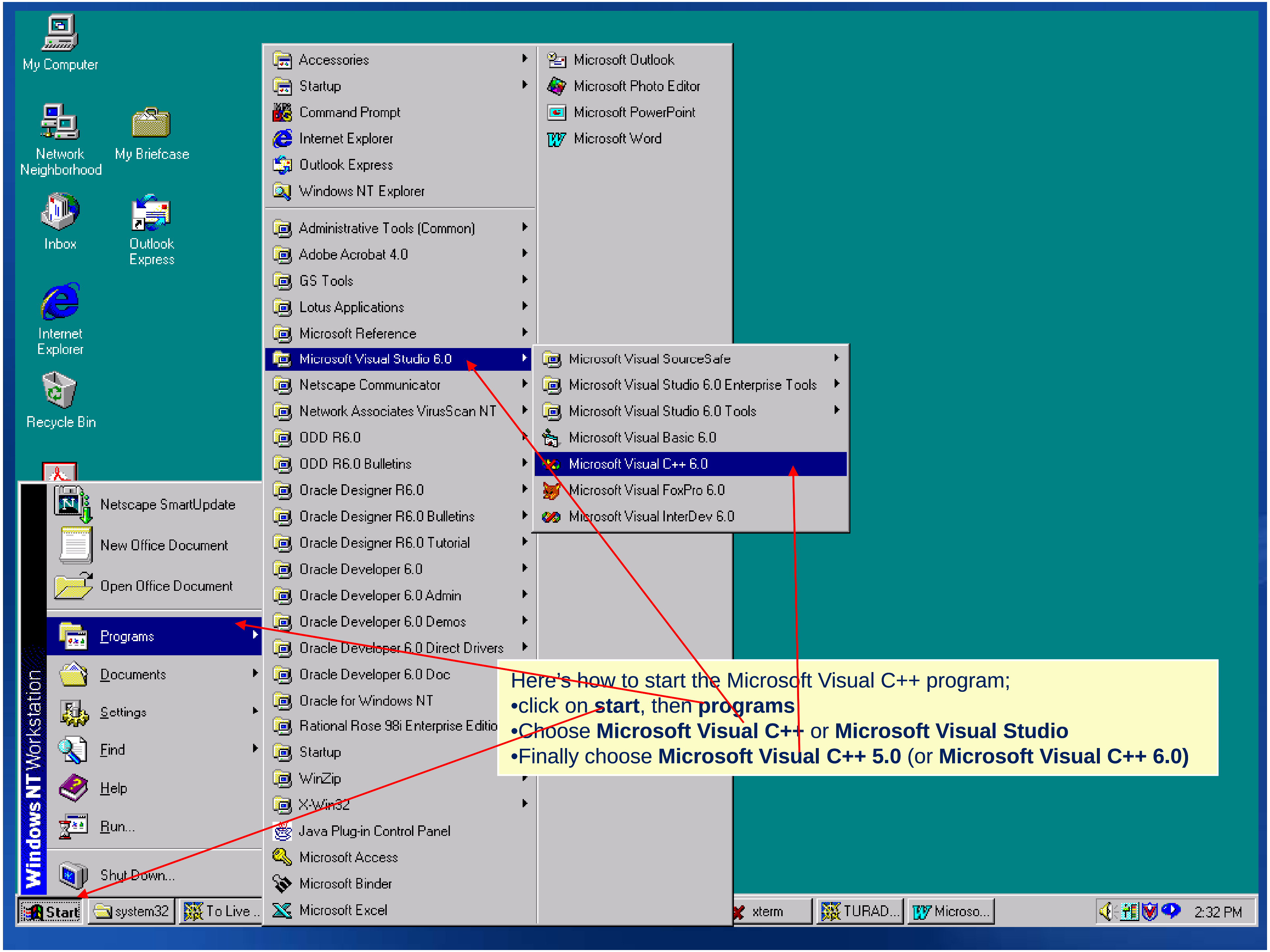
Chapter 2
Introducing C

Chapter Outline

- Operator: =
- Functions: main(), printf()
- Putting together a simple C program
- Creating integer-valued variables, assigning them values, and displaying those values onscreen
- The newline character
- How to include comments in your programs, create programs containing more than one function, and find program errors
- What keywords are

Visual C++ 6.0 Tutorial

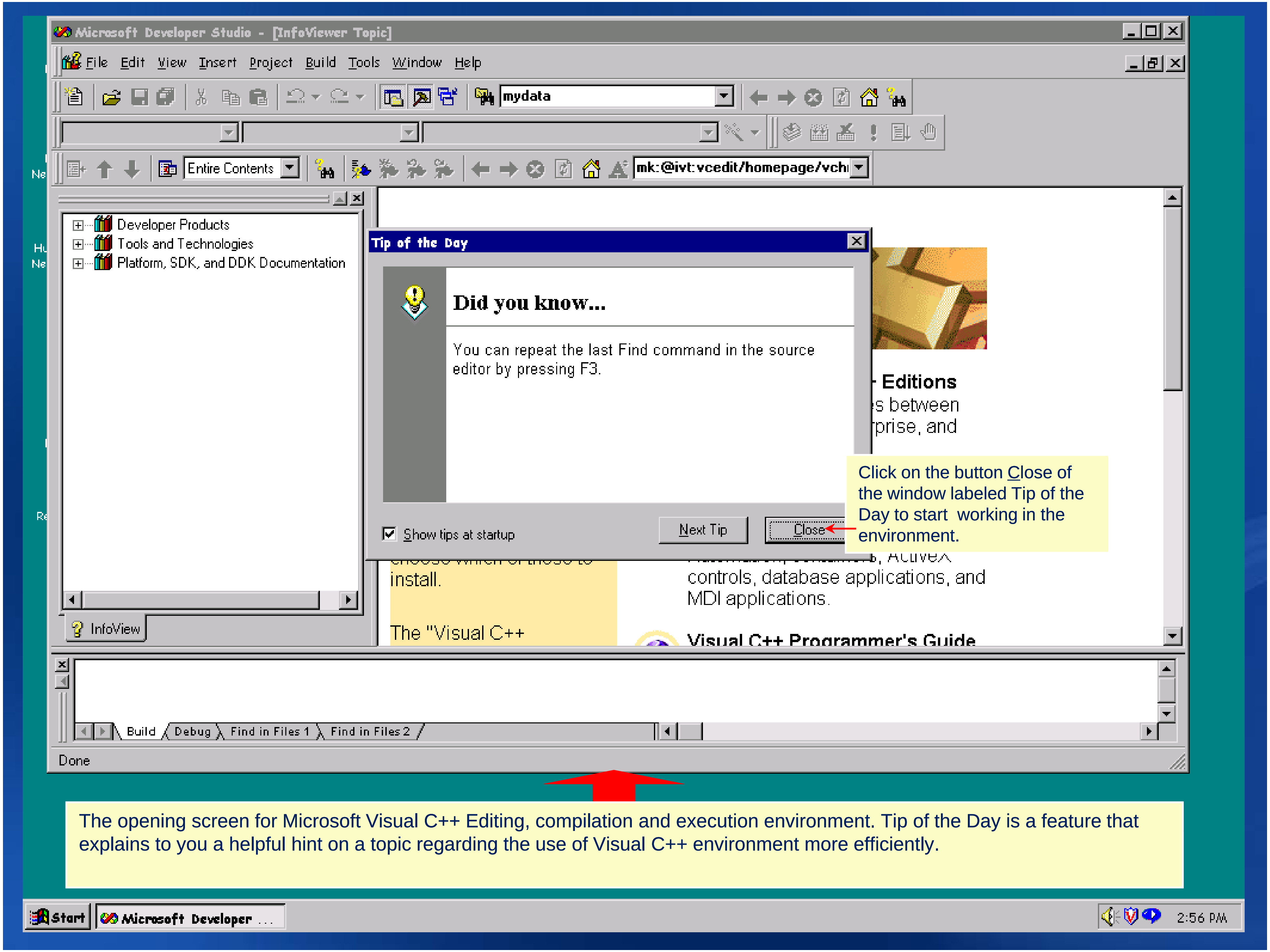


Here's how to start the Microsoft Visual C++ program;

•click on **start**, then **programs**

•Choose **Microsoft Visual C++** or **Microsoft Visual Studio**

•Finally choose **Microsoft Visual C++ 5.0** (or **Microsoft Visual C++ 6.0)**



Did you know...

You can repeat the last Find command in the source editor by pressing F3.

☑ Show tips at startup

Click on the button Close of the window labeled Tip of the Day to start working in the environment.

The opening screen for Microsoft Visual C++ Editing, compilation and execution environment. Tip of the Day is a feature that explains to you a helpful hint on a topic regarding the use of Visual C++ environment more efficiently.

Workspaces
Projects
C Program Source files

In Visual C++ The programs you develop are organized as follows;

- C source file

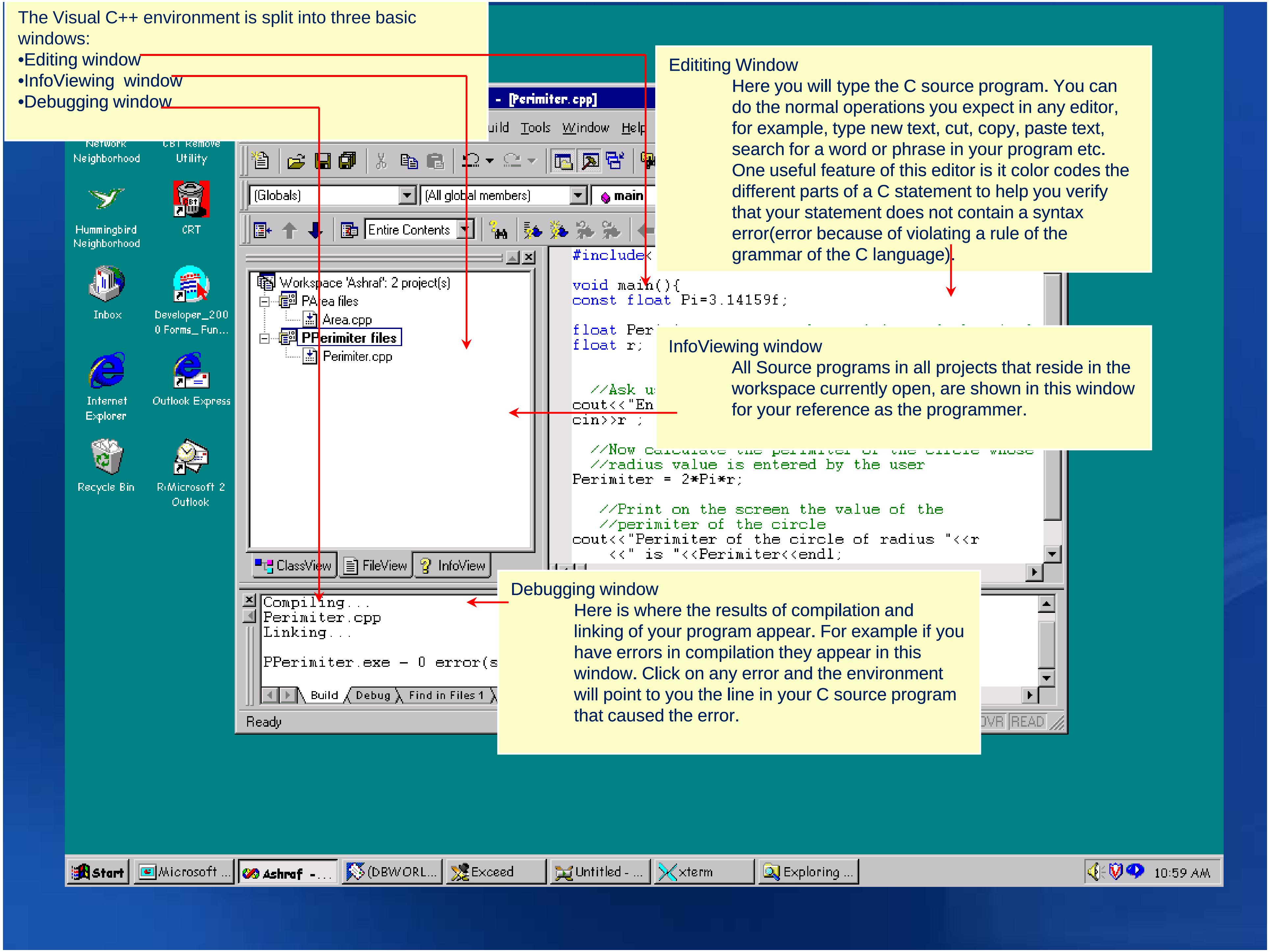
  A C program that implements a certain algorithm to solve a problem

- Project
  - Sometimes in big programs one has to distribute the C source statements over multiple files, not only one.
  - In these cases all the files that logically make up one program must be collected together.
  - A project is the folder in visual C where all pieces of the same program are stored.

- Workspaces
  - In big big software development projects there may be a need to develop multiple big programs (i.e. multiple projects), for example, In a financial software package , you find Payroll, Accounts payable, Accounts receivable programs.
  - A workspace is the way visual C allows us to collect related projects

The Visual C++ environment is split into three basic windows:

- Editing window
- InfoViewing window
- Debugging window



```
#include<
void main(){
const float Pi=3.14159f;

float Per
float r;


  //Ask u
cout<<"En
cin>>r ;

  //Now calculate the perimiter of the circle whose
  //radius value is entered by the user
Perimiter = 2*Pi*r;

   //Print on the screen the value of the
   //perimiter of the circle
cout<<"Perimiter of the circle of radius "<<r
    <<" is "<<Perimiter<<endl;
```

```
Compiling...
Perimiter.cpp
Linking...

PPerimiter.exe - 0 error(s
```

Edititing Window

Here you will type the C source program. You can do the normal operations you expect in any editor, for example, type new text, cut, copy, paste text, search for a word or phrase in your program etc. One useful feature of this editor is it color codes the different parts of a C statement to help you verify that your statement does not contain a syntax error(error because of violating a rule of the grammar of the C language).

InfoViewing window

All Source programs in all projects that reside in the workspace currently open, are shown in this window for your reference as the programmer.

Debugging window

Here is where the results of compilation and linking of your program appear. For example if you have errors in compilation they appear in this window. Click on any error and the environment will point to you the line in your C source program that caused the error.

