A Review of C Language

C++ Object Oriented Programming

Pei-yih Ting

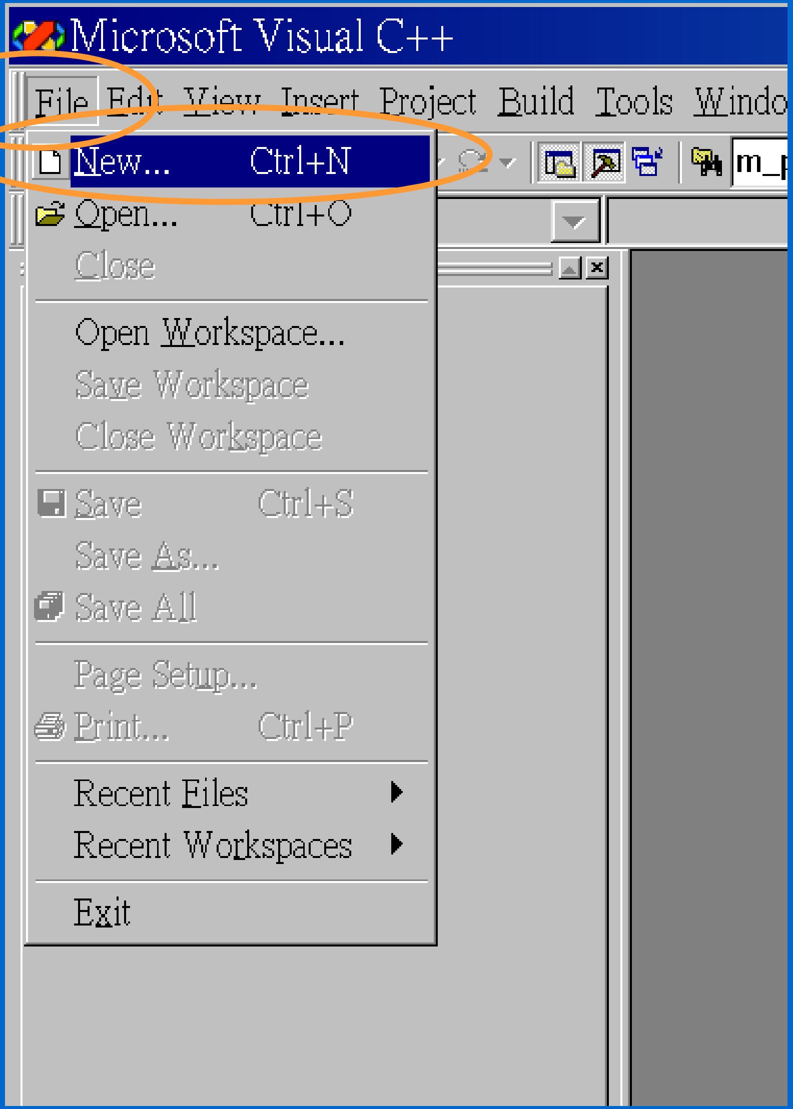
NTOU CS

Modified from www.**cse**.**c**uhk.edu.hk/~**csc**2520/tuto/**csc**2520_tuto01**.ppt**

