functions are located in the **INC** and **LIB** folders within the **software installation path**. These functions need to be declared at the beginning of program before can use them.

Syntax

```
#include <library_name.h>
```

Example


File Edit Search Run Compile Debug Project Options Window Help
Test14.c
1:1
Output
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```c
#include <math.h>
#include <stdio.h>
int main(){
  double Number;
  Number = 45;
  // Computing the square root with
  // the help of predefined C
  // library function
  double squareRoot = sqrt(Number);
  printf("The Square root of %.2lf = %.2lf", Number, squareRoot);
  return 0;
}
```

```
The Square root of 49.00 = 7.00
```

Functions in C [AVR]

Types of Function

There are two types of functions in C:

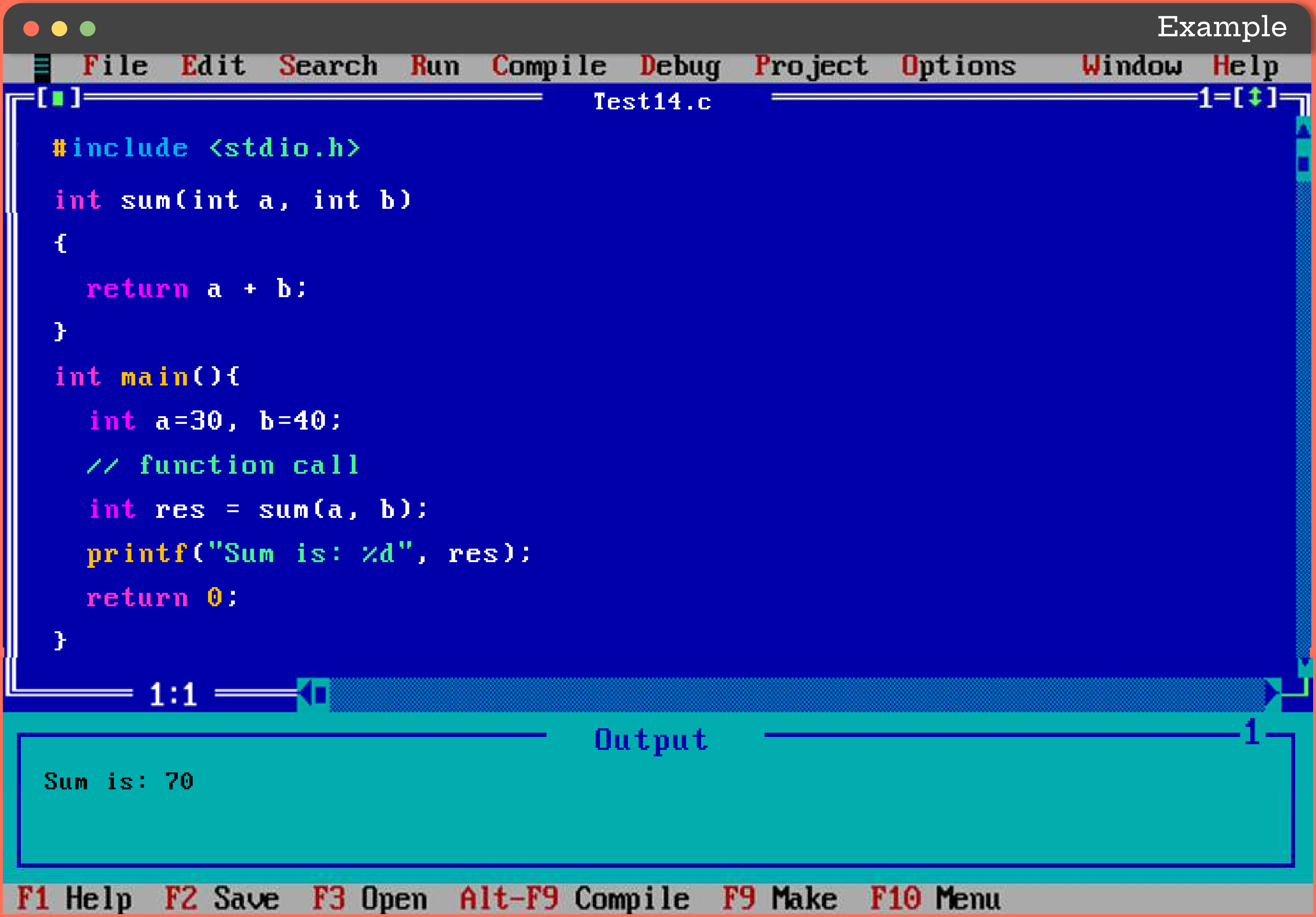
1.Library Functions
2.User Defined Functions

User Defined Functions:

✓ Functions that the programmer creates are known as User-Defined functions or “tailor-made functions”.

✓ Whenever we write a function that is case-specific and is not defined in any header file, we need to declare and define our own functions according to the syntax.

Example