Using Microsoft Visual C++ Environment
Scenario I

The following slide illustrate a scenario for creating a new C source program from scratch, compiling it, and executing it. The steps are summarized as follows:

- Create New Source file

  This step allows a new C source file that will contain your program to be opened

- Editing & storing a C source program inside the project workspace

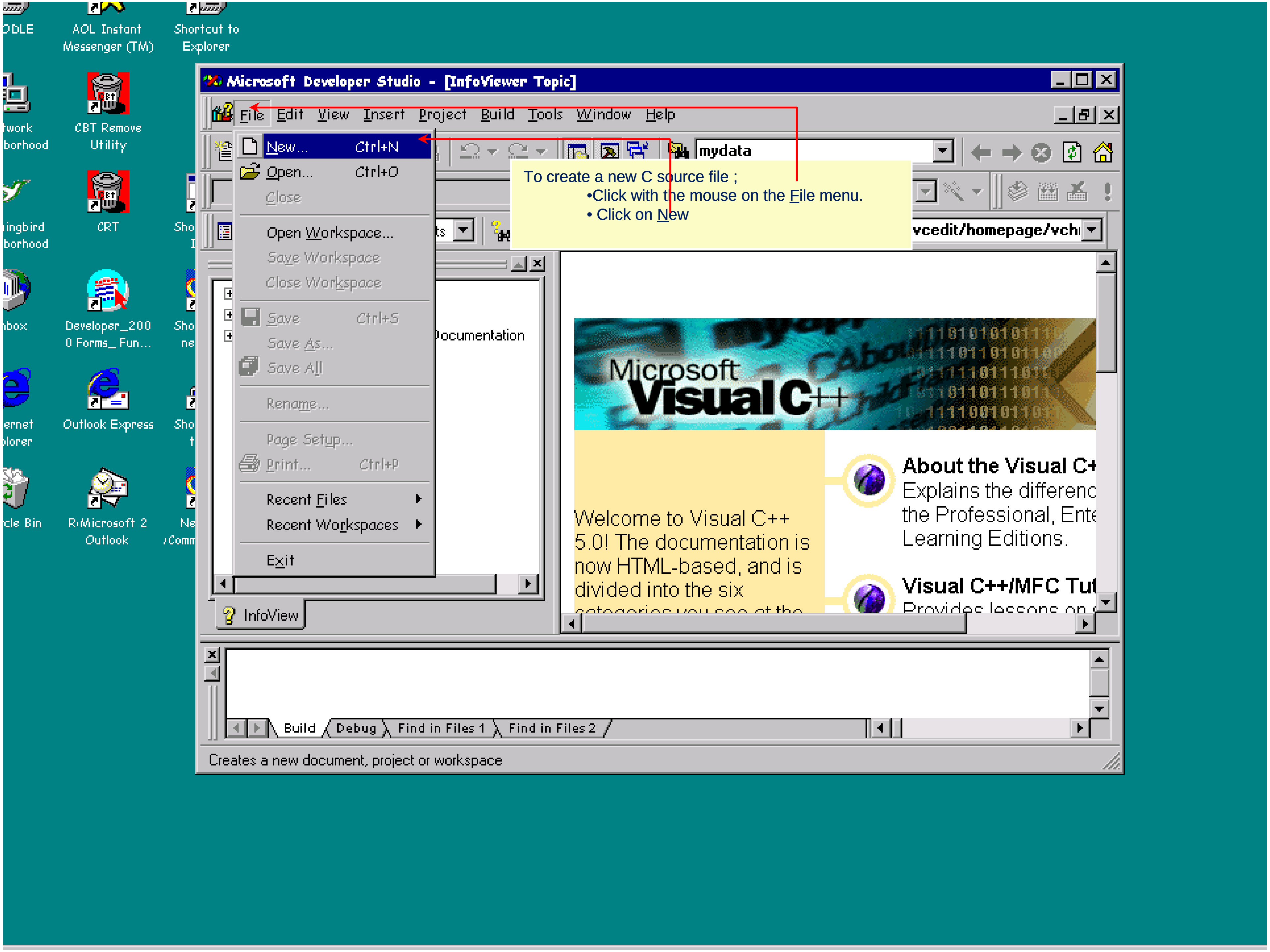
  In the editing window we will type the statements of a simple but complete C source program.

- Compiling & linking the C source program to produce the executable machine language program

  The source program written in the C language is translated into a program that does the same exact thing but written in the machine language of the computer you are using. The later program is called the executable program since it can be run directly on the machine.(For example, if your machine has a Pentium processor translation will produce a program written in the Pentium processor's instruction set)

- Executing the executable program

  Basically, Visual C++ environment asks the Operating system to load the machine language instructions of your executable program and have them executed.



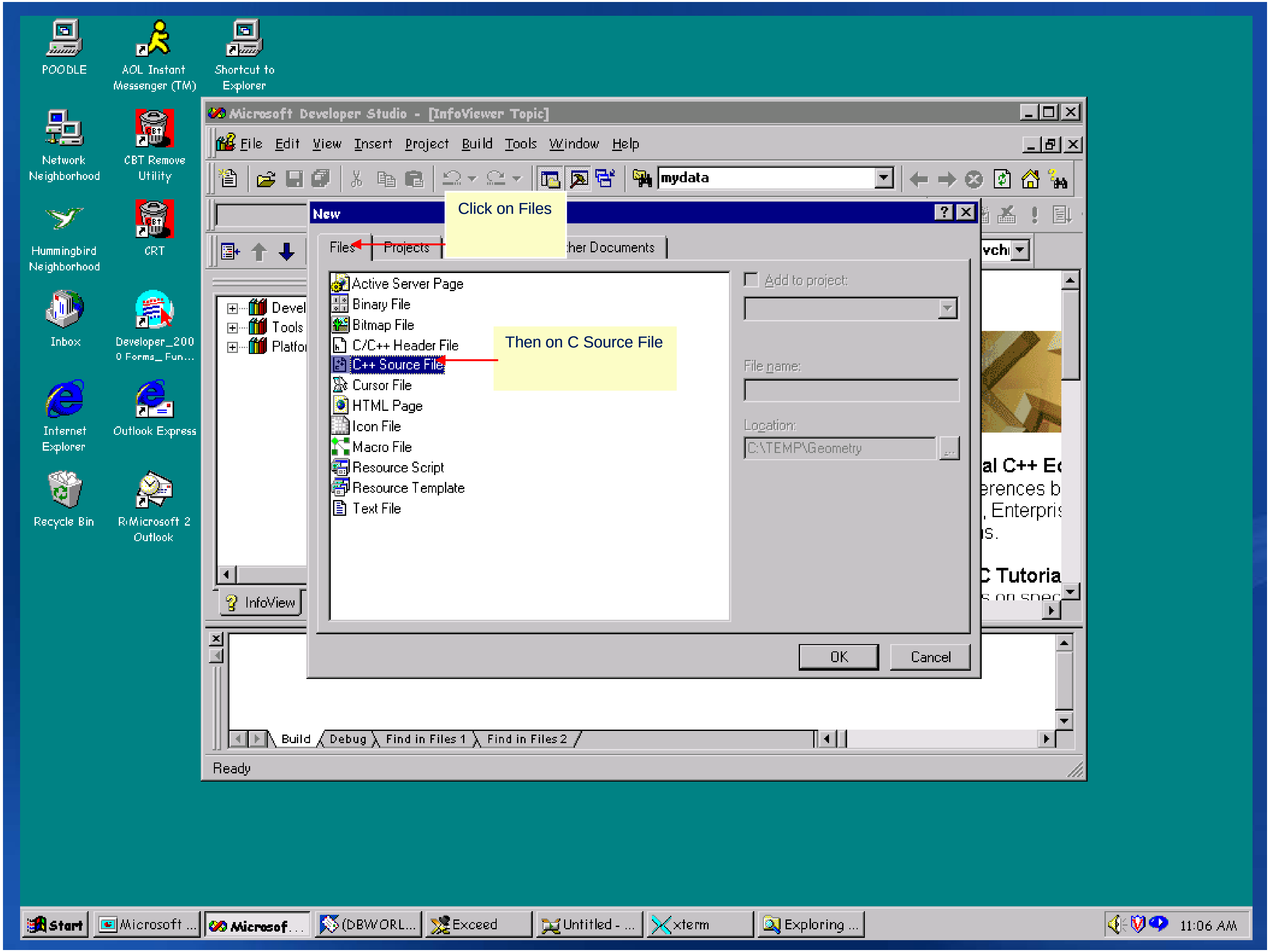
To create a new C source file ;

- Click with the mouse on the File menu.
- Click on New

Welcome to Visual C++ 5.0! The documentation is now HTML-based, and is divided into the six

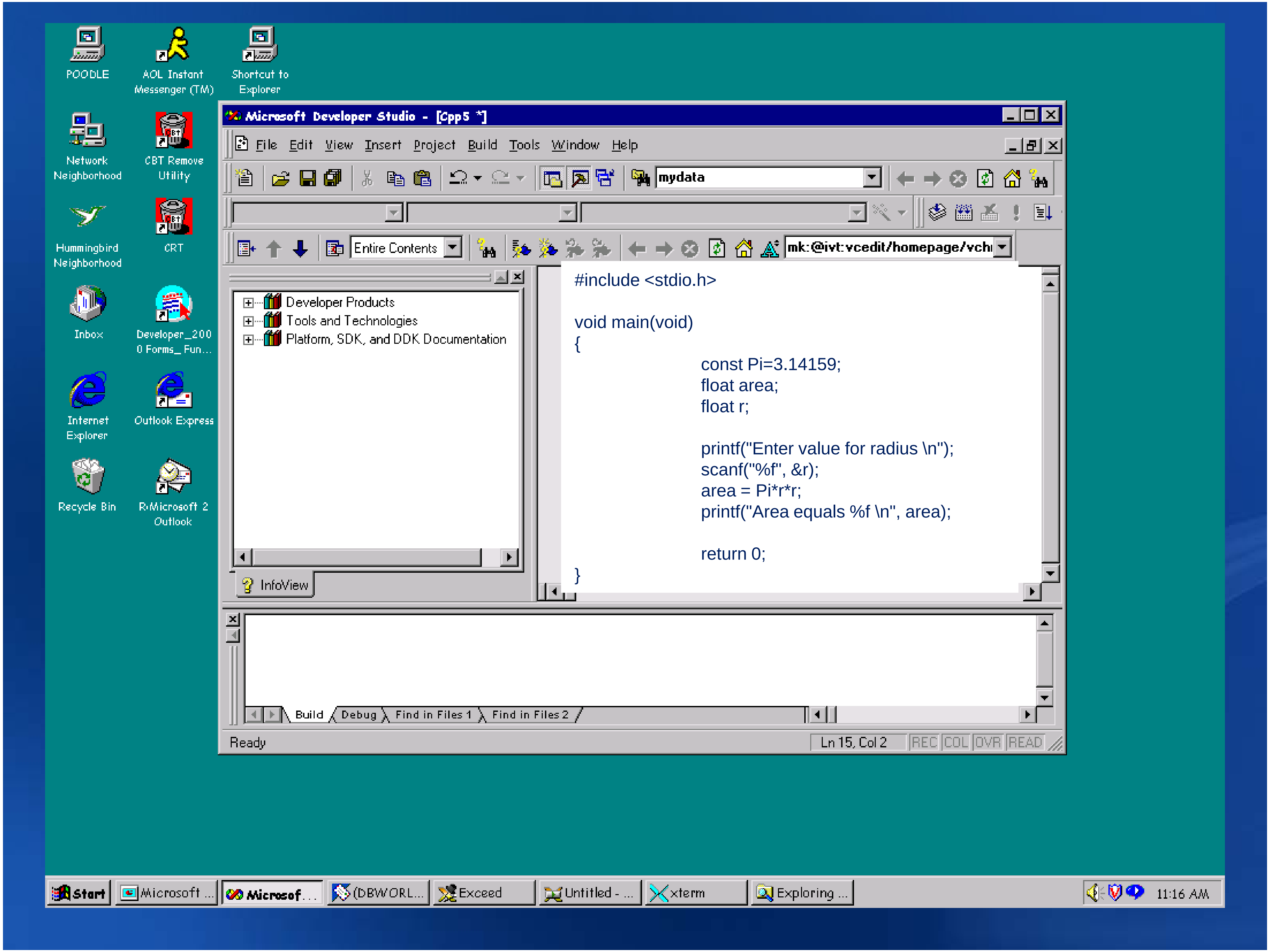
Explains the differenc the Professional, Ente Learning Editions.