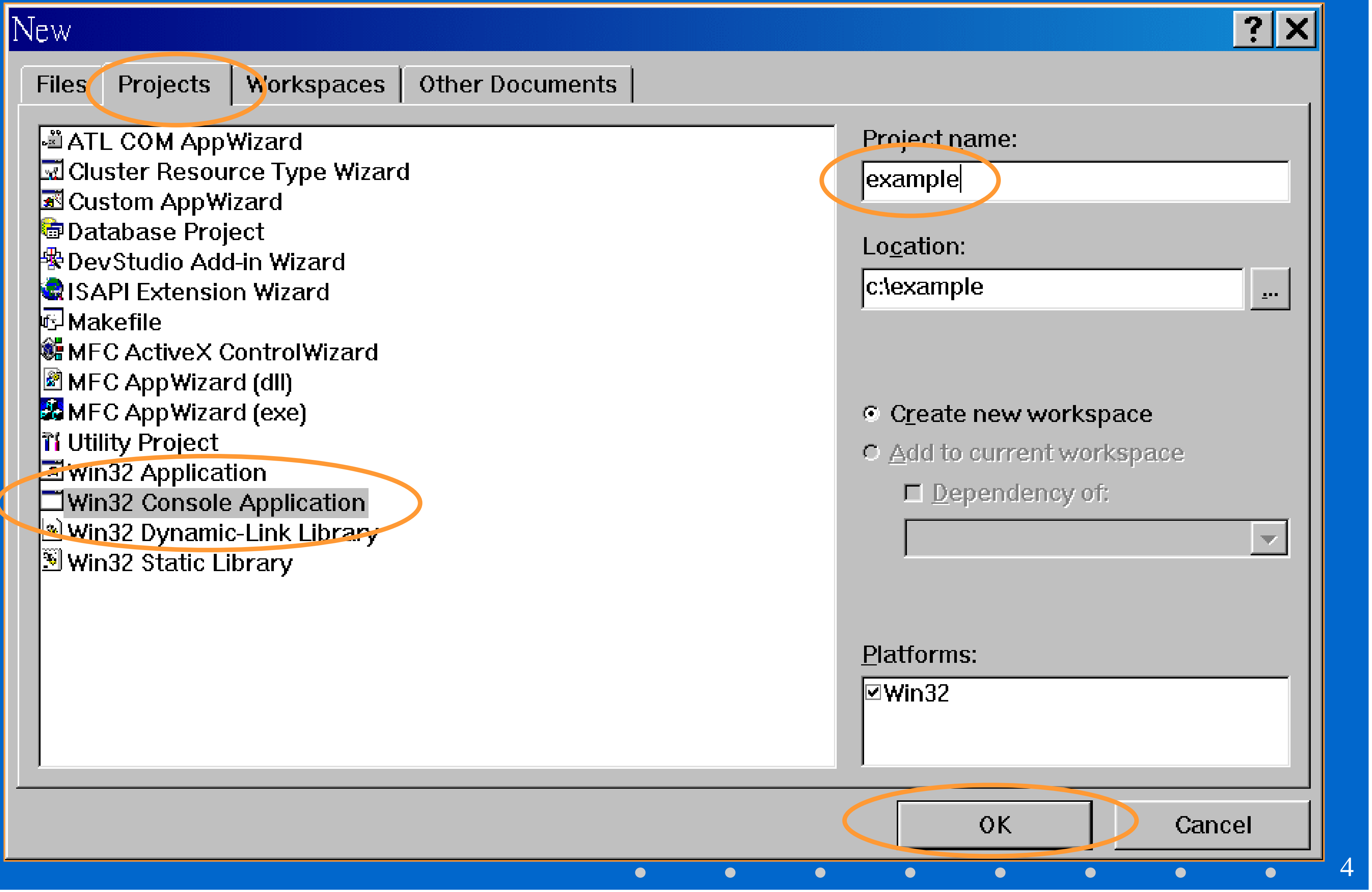
Contents

- **C Development Environment**
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

Visual C++ 6.0


Microsoft Visual C++
File Edit View Insert Project Build Tools Windo
New... Ctrl+N
Open... Ctrl+O
Close
Open Workspace...
Save Workspace
Close Workspace
Save Ctrl+S
Save As...
Save All
Page Setup...
Print... Ctrl+P
Recent Files
Recent Workspaces
Exit
m_

Visual C++ 6.0



New

Files | Projects | Workspaces | Other Documents

- ATL COM AppWizard
- Cluster Resource Type Wizard
- Custom AppWizard
- Database Project
- DevStudio Add-in Wizard
- ISAPI Extension Wizard
- Makefile
- MFC ActiveX ControlWizard
- MFC AppWizard (dll)
- MFC AppWizard (exe)
- Utility Project
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Static Library