```c
#include <stdio.h>
int sum(int a, int b)
{
  return a + b;
}
int main(){
  int a=30, b=40;
  // function call
  int res = sum(a, b);
  printf("Sum is: %d", res);
  return 0;
}
```

```
Sum is: 70
```

Functions in C [AVR]

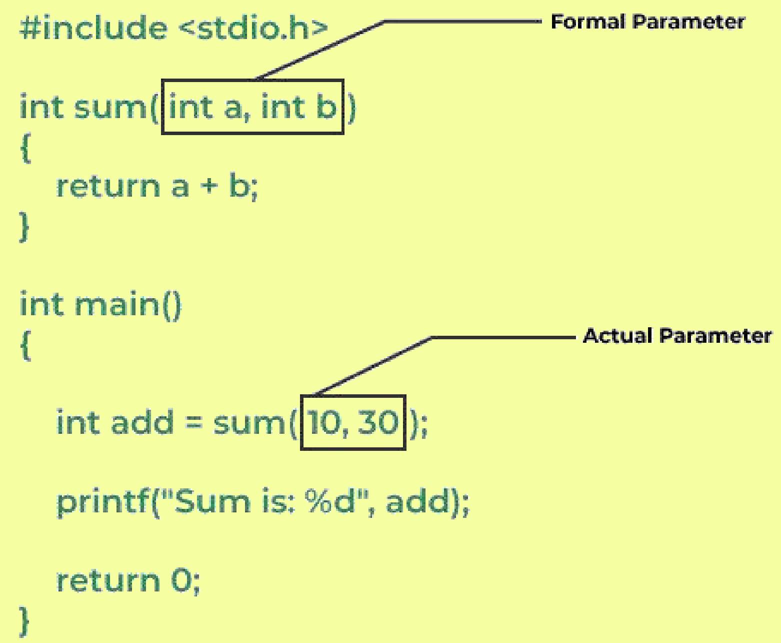
Passing Parameters to Functions

✓ The data passed when the function is being invoked is known as the Actual parameters.

✓ In the below program, 10 and 30 are known as actual parameters.

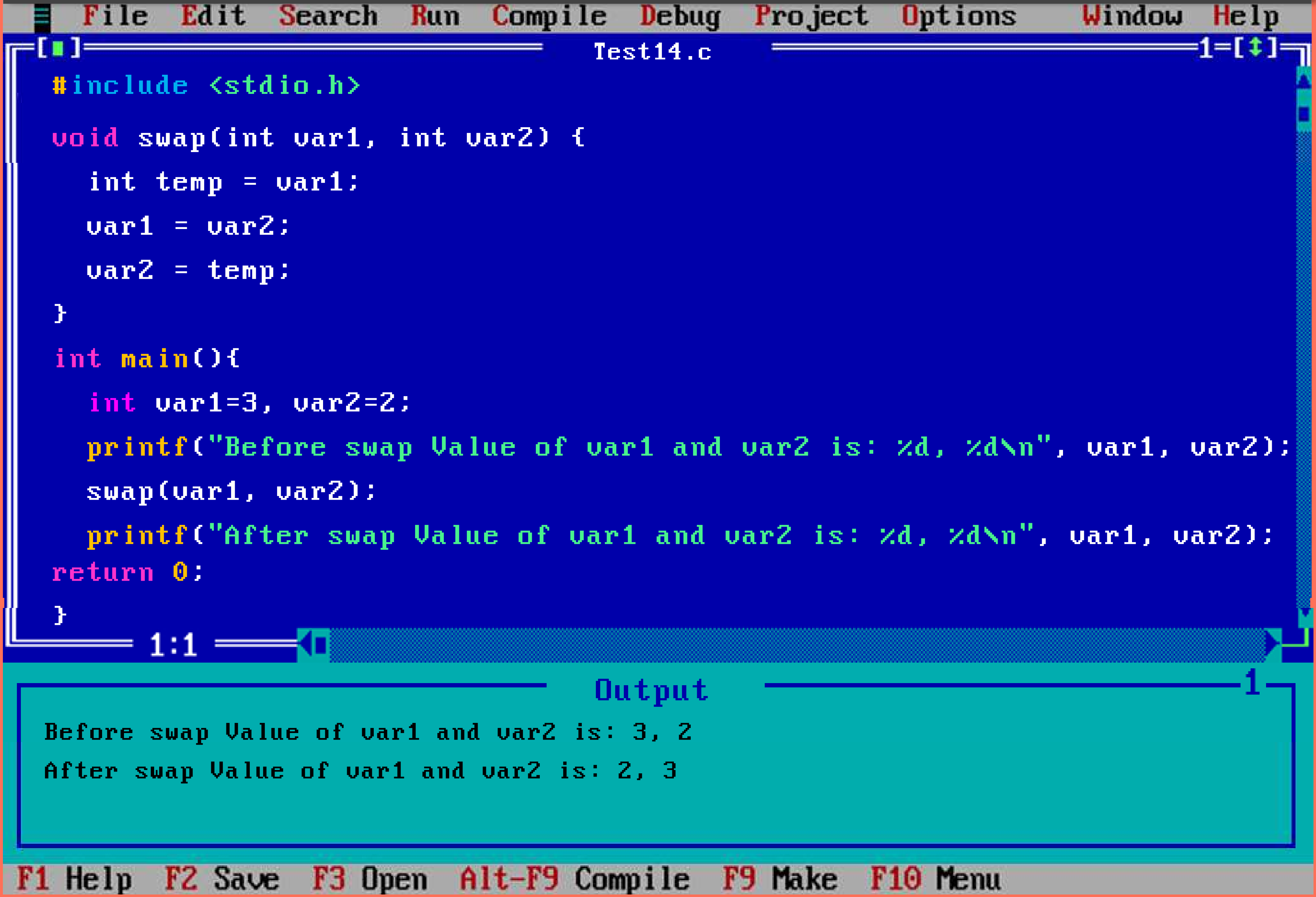
✓ Formal Parameters are the variable and the data type as mentioned in the function declaration.

✓ In the below program, a and b are known as formal parameters.