Creates a new document, project or workspace

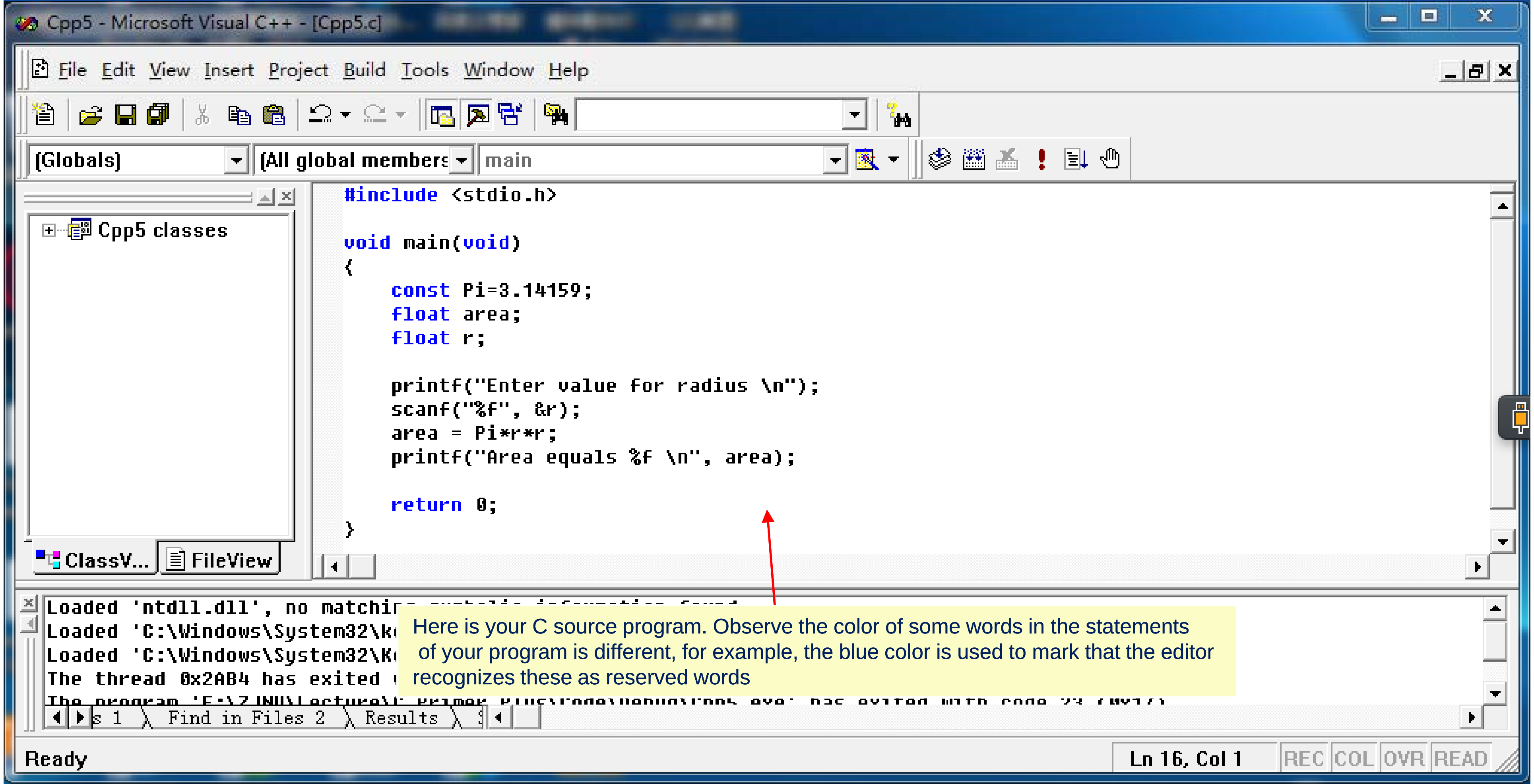


Click on Files

Then on C Source File



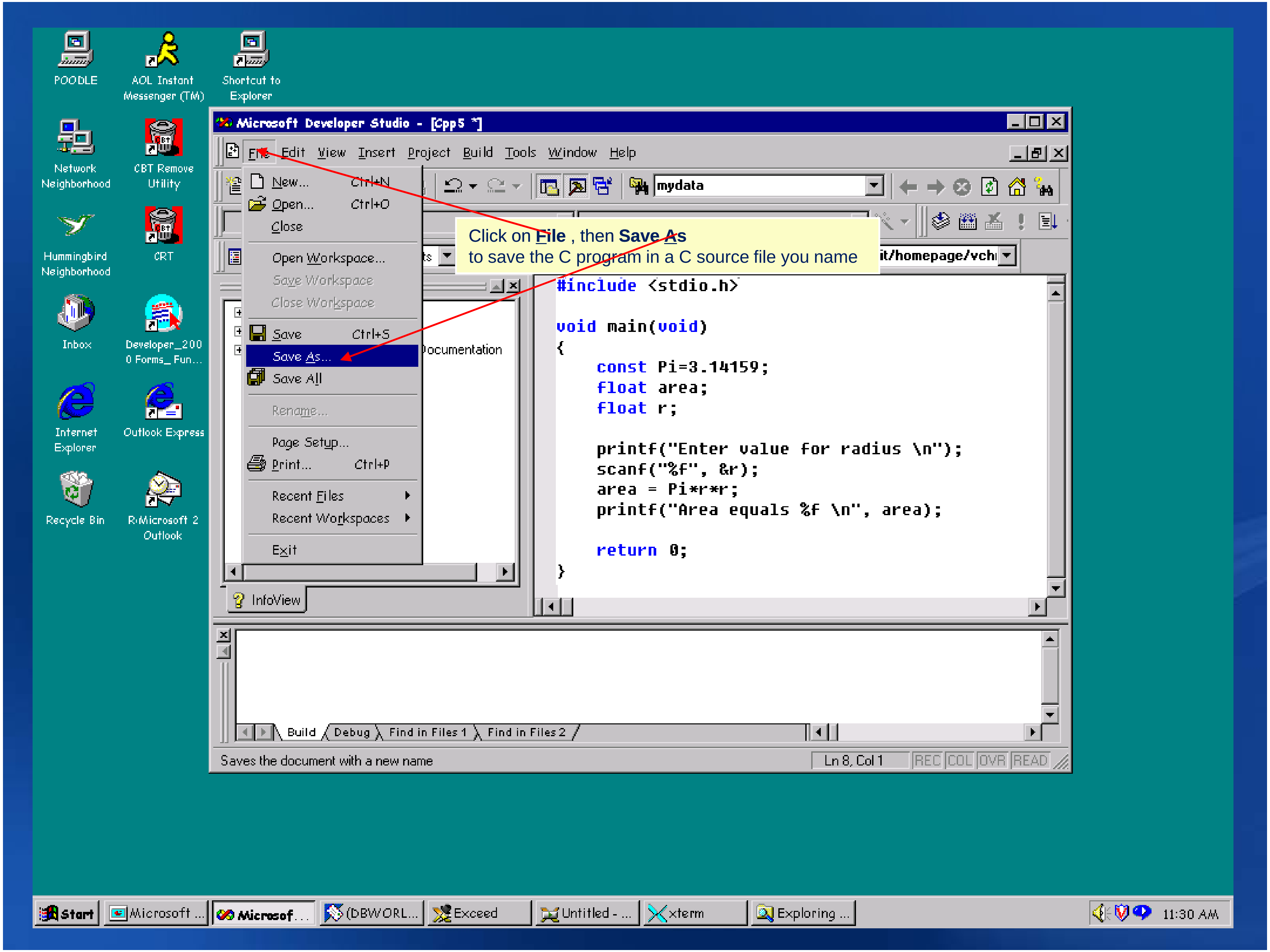
```
#include <stdio.h>

void main(void)
{
                const Pi=3.14159;
                float area;
                float r;

                printf("Enter value for radius \n");
                scanf("%f", &r);
                area = Pi*r*r;
                printf("Area equals %f \n", area);

                return 0;
}
```



```
#include <stdio.h>

void main(void)
{
    const Pi=3.14159;
    float area;
    float r;

    printf("Enter value for radius \n");
    scanf("%f", &r);
    area = Pi*r*r;
    printf("Area equals %f \n", area);

    return 0;
}
```

```
Loaded 'ntdll.dll', no matching symbolic information found.
Loaded 'C:\Windows\System32\k
Loaded 'C:\Windows\System32\K
The thread 0x2AB4 has exited
The program 'F:\ZJNU\Lecture\C primer plus\code\Debug\Cpp5.exe' has exited with code 23 (0x17).
```

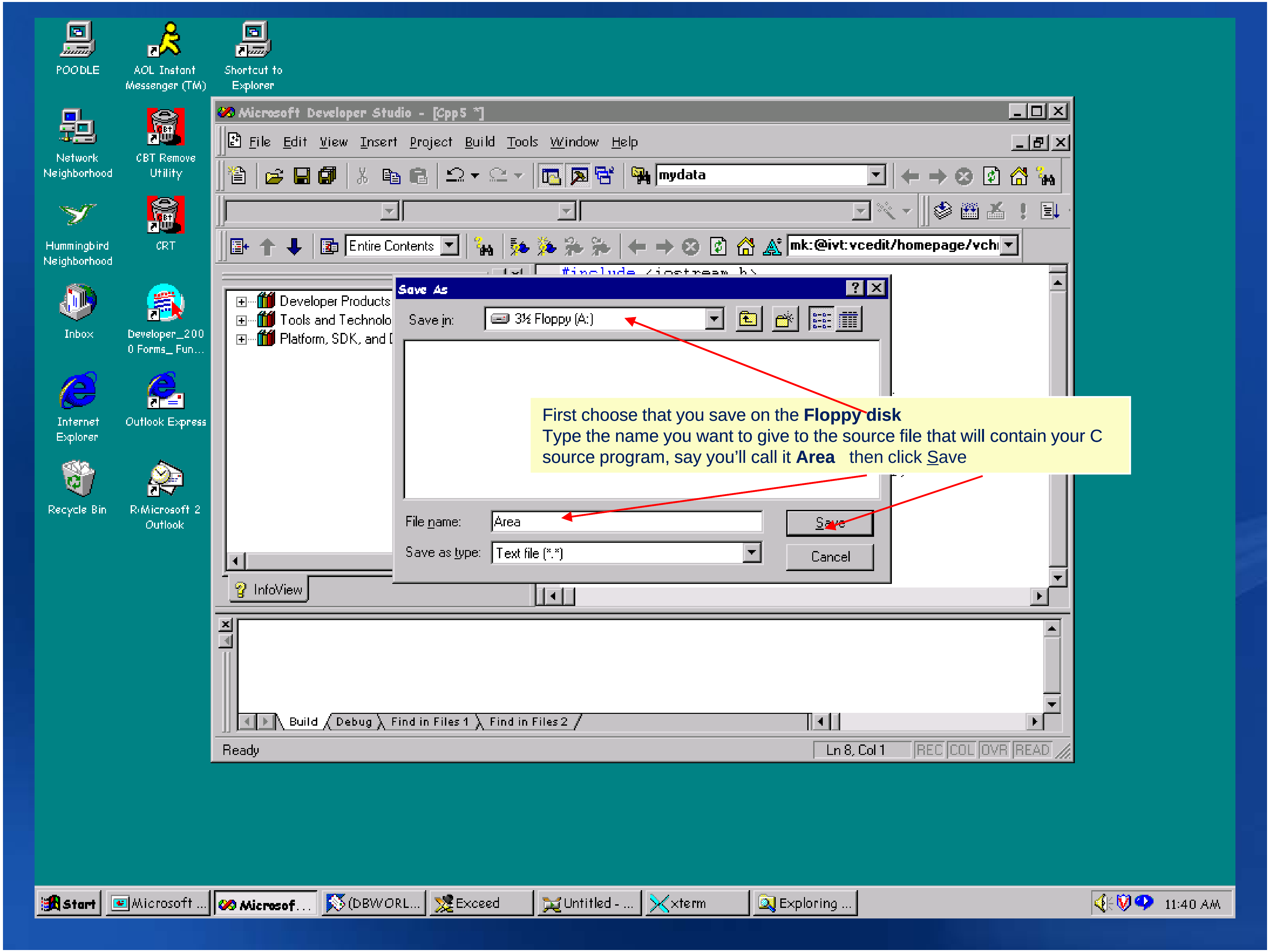
Here is your C source program. Observe the color of some words in the statements of your program is different, for example, the blue color is used to mark that the editor recognizes these as reserved words



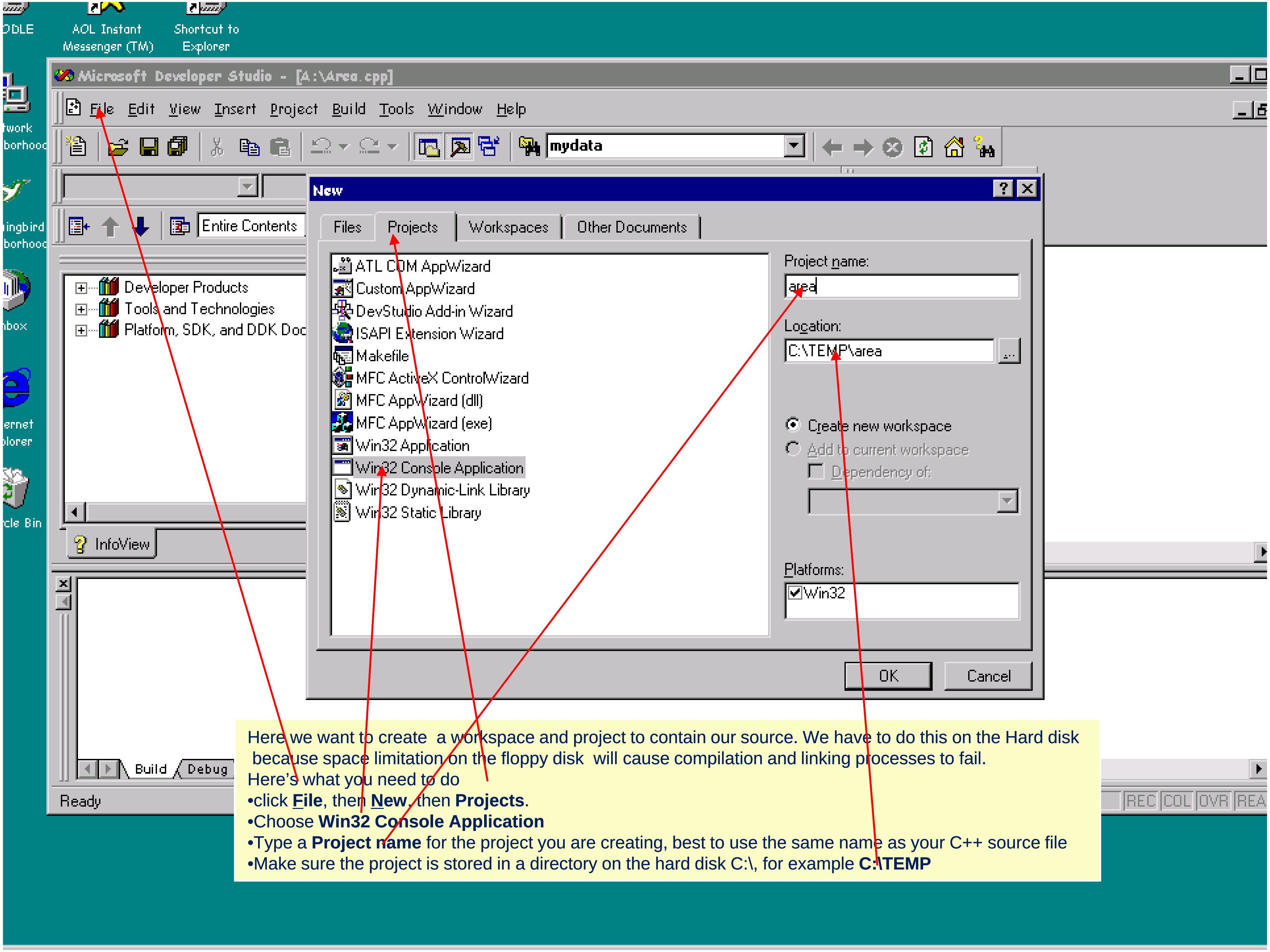
Click on **File** , then **Save As**
to save the C program in a C source file you name

```
#include <stdio.h>

void main(void)
{
    const Pi=3.14159;
    float area;
    float r;

    printf("Enter value for radius \n");
    scanf("%f", &r);
    area = Pi*r*r;
    printf("Area equals %f \n", area);

    return 0;
}
```

Saves the document with a new name



First choose that you save on the **Floppy disk**
Type the name you want to give to the source file that will contain your C source program, say you'll call it **Area** then click Save



Project name: area

Location: C:\TEMP\area

● Create new workspace

○ Add to current workspace

□ Dependency of:

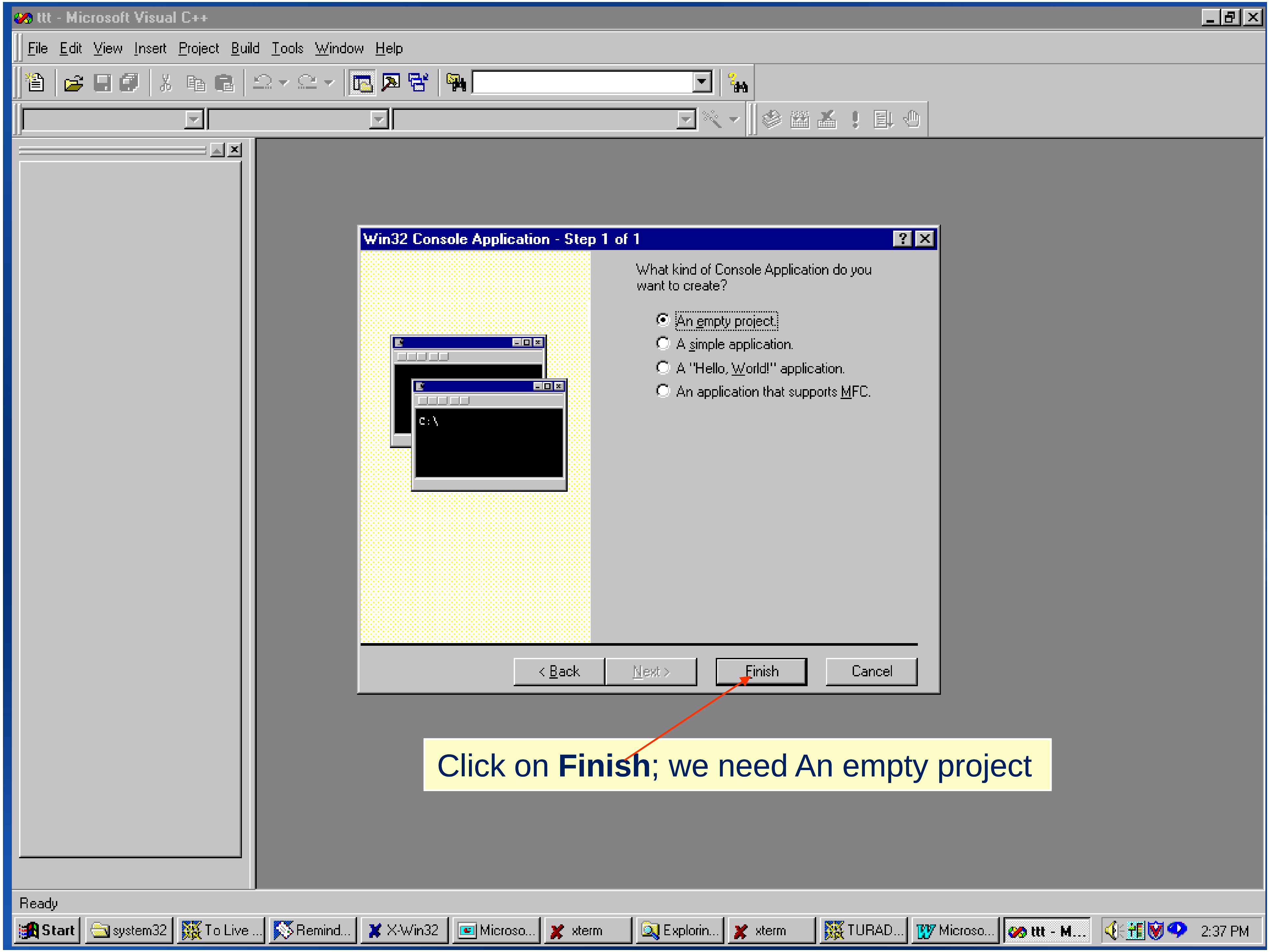
Platforms:

☑ Win32

Here we want to create a workspace and project to contain our source. We have to do this on the Hard disk because space limitation on the floppy disk will cause compilation and linking processes to fail.