Project name: example

Location: c:\example

● Create new workspace

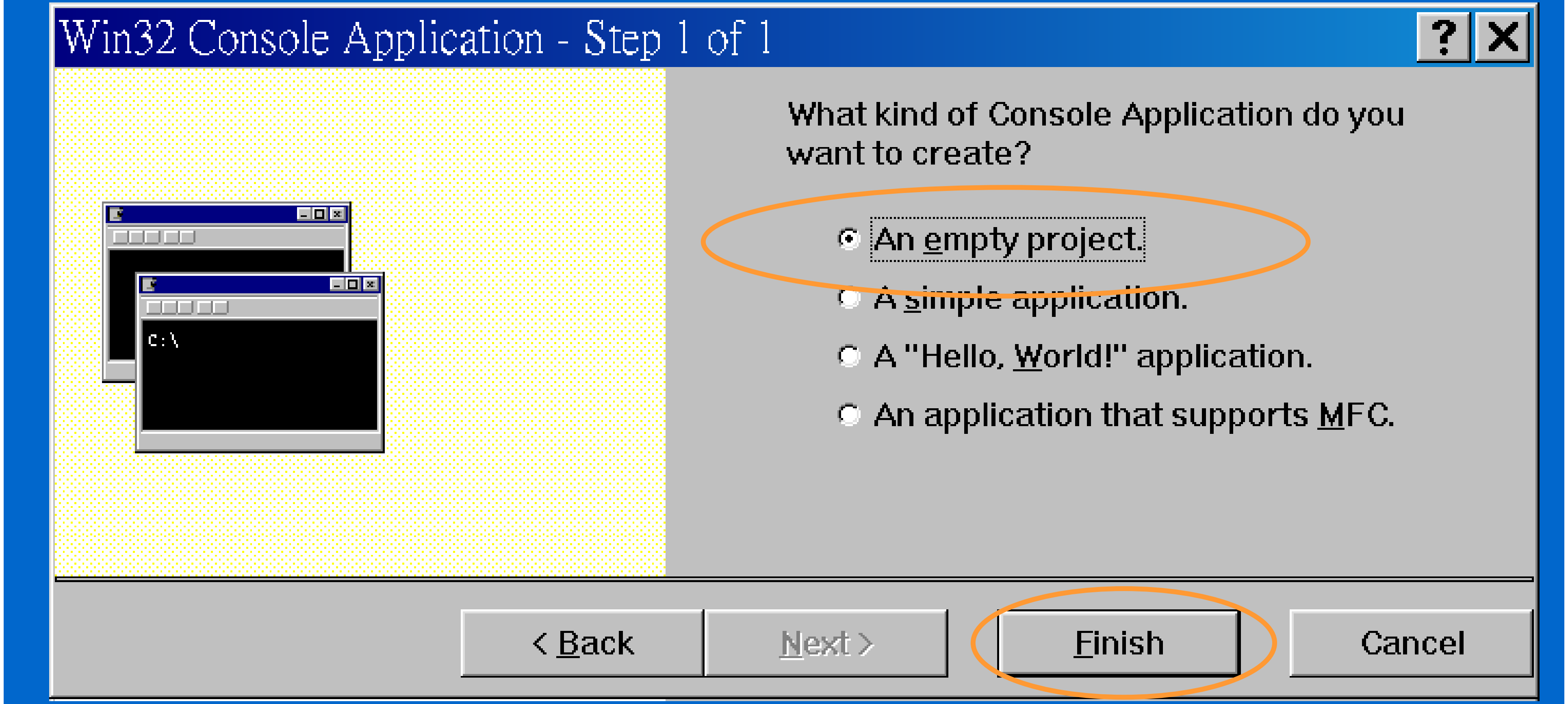
○ Add to current workspace

□ Dependency of:

Platforms:

☑ Win32

Visual C++ 6.0



Win32 Console Application - Step 1 of 1

What kind of Console Application do you want to create?