```
#include <stdio.h>

int sum(int a, int b)
{
   return a + b;
}

int main()
{

   int add = sum(10, 30);

   printf("Sum is: %d", add);

   return 0;
}
```

Example



File Edit Search Run Compile Debug Project Options Window Help
Test14.c
1
1:1
Output
1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

```
#include <stdio.h>
void swap(int var1, int var2) {
   int temp = var1;
   var1 = var2;
   var2 = temp;
}
int main(){
   int var1=3, var2=2;
   printf("Before swap Value of var1 and var2 is: %d, %d\n", var1, var2);
   swap(var1, var2);
   printf("After swap Value of var1 and var2 is: %d, %d\n", var1, var2);
return 0;
}
```

```
Before swap Value of var1 and var2 is: 3, 2
After swap Value of var1 and var2 is: 2, 3
```

Time Delays in C

- You can use for to make time delay

```
void delay(void)
{
volatile unsigned int i;
for(i = 0; i < 42150; i++)
{ }
}
```

If you use for loop

- The clock frequency can change your delay duration !
- The compiler has direct effect on delay duration!

Time Delays in C

- You can use predefined functions of compilers to make time delays

In Atmel Studio:

First you should include:

```c
#define F_CPU 8000000UL
#include <util/delay.h>
```

and then you can use

```c
_delay_us(200);   //200 microseconds
_delay_ms(100);   //100 milliseconds
```

- It is compiler dependant

I/O programming in C

Byte size I/O programming in C