Here's what you need to do

- click **File**, then **New**, then **Projects**.
- Choose **Win32 Console Application**
- Type a **Project name** for the project you are creating, best to use the same name as your C++ source file
- Make sure the project is stored in a directory on the hard disk C:\, for example **C:\TEMP**

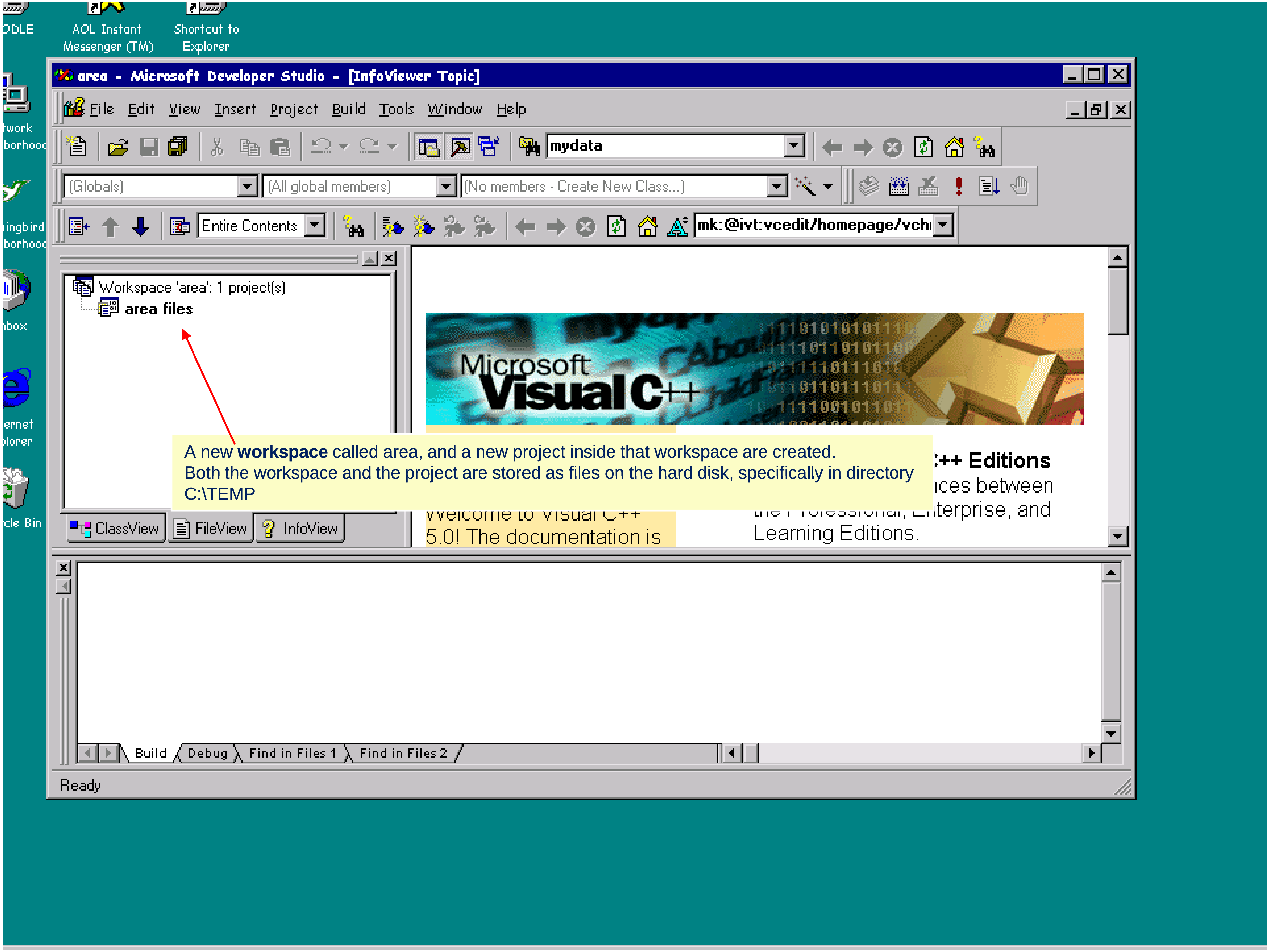


Win32 Console Application - Step 1 of 1

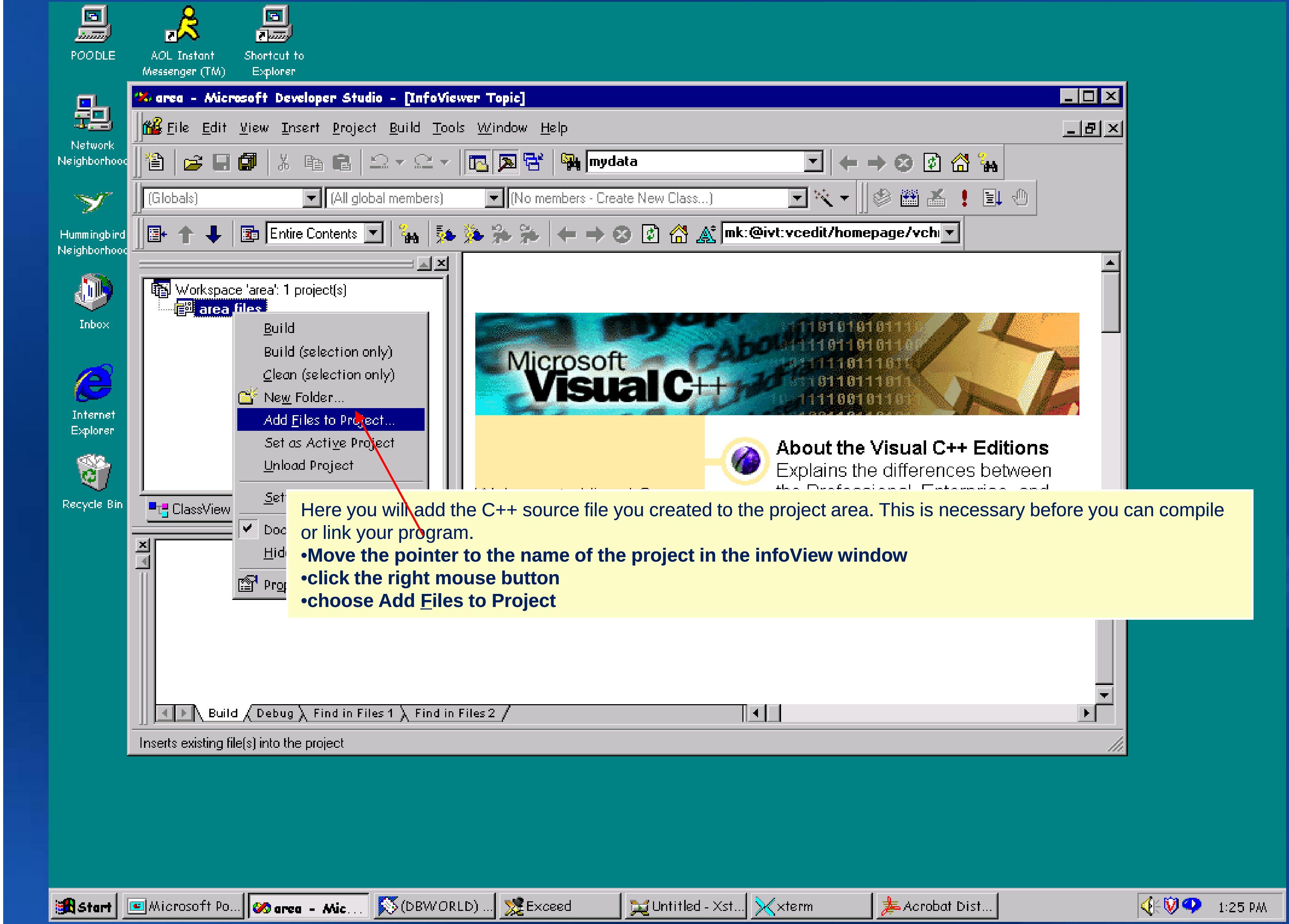
What kind of Console Application do you want to create?

- ● An empty project.
- ○ A simple application.
- ○ A "Hello, World!" application.
- ○ An application that supports MFC.

Click on **Finish**; we need An empty project

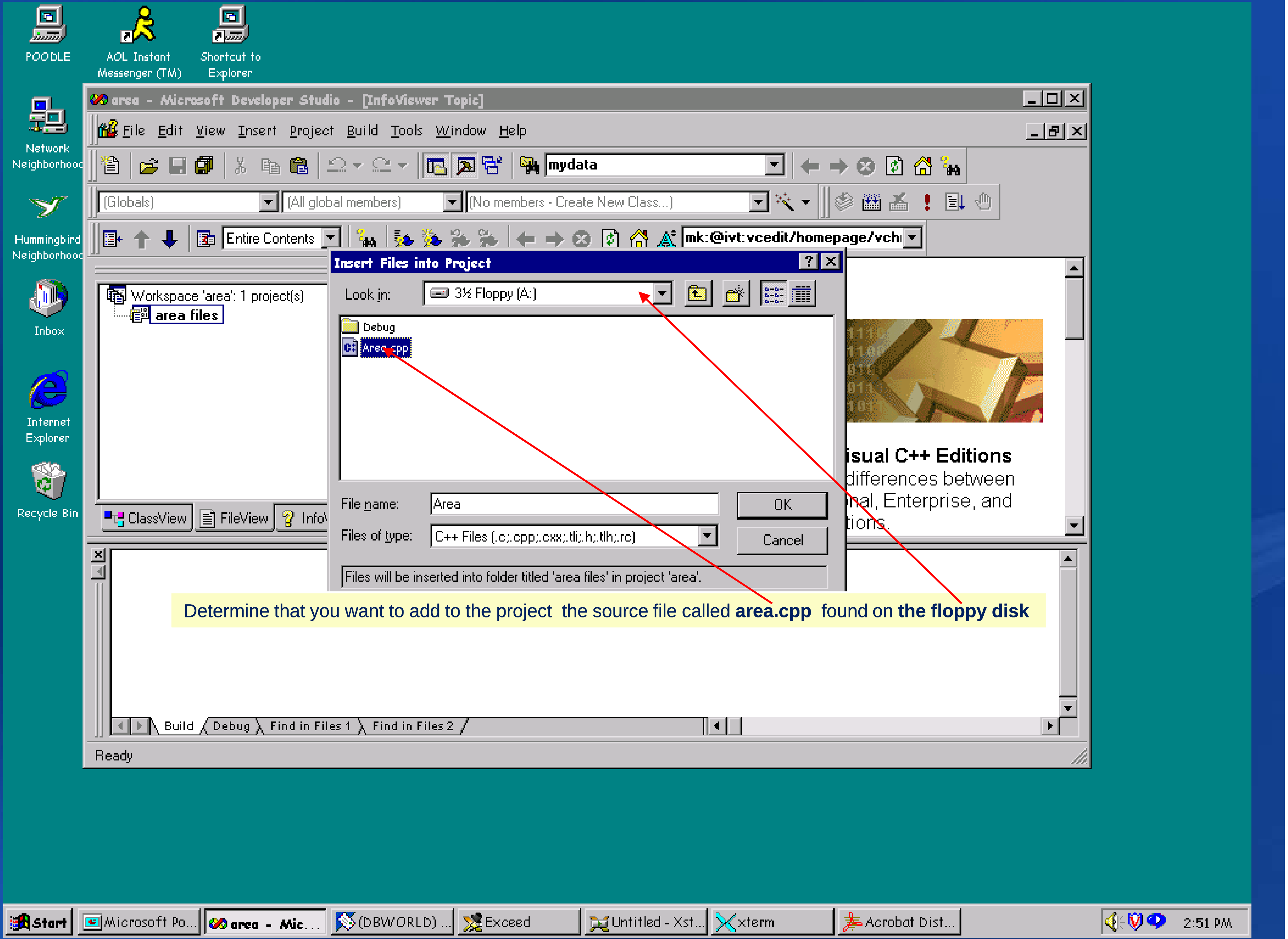

AOL Instant Messenger (TM)
Shortcut to Explorer
area - Microsoft Developer Studio - [InfoViewer Topic]
File Edit View Insert Project Build Tools Window Help
mydata
(Globals)
(All global members)
(No members - Create New Class...)
Entire Contents
mk:@ivt:vcedit/homepage/vchi
Workspace 'area': 1 project(s)
area files
ClassView
FileView
InfoView
Microsoft Visual C++
C++ Editions
ces between
the Professional, Enterprise, and
Learning Editions.
Welcome to Visual C++
5.0! The documentation is
Build
Debug
Find in Files 1
Find in Files 2
Ready

A new **workspace** called area, and a new project inside that workspace are created.
Both the workspace and the project are stored as files on the hard disk, specifically in directory C:\TEMP


area - Microsoft Developer Studio - [InfoViewer Topic]
File Edit View Insert Project Build Tools Window Help
mydata
(Globals)
(All global members)
(No members - Create New Class...)
Entire Contents
mk:@ivt:vcedit/homepage/vchi
Workspace 'area': 1 project(s)
area files
Build
Build (selection only)
Clean (selection only)
New Folder...
Add Files to Project...
Set as Active Project
Unload Project
ClassView
Microsoft Visual C++
About the Visual C++ Editions
Explains the differences between
Build
Debug
Find in Files 1
Find in Files 2
Inserts existing file(s) into the project
Start
1:25 PM

Here you will add the C++ source file you created to the project area. This is necessary before you can compile or link your program.