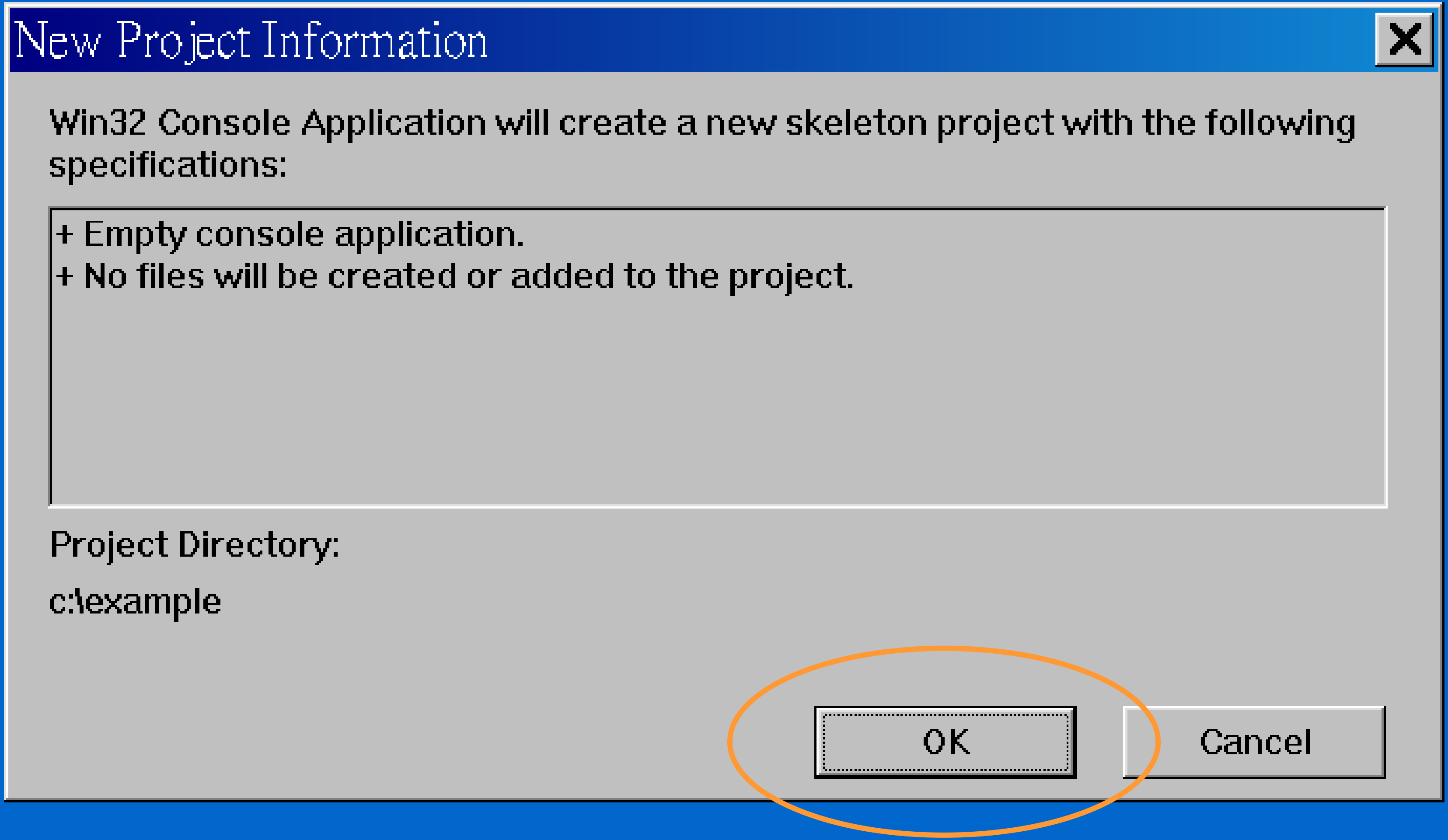
● An empty project.