```
 DDRB = 0xFF;
while (1) {
  PORTB = 0xFF;
 _delay_ms(500);
  PORTB = 0x55;
 _delay_ms(500);
}
```

Different compilers have different syntax for bit manipulations!

It is better to use
Bit-wise logical operators

Memory Types In AVR

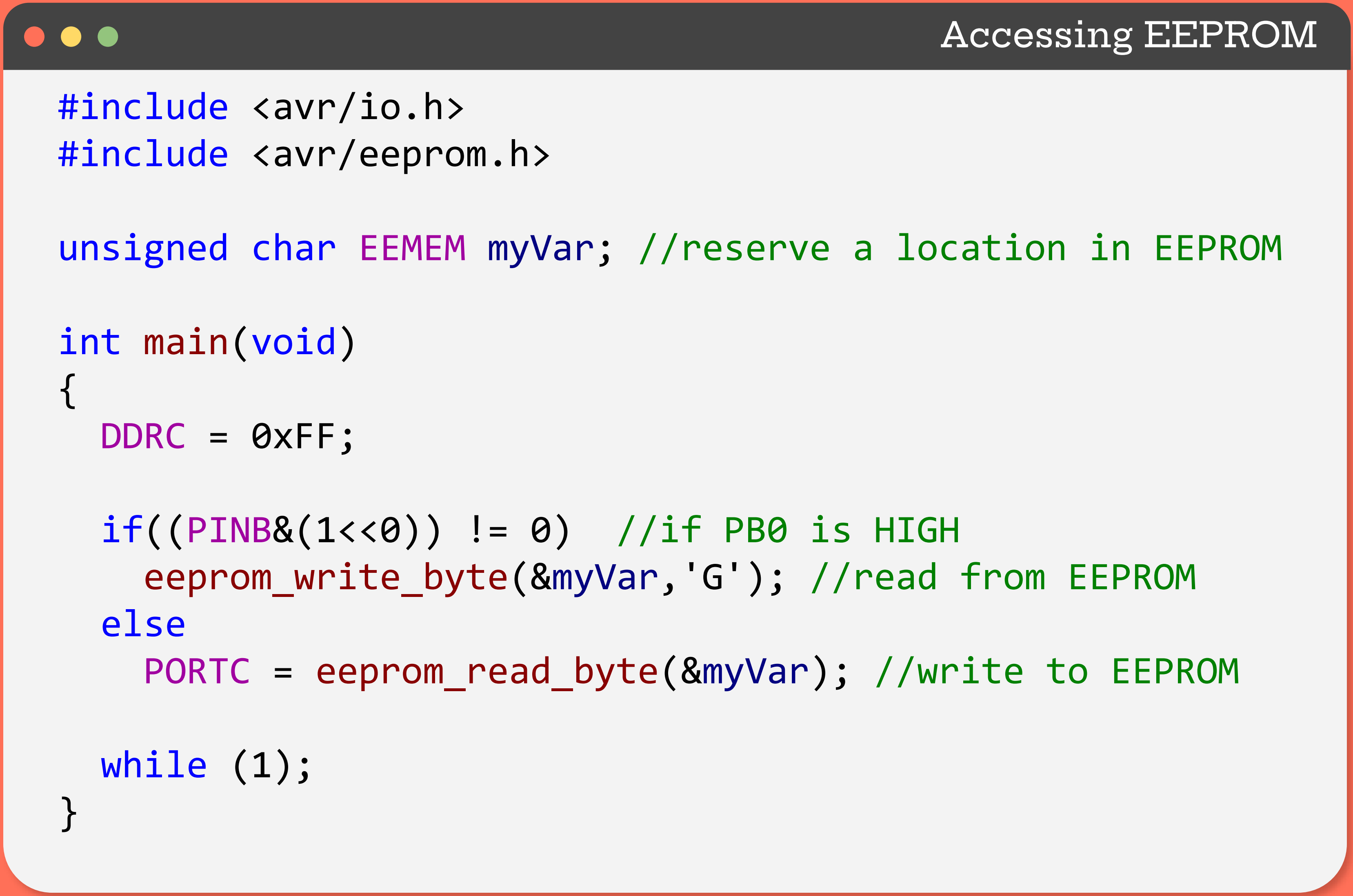
- Flash Memory
  - Not deleted when power is off
  - Big in size
  - Suitable for codes, tables and fixed data
- EEPROM
  - Not deleted when power is off
  - Not very big in size
  - Suitable for small data that may be modified but should not be lost when power is off
- RAM
  - deleted when power is off
  - Suitable for storing the data we want to manipulate because we have fast access to read or modify them.

Accessing Flash

- `#include <avr/pgmspace.h>`
- const unsigned char PROGMEM lookup[] ={5,6,7,4};
  ourData = pgm_read_byte(&lookup[i]);

Example

```c
#include <avr/pgmspace.h>

const unsigned char PROGMEM lookup[] ={5,6,7,4};

int main(void)
{
  unsigned char a;
  a = pgm_read_byte(&lookup[i]);

  while (1);
}
```



Accessing EEPROM

```
#include <avr/io.h>
#include <avr/eeprom.h>

unsigned char EEMEM myVar; //reserve a location in EEPROM

int main(void)
{
  DDRC = 0xFF;

  if((PINB&(1<<0)) != 0)  //if PB0 is HIGH
    eeprom_write_byte(&myVar,'G'); //read from EEPROM
  else
    PORTC = eeprom_read_byte(&myVar); //write to EEPROM

  while (1);
}
```

Thanks!

Do you have any questions?

razeghizade@gmail.com

+98 922 483 1345

www.pudica.ir

CREADITS: This presentation was created by M.Razeghizadeh
Please keep this slide for attribution