- **Move the pointer to the name of the project in the infoView window**
- **click the right mouse button**
- **choose Add Files to Project**



Insert Files into Project

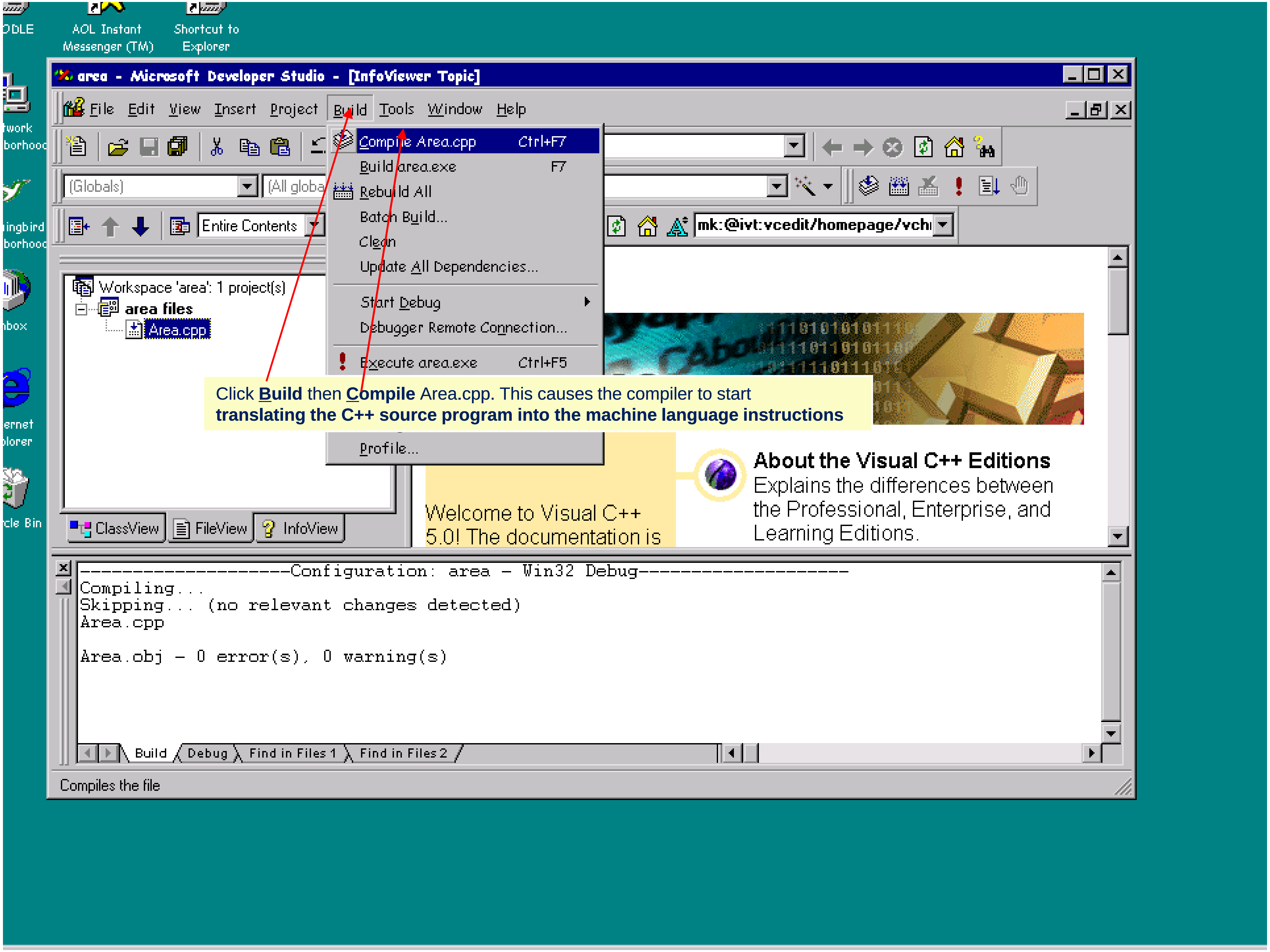
Look in: 3½ Floppy (A:)

File name: Area

Files of type: C++ Files (.c;.cpp;.cxx;.tli;.h;.tlh;.rc)

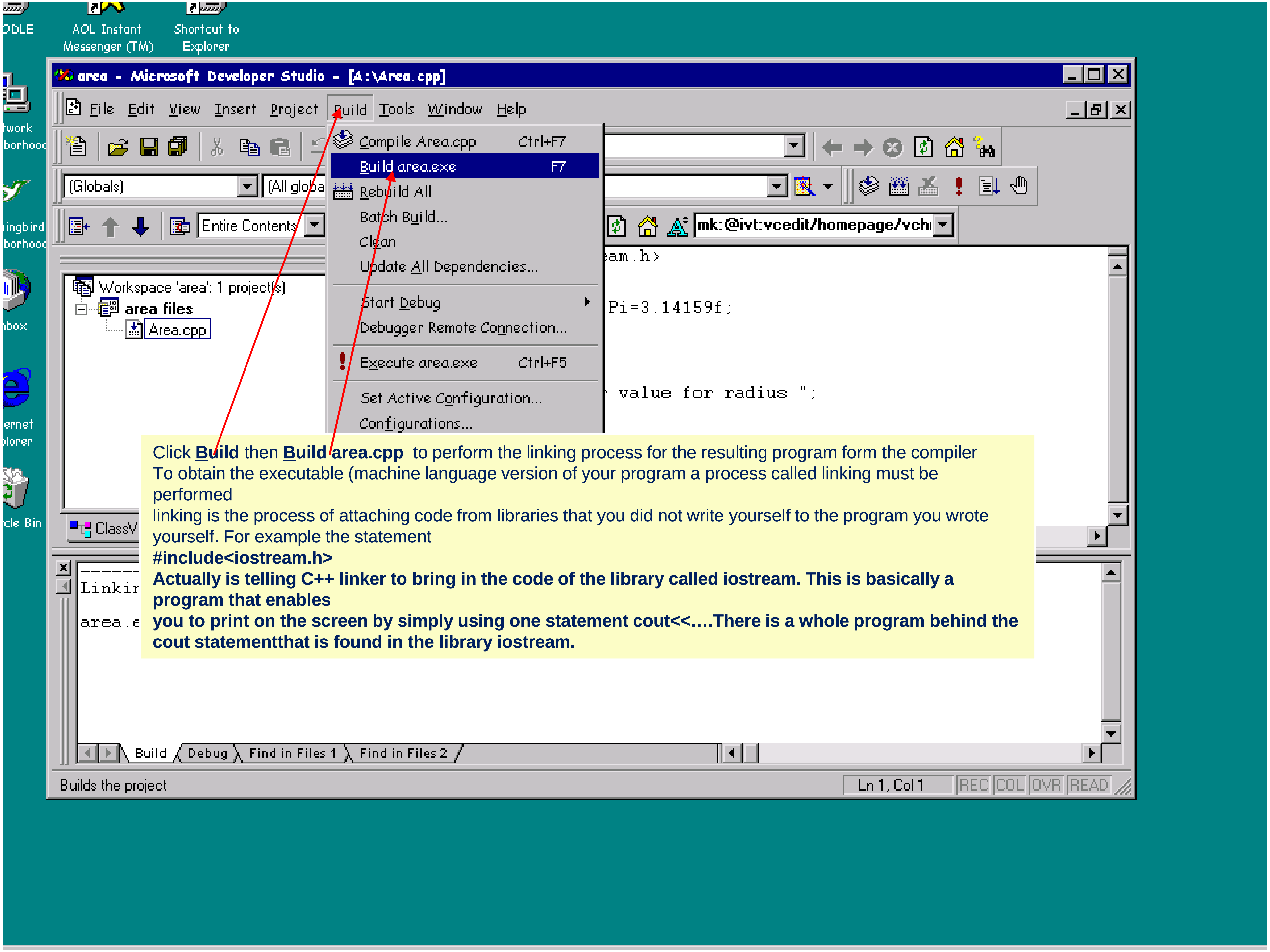
Files will be inserted into folder titled 'area files' in project 'area'.

Determine that you want to add to the project the source file called **area.cpp** found on **the floppy disk**


area - Microsoft Developer Studio - [InfoViewer Topic]
File Edit View Insert Project Build Tools Window Help
Compile Area.cpp Ctrl+F7
Build area.exe F7
Rebuild All
Batch Build...
Clean
Update All Dependencies...
Start Debug
Debugger Remote Connection...
Execute area.exe Ctrl+F5
Profile...
Workspace 'area': 1 project(s)
area files
Area.cpp
ClassView
FileView
InfoView
About the Visual C++ Editions
Explains the differences between the Professional, Enterprise, and Learning Editions.
Welcome to Visual C++ 5.0! The documentation is
Build
Debug
Find in Files 1
Find in Files 2
Compiles the file

Click **Build** then **Compile** Area.cpp. This causes the compiler to start **translating the C++ source program into the machine language instructions**

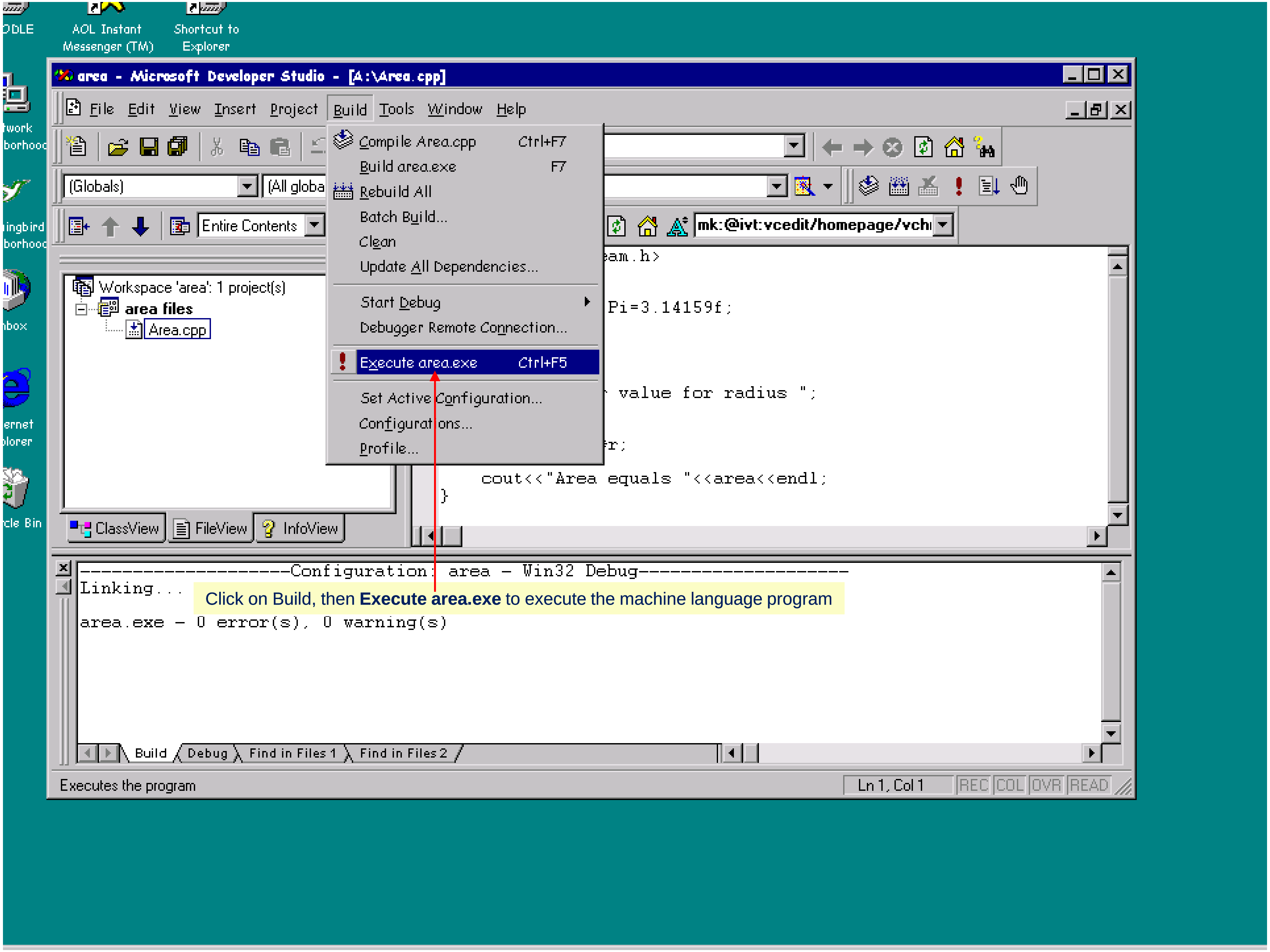
```
--------------------Configuration: area - Win32 Debug--------------------
Compiling...
Skipping... (no relevant changes detected)
Area.cpp

Area.obj - 0 error(s), 0 warning(s)
```



```
eam.h>
Pi=3.14159f;
 value for radius ";
```

```
--------
Linkir
area.e
```

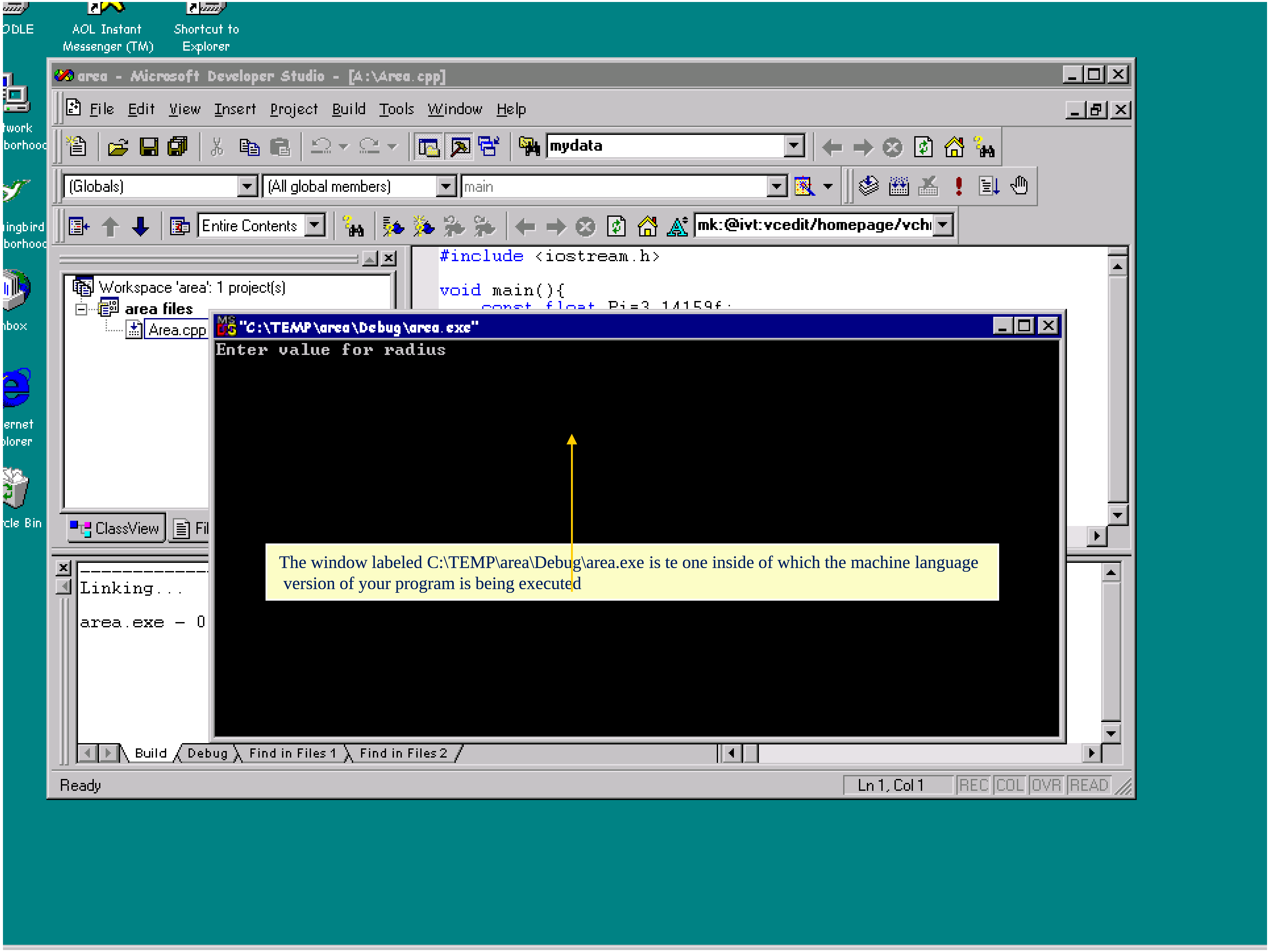
Click **Build** then **Build area.cpp** to perform the linking process for the resulting program form the compiler
To obtain the executable (machine language version of your program a process called linking must be performed
linking is the process of attaching code from libraries that you did not write yourself to the program you wrote yourself. For example the statement
#include<iostream.h>
Actually is telling C++ linker to bring in the code of the library called iostream. This is basically a program that enables
you to print on the screen by simply using one statement cout<<….There is a whole program behind the cout statementthat is found in the library iostream.



```
eam.h>
Pi=3.14159f;
 value for radius ";
r;
    cout<<"Area equals "<<area<<endl;
}
```

```
--------------------Configuration: area - Win32 Debug--------------------
Linking...

area.exe - 0 error(s), 0 warning(s)
```

Click on Build, then **Execute area.exe** to execute the machine language program



AOL Instant Messenger (TM)
Shortcut to Explorer
area - Microsoft Developer Studio - [A:\Area.cpp]
File Edit View Insert Project Build Tools Window Help
mydata
(Globals)
(All global members)
main
Entire Contents
mk:@ivt:vcedit/homepage/vchi
Workspace 'area': 1 project(s)
area files
Area.cpp
ClassView
Build
Debug
Find in Files 1
Find in Files 2
Ready
Ln 1, Col 1
REC COL OVR READ

```
#include <iostream.h>

void main(){
    const float Pi=3.14159f;
```

"C:\TEMP\area\Debug\area.exe"

```
Enter value for radius
```

```
Linking...

area.exe - 0
```

The window labeled C:\TEMP\area\Debug\area.exe is te one inside of which the machine language version of your program is being executed

1. A Simple Example of C

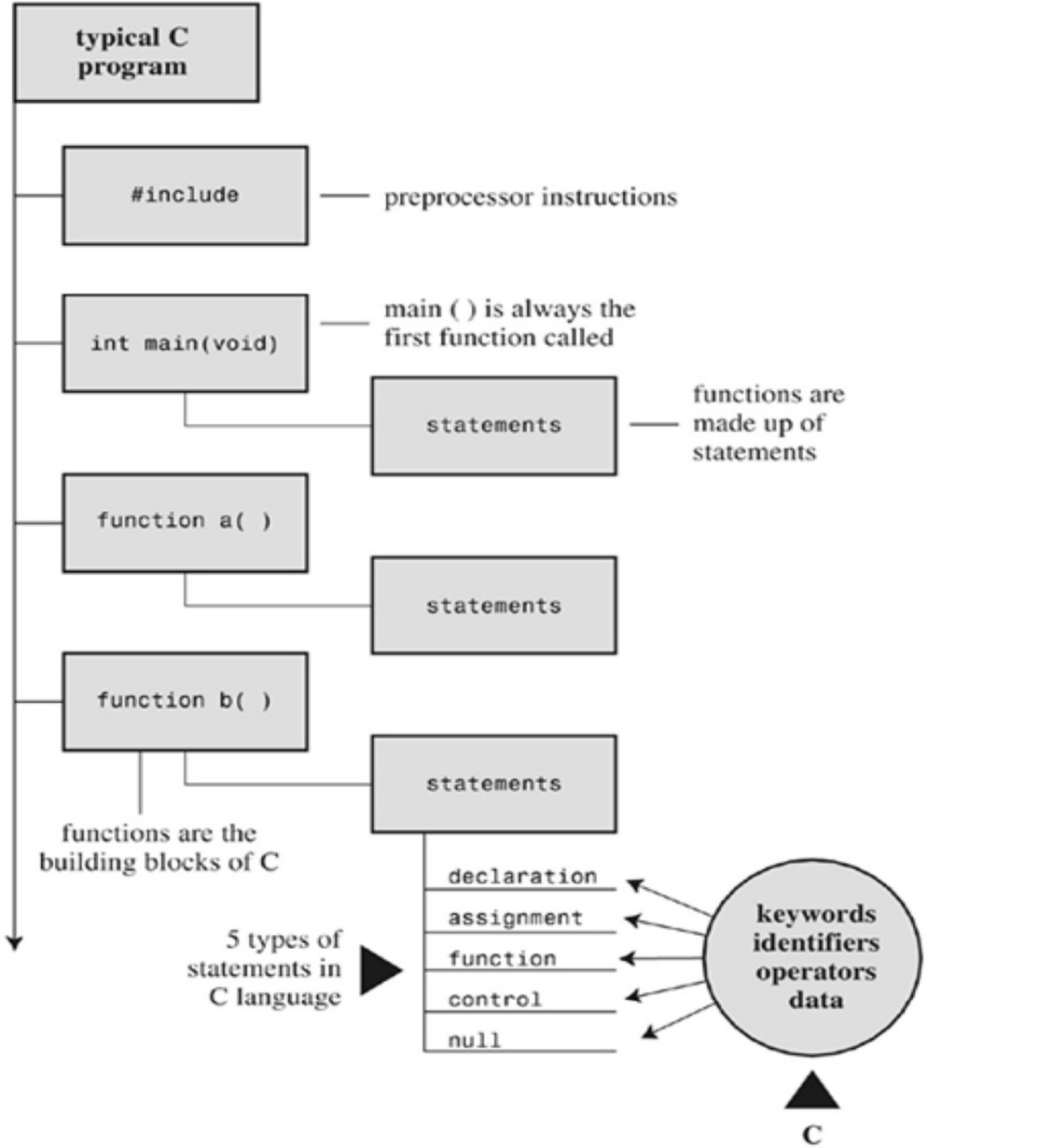
Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                          /* a simple program */
{
        int num;                                /* define a variable called num */
        num = 1;                                /* assign a value to num */

        printf("I am a simple ");               /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

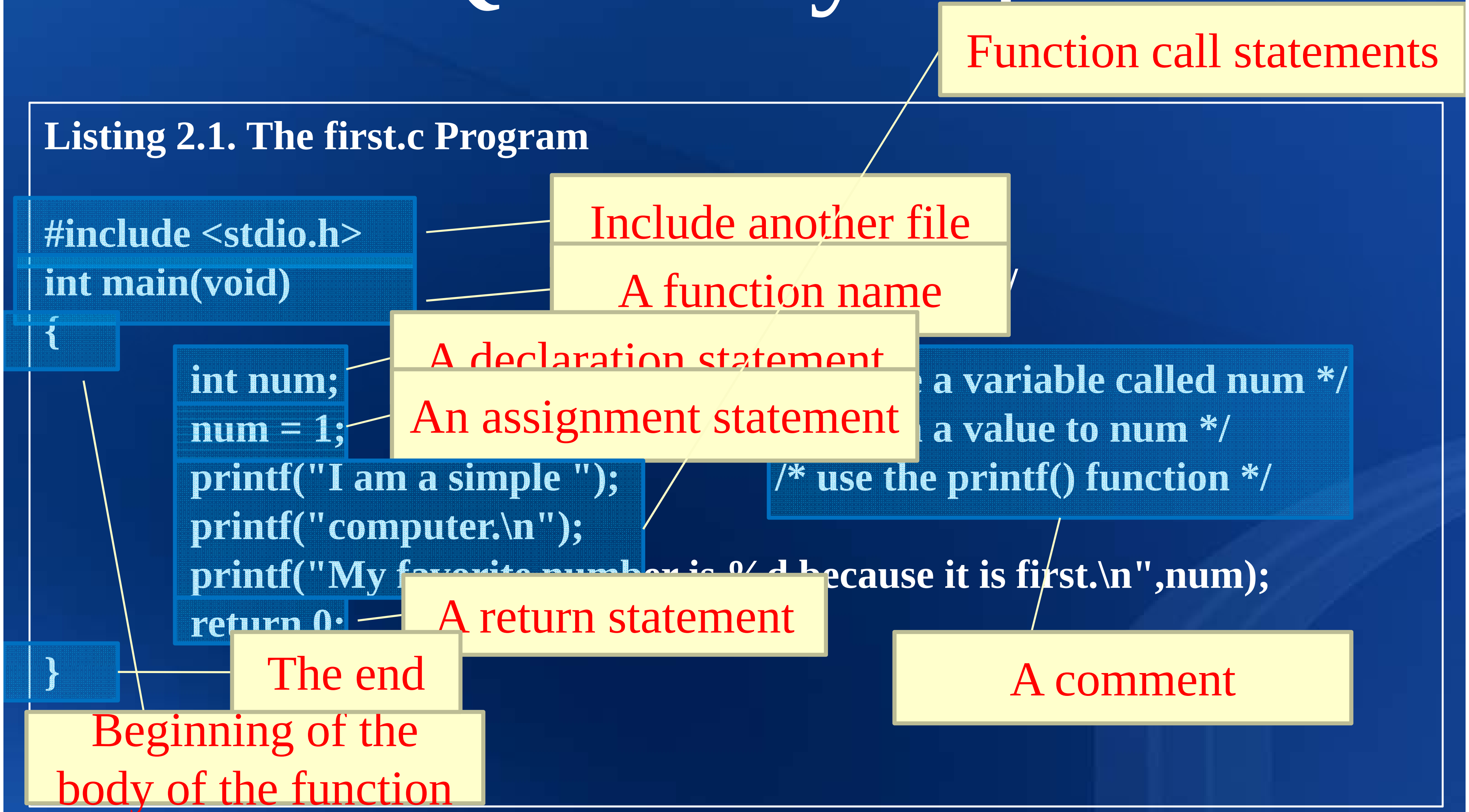
```
I am a simple computer.
My favorite number is 1 because it is first.
```

2. The Example Explained

- Take **two passes**
  - Pass 1: Quick Synopsis
  - Pass 2: Program Details


typical C program
#include
preprocessor instructions
int main(void)
main ( ) is always the first function called
statements
functions are made up of statements
function a( )
statements
function b( )
statements
functions are the building blocks of C
5 types of statements in C language
declaration
assignment
function
control
null
keywords
identifiers
operators
data
C

Pass 1: Quick Synopsis



Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)
{
        int num;                          a variable called num */
        num = 1;                          a value to num */
        printf("I am a simple ");        /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

Pass 2: Program Details

Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                  /* a simple program */
{
        int num;                        /* define a variable called num */
        num = 1;                        /* assign a value to num */
        printf("I am a simple ");       /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

Include another file

#include **Directives and Header Files**

- *#include <stdio.h>*
- The line that **begins** the program
- The same as if you had typed the entire contents of the stdio.h file into your file at the point where the *#include* line appears
- A **cut-and-paste operation**
- *include* files provide a convenient way to **share information** that is common to many programs
- An example of a C preprocessor directive – ***Preprocessing***

#include Directives and Header Files

- The stdio.h file is supplied as part of all C compiler packages
- Contains information about input and output functions
- The name stands for **standard input/output header**
- C people call a collection of information that goes at the top of a file **a header**, and C implementations typically come with several header files
- Header files contain information used by the compiler to build the final executable program
- The actual code **precompiled co**

Header files help guide the compiler in putting your program together correctly

Question

Why Are Not Input and Output Built In?

One answer is that not all programs use this I/O (input/output) package, and part of the C philosophy is to avoid carrying unnecessary weight.

Pass 2: Program Details

Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                                        /
{
        int num;                          /* define a variable called num */
        num = 1;                          /* assign a value to num */
        printf("I am a simple ");         /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

A function name

The *main()* Function

- *int main(void)*
- A C program **always begins execution with the function called *main()***
- You are free to choose names for other functions you use, but *main()* must be there to start things
- What about the parentheses? – **identify** *main()* as **a function**

Functions are the basic modules of a C program

The *main()* Function

- The ***int*** is the main() function's **return type** – the kind of value main() can return is an integer
- Return where? **To the Operating System** (Keep it for now)
- The **parentheses** following a function name generally **enclose information being passed along to the function**
- For this simple example, nothing is being passed along, so the parentheses contain the word ***void***
- The different formats of main (): *main()* or *void main()*

Pass 2: Program Details

Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                          /* a simple program */
{
        int num;                                    /* define a variable called num */
        num = 1;                                    /* assign a value to num */
        printf("I am a simple ");                   /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

A comment

Comments

- */* a simple program */*
- The parts of the program enclosed in the /* */ symbols are comments
- Can be placed **anywhere**
- Everything between the opening /* and the closing */ is **ignored** by the compiler

```
/* This is a C comment. */
/* This comment is spread over
two lines. */
/*
You can do this, too.
*/
```

```
/* But this is invalid
because there is no end
marker
```

Comments

- Use the symbols // to create comments that are confined to a single line
- Because the end of the line marks the end of the comment, this style needs comment markers just at the beginning of the comment.

```
// Here is a comment confined to one line.
int rigue; // Such comments can go here, too.
```

```
/*
I hope this works.
*/
    100
```

```

/*
I hope this works.
    200
```

Because the // form doesn't extend over more than one line, it can't lead to this "disappearing code" problem.

Pass 2: Program Details

Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                          /* a simple program */
{
        int num;                                /* define a variable called num */
        num = 1;                                /* assign a value to num */
        printf("I am a simple ");               /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

The end

Beginning of the body of the function

Braces, Bodies, and Blocks

```
{
...
}
```

- Braces **delimited** the *main()* function.
- In general, **all C functions** use **braces** to **mark the beginning as well as the end of the body of a function**.
- It is mandatory, so don't leave them out
- Only **braces ({ })** work for this purpose, not parentheses (( )) and not brackets ([ ])
- Braces can also be used to **gather statements within a function into a unit or block**

Pass 2: Program Details

Listing 2.1. The first.c Program

```
#include <stdio.h>
int main(void)                    /* a simple program */
{
        int num;                          a variable called num */
        num = 1;                          /* assign a value to num */
        printf("I am a simple ");         /* use the printf() function */
        printf("computer.\n");
        printf("My favorite number is %d because it is first.\n",num);
        return 0;
}
```

A declaration statement

Declarations

- *int num;*
- Declare two things:
  - Somewhere in the function, you have **a variable** called *num*.
  - The *int* proclaims *num* as **an integer**—that is, a number without a decimal point or fractional part.
- The compiler uses this information to arrange for **suitable storage space in memory** for the *num* variable.
- The semicolon at the end of the line identifies the line as a C **statement** or **instruction**.

Declarations

- The word *int* is a C **keyword** identifying one of the basic C data types.
- Keywords are **the words used to express a language, and you can't usurp them for other purposes**.
- The word *num* in this example is an **identifier**—that is, **a name you select fo** **other entity**.
- The declaration **con** particular location in establishes the type c stored at that locatio

Traditionally, C has required that variables be declared at the beginning of a block with no other kind of statement allowed to come before any of the declarations.

In C, **all** variables must be declared **before** they are used.

A Simple Example of C

```
int main()      // traditional rules
{
        int doors;
        int dogs;

        doors = 5;
        dogs = 3;
        // other statements
}
```

A Simple Example of C

```
int main()        // C99 rules
{
// some statements
        int doors;
        doors = 5; // first use of doors
// more statements
        int dogs;
        dogs = 3; // first use of dogs
// other statements
}
```

Questions

- What are data types?

Data Types

- C deals with **several kinds** (or types) of data: integers, characters, and floating point, for example
- Declaring a variable to be an integer or a character type makes it possible for the computer to store, fetch, and interpret the data properly

Questions

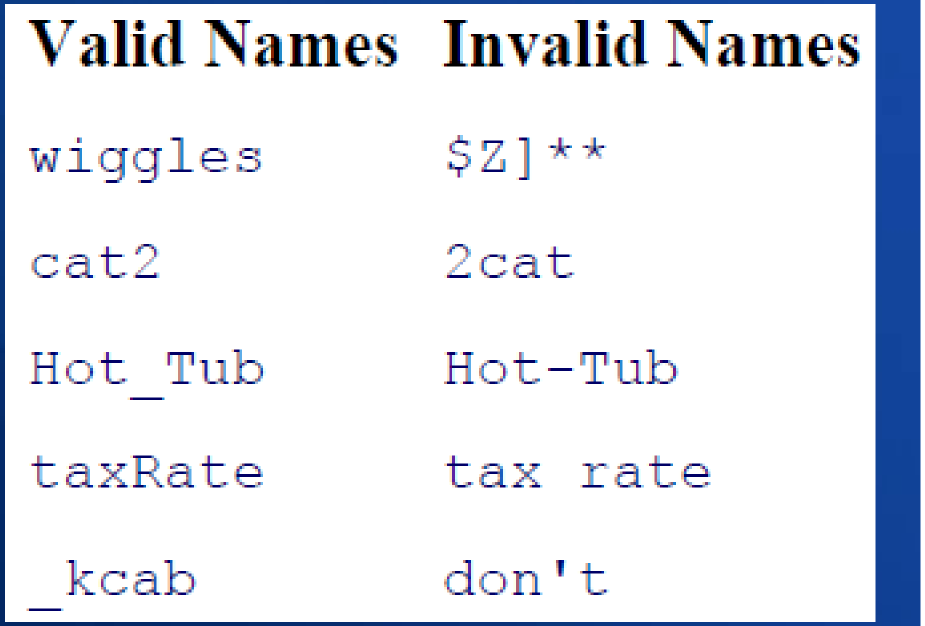
- What choices do you have in selecting a name?

Name Choice

- Use **meaningful** names for variables
- If the name doesn't suffice, use **comments** to explain what the variables represent.
- The number of characters
  - The C99 standard calls for **up to 63 characters**, except for **external identifiers**, for which only 31 characters need to be recognized.
  - You can use more than the maximum number of characters, but the compiler won't pay attention to the extra characters.

Name Choice

- The characters at your disposal are **lowercase letters**, **uppercase letters**, **digits**, and **the underscore (_)**.
- The first character must be a letter or an underscore.
- C names are **case sensitive**



Valid Names	Invalid Names
`wiggles`	`$Z]**`
`cat2`	`2cat`
`Hot_Tub`	`Hot-Tub`
`taxRate`	`tax rate`
`_kcab`	`don't`

Operating systems and the C library often use identifiers with one or two initial underscore characters, such as in _kcab, so it is better to avoid that usage yourself.

Questions

- Why do you have to declare variables at all?

Four Good Reasons to Declare Variables

- Putting all the variables in one place makes it **easier** for a reader to grasp what the program is about.
- Thinking about which variables to declare encourages you to **do some planning** before plunging into writing a program.
- Declaring variables helps **prevent** one of

For example, suppose that in some language that lacks declarations, you made the statement

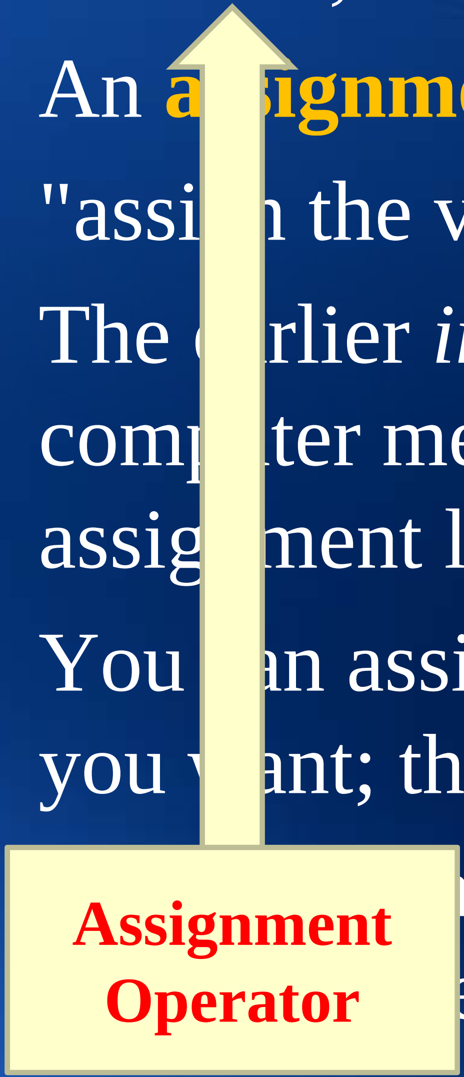
• RADIUS1 = 20.4;

and that elsewhere in the program you mistyped

CIRCUM = 6.28 * RADIUSl;

Assignment

- *num = 1;*
- An **assignment statement**
- "assign the value 1 to the variable *num*."
- The earlier *int num;* line set aside space in computer memory for the variable num, and the assignment line stores a value in that location.
- You can assign *num* a different value later, if you want; that is why *num* is termed a **variable**
- ent statement assigns a value from e to the left side



Assignment Operator

The printf() Function

- *printf("I am a simple ");*
- *printf("computer.\n");*
- *printf("My favorite number is %d because it is first.\n", num);*
- The parentheses signify that printf is **a function name**.
- The material enclosed in the parentheses is **information passed from the main() function to the printf() function**.
- Such information is called the **argument** or, more fully**, the actual argument** of a function

The printf() Function

- The *printf("I am a simple ");* line is an example of how you **call** or **invoke** a function in C.
- You need type only the name of the function, placing the desired argument(s) within the parentheses.
- When the program reaches this line, **control** is **turned over** to the named function
- When the function is finished with whatever it does, **control** is **returned to** the original (the *calling*) function—*main()*, in this example.

The printf() Function

- *printf("computer.\n");* line
- The characters \n – didn't get printed!
- The \n symbol means to start a new line.
- The \n combination (typed as two characters) represents a single character called the *newline character*.
- To printf(), it means "start a new line at the far-left margin."
- Perform the same function as pressing the Enter key of a typical keyboard.

Questions

- Why not just use the Enter key when typing the printf() argument?

The printf() Function

- Because that would be interpreted as **an immediate command to your editor**, not as **an instruction** to be stored in your source code.
- The newline character is an example of an ***escape sequence***.
- An escape sequence is used to represent **difficult-or impossible-to-type characters**.
- In each case, the escape sequence begins with the backslash character, \

The printf() Function

- *printf("My favorite number is %d because it is first.\n", num);*
- What happened to the %d when the line was printed?
- The %d is a **placeholder** to show where the value of num is to be printed.
- The % alerts the program that a variable is to be printed at that location, and the d tells it to print the variable as a decimal (base 10) integer.
- Allow several choices for the format of printed variables

Return Statement

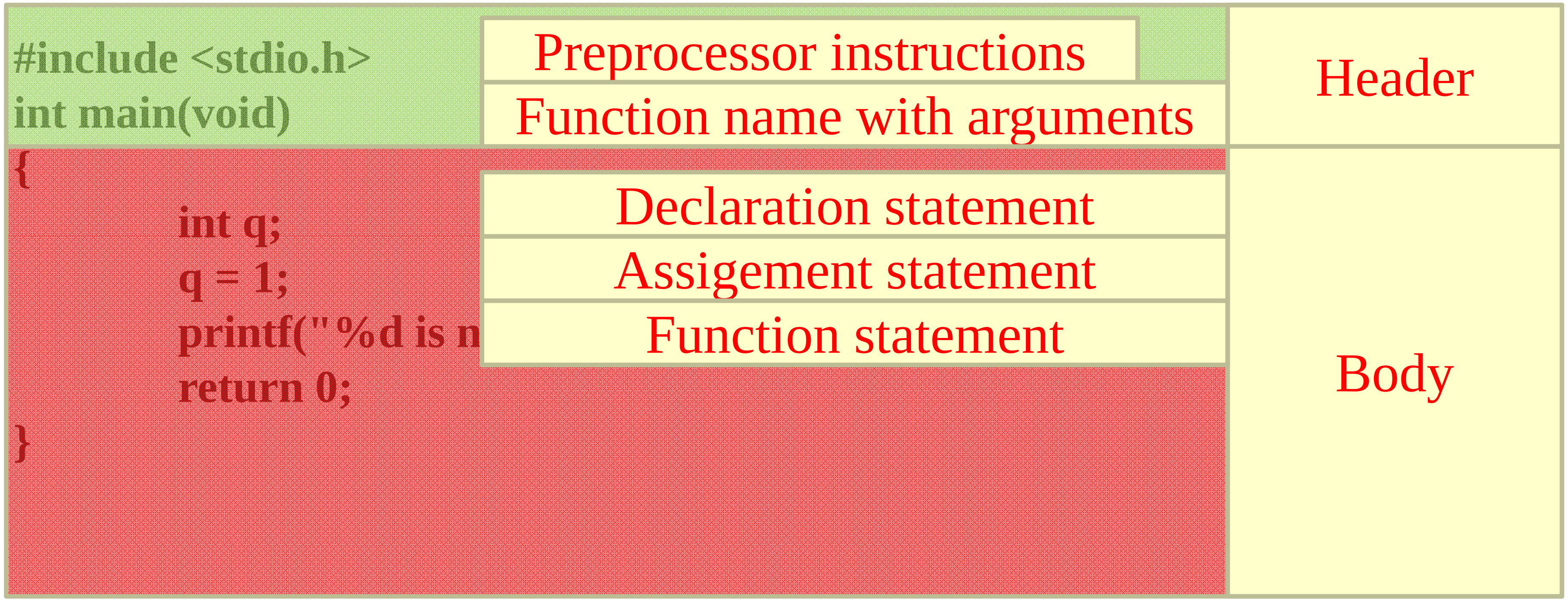
- *return 0;*
- The **final statement** of the program.
- C functions that return values do so with a return statement, which consists of the keyword **return**, **followed by the returned value**, followed by a semicolon.
- Regard the return statement in main() as something required for logical consistency, but it has a practical use with some operating systems

3. The Structure of a Simple Program

- A few general rules about C programs:
  - A program consists of a collection of one or more functions
  - one of which must be called ***main()***.
  - The description of a function consists of *a* ***header*** and *a* ***body***.
  - The *header* contains *preprocessor statements*
  - The body is enclosed by braces ({}) and consists of a series of statements, each terminated by a semicolon

A Simple Example of C

Listing 2.1. The first.c Program



```
#include <stdio.h>
int main(void)
{
		int q;
		q = 1;
		printf("%d is n
		return 0;
}
```

A Simple Standard C Program

```
#include <stdio.h>
int main(void)
{
        statements
        return 0;
}
```

Tips on Making Your Programs Readable

- Making your programs **readable** is good programming practice:
  - Choose meaningful variable names
  - Use comments
  - Note that these two techniques complement each other
  - Use blank lines to separate one conceptual section of a function from another
  - Use one line per statement
- This is a readability convention, not a C requirement

C has a free-form format.

Legitimate But Ugly Code

```
int main( void ) { int four; four
=
4
;
printf(
"%d\n",
four); return 0;}
```

Making your program readable

```
int main(void) /* converts 2 fathoms to feet */

{
int feet, fathoms;

fathoms=2;
feet=6*fathoms;
printf("There are %d feet in %d fathoms! \n", feet, fathoms);
return 0;
}
```

Use comments

Pick meaningful names

Use space

One statement per line

Taking Another Step in Using C

Listing 2.2. The fathm_ft.c Program

```
// fathm_ft.c -- converts 2 fathoms to feet
#include <stdio.h>
int main(void)
{
int feet, fathoms;
fathoms = 2;
feet = 6 * fathoms;
printf("There are %d feet in %d fathoms!\n", feet, fathoms);
printf("Yes, I said %d feet!\n", 6 * fathoms);
return 0;
}
```

Provide a program description

Declare multiple variables

Do some multiplication

Print the values of two variables

Taking Another Step in Using C

- What's new?
  - Provide a program description
  - Declare multiple variables
  - Do some multiplication
  - Print the values of two variables

Documentation

- Use the new comment style
- Take but a moment to do and be helpful later when you browse through several files or print them.

Multiple Declarations

- Declare two variables instead of just one in a single declaration statement
- Separate the two variables (*feet* and *fathoms*) by a **comma** in the declaration statement

```
int feet, fathoms;
```

```
int feet;
int fathoms;
```

Multiplication

- It harnesses the tremendous computational power of a computer system to multiply 2 by 6.
- * is the symbol for multiplication

```
feet = 6 * fathoms;
```

- "look up the value of the variable *fathoms*, multiply it by 6, and assign the result of this calculation to the variable *feet*."

Printing Multiple Values

```
printf("There are %d feet in %d fathoms!\n", feet, fathoms);
printf("Yes, I said %d feet!\n", 6 * fathoms);
```

- If you compile and run the example, the output should look like this:

```
There are 12 feet in 2 fathoms!
Yes, I said 12 feet!
```

- The code made **two substitutions** in the first use of *printf():*
  - The first %d was replaced by the value of the first variable (feet) in the list following the quoted segment
  - The second %d was replaced by the value of the second variable (fathoms) in the list.

Printing Multiple Values

- Note that the list of variables to be printed comes at the tail end of the statement after the quoted part.
- Note that each item is separated from the others by a **comma**.
- The second use of *printf()* illustrates that the value printed doesn't have to be a variable: It just has to be something, such as 6 * fathoms, that reduces to a value of the right type.

While You're at It—Multiple Functions

Show you how to incorporate a function of your own—besides main()—into a program.

Listing 2.3. The two_func.c Program

```
/* two_func.c -- a program using two functions in one file */
#include <stdio.h>
void butler(void); /* ISO/ANSI C function prototyping */
int main(void)
{
printf("I will summon the butler function.\n");
butler();
printf("Yes. Bring me some tea and writeable CD-ROMS.\n");
return 0;
}
void butler(void) /* start of function definition */
{
printf("You rang, sir?\n");
}
```

The Output of Listing 2.3

```
I will summon the butler function.
You rang, sir?
Yes. Bring me some tea and writeable CD-ROMS.
```

While You're at It—Multiple Functions

- The *butler()* function appears **three times** in this program:
  - The first appearance is in the ***prototype***, which informs the compiler about the functions to be used.
  - The second appearance is in *main()* in the form of a ***function call***.
  - Finally, the program presents the ***function definition***, which is the source code for the function itself.

While You're at It—Multiple Functions

- What is the prototype? – ***A prototype*** is **a form of declaration** that tells the compiler that you are using a particular function. It also specifies **properties of the function**.

Listing 2.3. The two_func.c Program

```
/* two_func.c -- a program using two functions in one file */
#include <stdio.h>
void butler(void); /* ISO/ANSI C function prototyping */
int main(void)
{
printf("I will summon the butler function.\n");
butler();
D-ROMS.\n");
return 0;
}
void butler(void) /* start of function definition */
{
printf("You rang, sir?\n");
}
```

The first *void* in the prototype for the butler() function indicates that butler() does not have a return value.

The second *void* means that the butler() function has no arguments.

Note that void is used to mean "empty," not "invalid."

In general, a function can return a value to the calling function for its use, but *butler()* doesn't.

While You're at It—Multiple Functions

- Older C supported a more limited form of function declaration
- You just specified the return type but omitted describing the arguments

```
void butler();
```

- Older C code uses function declarations like the preceding one instead of function prototypes.

While You're at It—Multiple Functions

- Next, you **invoke** butler() in main() simply by giving its name, including parentheses.

Listing 2.3. The two_func.c Program

```
/* two_func.c -- a program using two functions in one file */
#include <stdio.h>
void butler(void); /* ISO/ANSI C function prototyping */
int main(void)
{
printf("I will summon the butler function.\n");
butler();
printf("Yes. Bring me some tea and writeable CD-ROMS.\n");
return 0;
}
void butler(void) /* start of function definition */
{
printf("You rang, sir?\n");
}
```

Give its name, including parentheses

While You're at It—Multiple Functions

- Finally, the function butler() is defined **in the same manner as main()**, with **a function header** and **the body enclosed in braces**
- The header repeats the information given in the prototype: butler() **takes no arguments and has no return value**
- For older compilers, omit the second void.

One point to note is that it is the location of the butler() call in main()—not the location of the butler() definition in the file—that determines when the butler() function is executed.

Listing 2.3. The two_func.c Program

```
/* two_func.c -- a program using two functions in one file */
#include <stdio.h>
void butler(void); /* ISO/ANSI C function prototyping */
int main(void)
{
printf("I will summon the butl
butler();
printf("Yes. B
return 0;
}
void butler(void) /* start of function definition */
{
printf("You rang, sir?\n");
}
```

Be defined in the same manner as main()

A function header

The body enclosed in braces

However, C practice is to list main() first because it normally provides the basic framework for a program.

Remember, all C programs begin execution with main(), no matter where main() is located in the program files.

While You're at It—Multiple Functions

- The C standard recommends that you **provide function prototypes for all functions you use**.
- The standard **include** files take care of this task for **the standard library functions**.

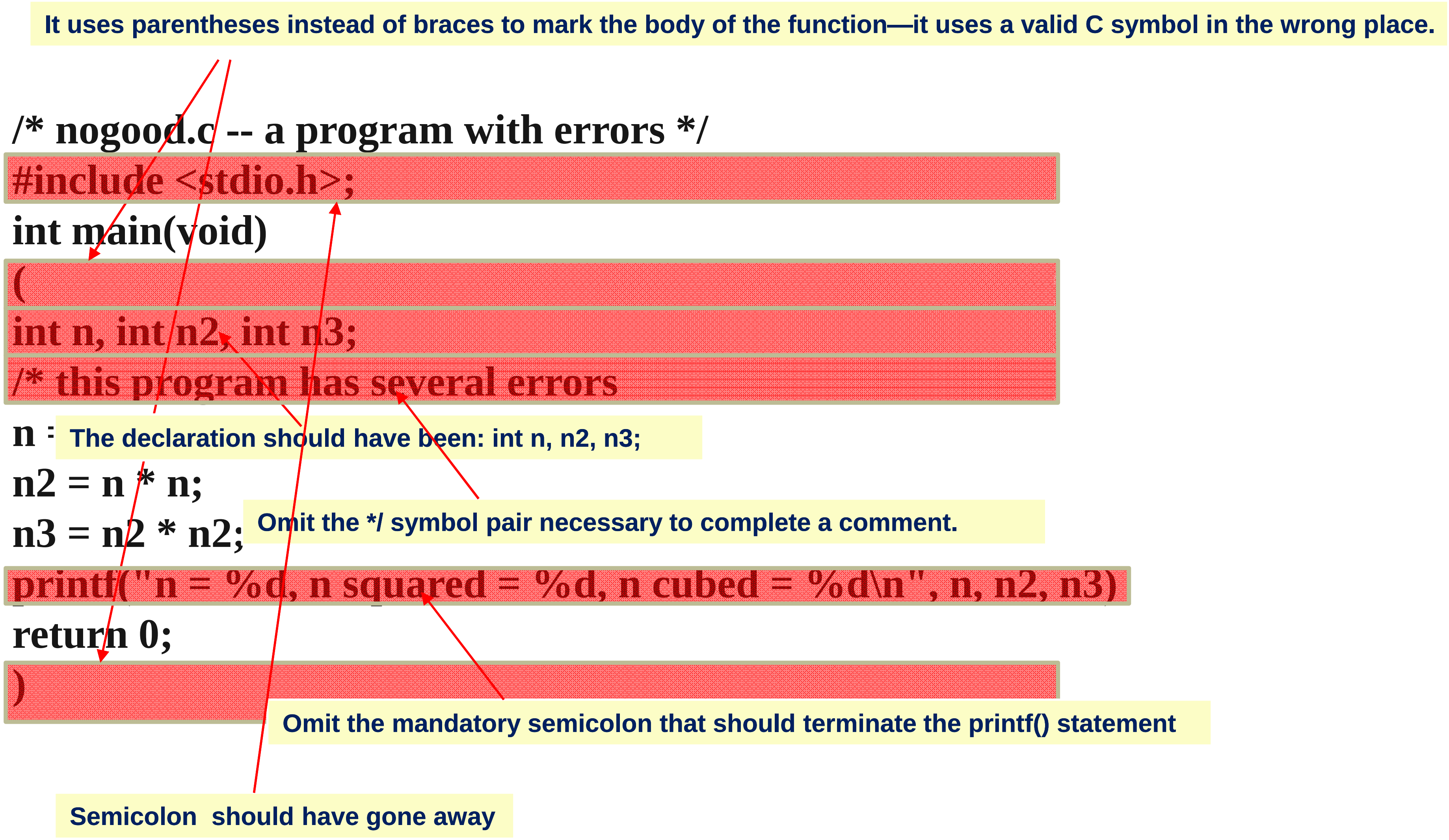
Introducing Debugging

- Program errors often are called ***bugs***.
- Finding and fixing the errors is called ***debugging***.
- You commit a **syntax error** when you don't follow C's rules (It's analogous to a grammatical error in English).

Bugs frustrate be can.

- C syntax errors use valid C symbols **in the wrong places**.

Listing 2.4. The nogood.c Program



It uses parentheses instead of braces to mark the body of the function—it uses a valid C symbol in the wrong place.

```
/* nogood.c -- a program with errors */
#include <stdio.h>;
int main(void)
(
int n, int n2, int n3;
/* this program has several errors
n =
n2 = n * n;
n3 = n2 * n2;
printf("n = %d, n squared = %d, n cubed = %d\n", n, n2, n3)
return 0;
)
```

The declaration should have been: int n, n2, n3;

Omit the */ symbol pair necessary to complete a comment.

Omit the mandatory semicolon that should terminate the printf() statement

Semicolon should have gone away

- What syntax errors did nogood.c make?

Syntax Errors

- How do you detect syntax errors?
  - First, before compiling, you can **look through the source code** and see whether you spot anything obvious.
  - Second, you can **examine errors found by the compiler** because part of its job is to detect syntax errors.
  - However, **the compiler can get confused**. A true syntax error in one location might cause the compiler to mistakenly think it has found other errors.
  - In fact, rather than trying to correct all the reported errors at once, you should **correct just the first one or two and then recompile**; some of the other errors may go away.
  - Continue in this way until the program works.
  - Another common compiler trick is reporting the error a line late.

Semantic Errors

- Semantic errors are errors in meaning.

Furry inflation thinks greenly.

- The syntax is fine because adjectives, nouns, verbs, and adverbs are in the right places, but the sentence doesn't mean anything.
- In C, you commit a semantic error when you **follow the rules of C correctly but to an incorrect end**.
- The compiler does not detect semantic errors, because they don't violate C rules.

Listing 2.5. The stillbad.c Program

```
/* stillbad.c -- a program with its syntax errors fixed */
#include <stdio.h>
int main(void)
{
int n, n2, n3;
/* this program has a semantic error */
n = 5;
n2 = n * n;
n3 = n2 * n2;
printf("n = %d, n squared = %d, n cubed = %d\n", n, n2, n3);
return 0;
}
```

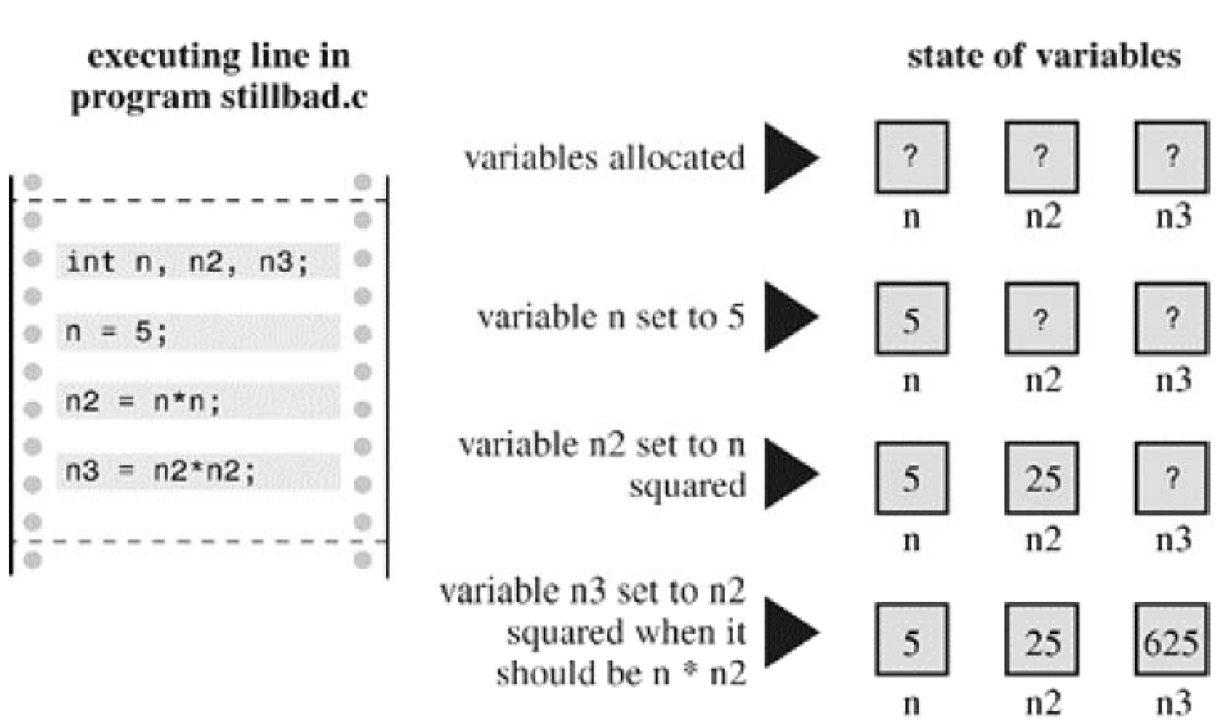
```
n = 5, n squared = 25, n cubed = 625
```

- You probably can spot the error by inspection.
- In general, however, you need to take a more systematic approach.

Semantic Errors

- One method is to pretend you are the computer and to follow the program steps one by one.
  - The body of the program starts by declaring three variables: n, n2, and n3.

Figure 2.6. Tracing a program.



```
int n, n2, n3;
n = 5;
n2 = n*n;
n3 = n2*n2;
```

executing line in program stillbad.c
state of variables
variables allocated
?
?
?
n
n2
n3
variable n set to 5
5
?
?
n
n2
n3
variable n2 set to n squared
5
25
?
n
n2
n3
variable n3 set to n2 squared when it should be n * n2
5
25
625
n
n2
n3

Perhaps this procedure is overkill for this example, but going through a program step-by-step in this fashion is often the best way to see what's happening.

Program State

- By tracing the program step-by-step manually, keeping track of each variable, you monitor the **program state**.
- The program state is simply **the set of values of all the variables at a given point in program execution**.
- A **snapshot** of the current state of computation.
- Execute the program step-by-step yourself.
- In a large program, you can go through a few

However, there is always the possibility that you will execute the steps as you intended them to be executed instead of as you actually wrote them, so try to be faithful to the actual code.

Program State

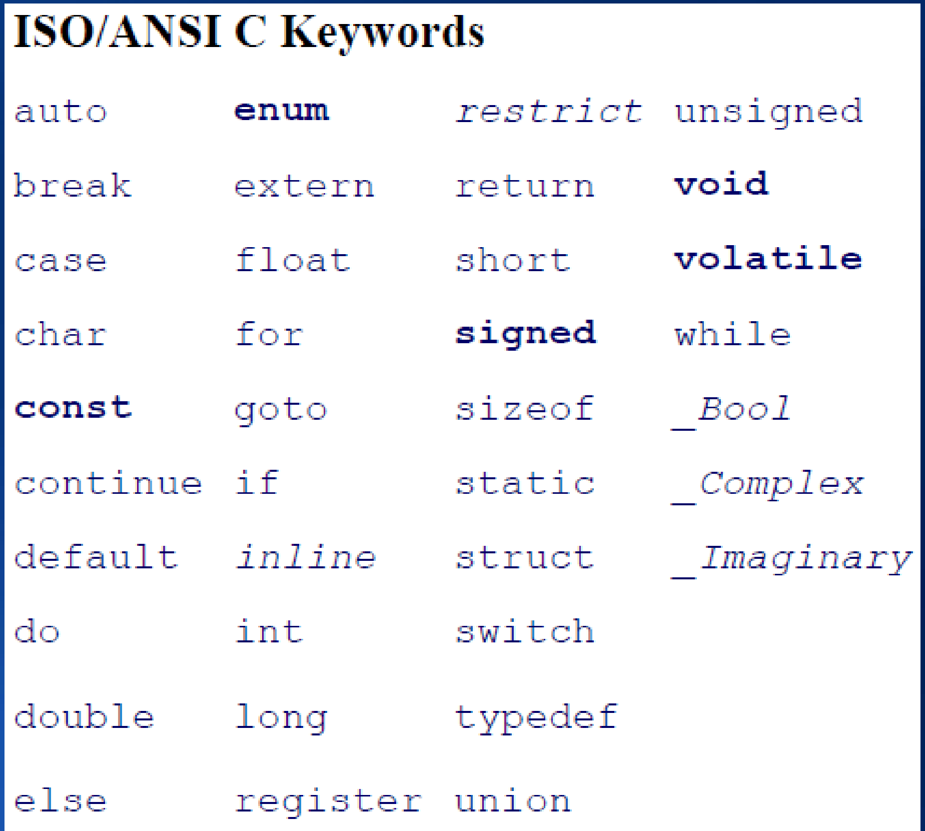
- Another approach to locating semantic problems is to **sprinkle extra printf() statements throughout to monitor the values of selected variables at key points in the program**.
- Seeing how the values change can illuminate what's happening.
- After you have the program working to your satisfaction, you can remove the extra statements and recompile.

Program State

- A third method for examining the program states is to use **a debugger**.
- A debugger is **a program** that enables you to run another program step-by-step and examine the value of that program's variables.
- Debuggers come in **various levels of ease of use and sophistication**.
- The more advanced debuggers show which line of source code is being executed.

Keywords and Reserved Identifiers

- **Keywords** are the vocabulary of C.
- Because they are special to C, you can't use them as identifiers or as variable names.



ISO/ANSI C Keywords

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

There are other identifiers, called **reserved identifiers**, that you shouldn't use: include those beginning with an underscore character and the names of the standard library functions, such as printf().

Key Concepts

- Computer programming is a challenging activity.
- It demands **abstract, conceptual thinking combined with careful attention to detail**.
- You'll find that **compilers enforce the attention to detail**.
- A compiler doesn't make such allowances; to it, **almost right is still wrong**.

Key Concepts

- For this chapter, your goal should be to **understand what a C program is**.
- You can think of a program as a **description** you prepare of how you want the computer to behave.
- The compiler handles the really detailed job of **converting your description to the underlying machine language**.
- Because the compiler has no real intelligence, you have to **express your description in the compiler's terms**, and these terms are **the formal**

Although restrictive, this still is far better than having to express your description directly in machine language!

Key Concepts

- The compiler **expects to receive its instructions in a specific format**
- Your job as a programmer is to **express your ideas about how a program should behave within the framework that the compiler**—guided by the C standard—can process successfully.

Summary

- A C program consists of one or more C functions.
- Every C program must contain a function called *main()* because it is the function called when the program starts up.
- A simple function consists of **a header** followed by **an opening brace**, followed by the **statements** constituting the function body, followed by a terminating, or **closing, brace**.

Summary

- Each C statement is an instruction to the computer and is marked by a terminating semicolon.
- A declaration statement creates a name for a variable and identifies the type of data to be stored in the variable.
- The name of a variable is an example of an identifier.
- An assignment statement assigns a value to a variable or, more generally, to a storage area.

Summary

- A function call statement causes the **named function** to be executed.
- When the called function is done, the program returns to the next statement after the function call.
- The **syntax** of a language is the set of rules that governs the way in which valid statements in that language are put together.
- The **semantics** of a statement is its meaning.
- **keywords** are the vocabulary of the C language.

Questions & Homeworks