○ A simple application.

○ A "Hello, World!" application.

○ An application that supports MFC.

Visual C++ 6.0



New Project Information

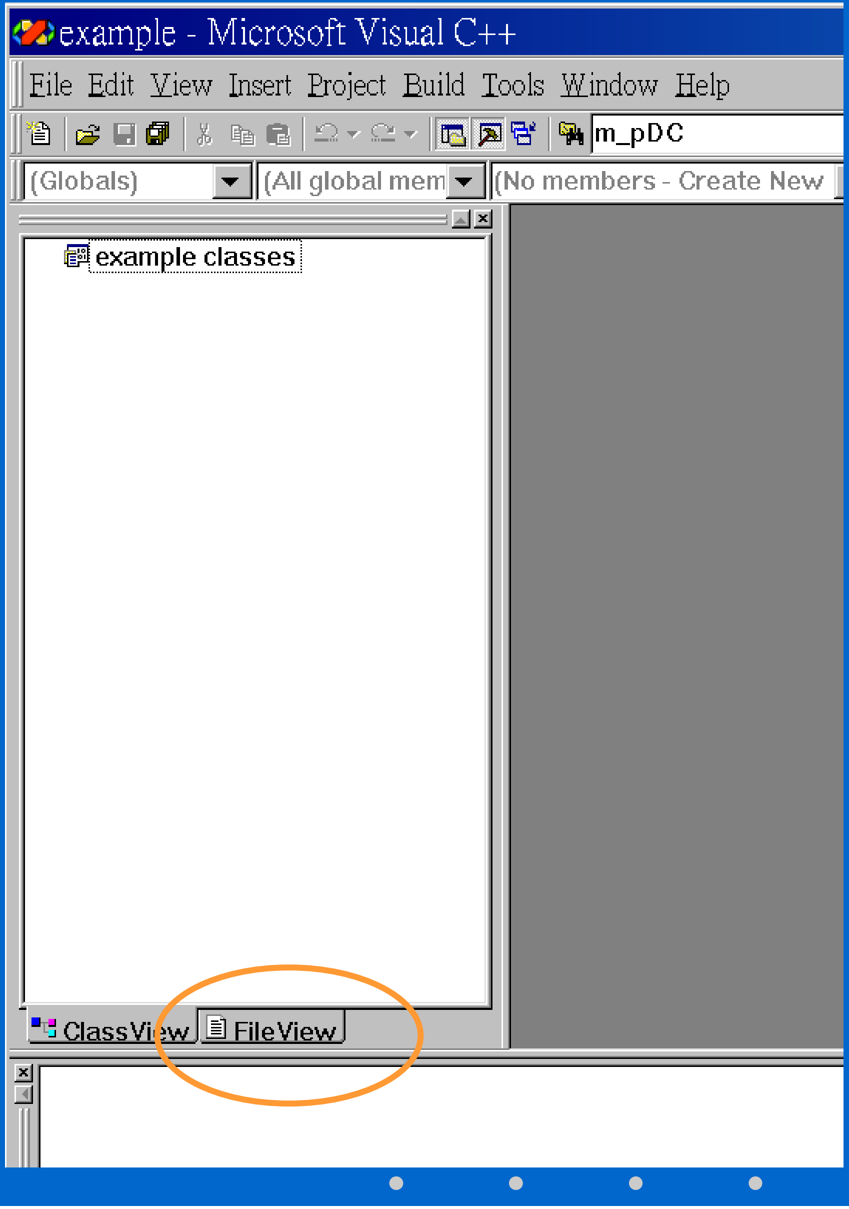
Win32 Console Application will create a new skeleton project with the following specifications:

+ Empty console application.
+ No files will be created or added to the project.

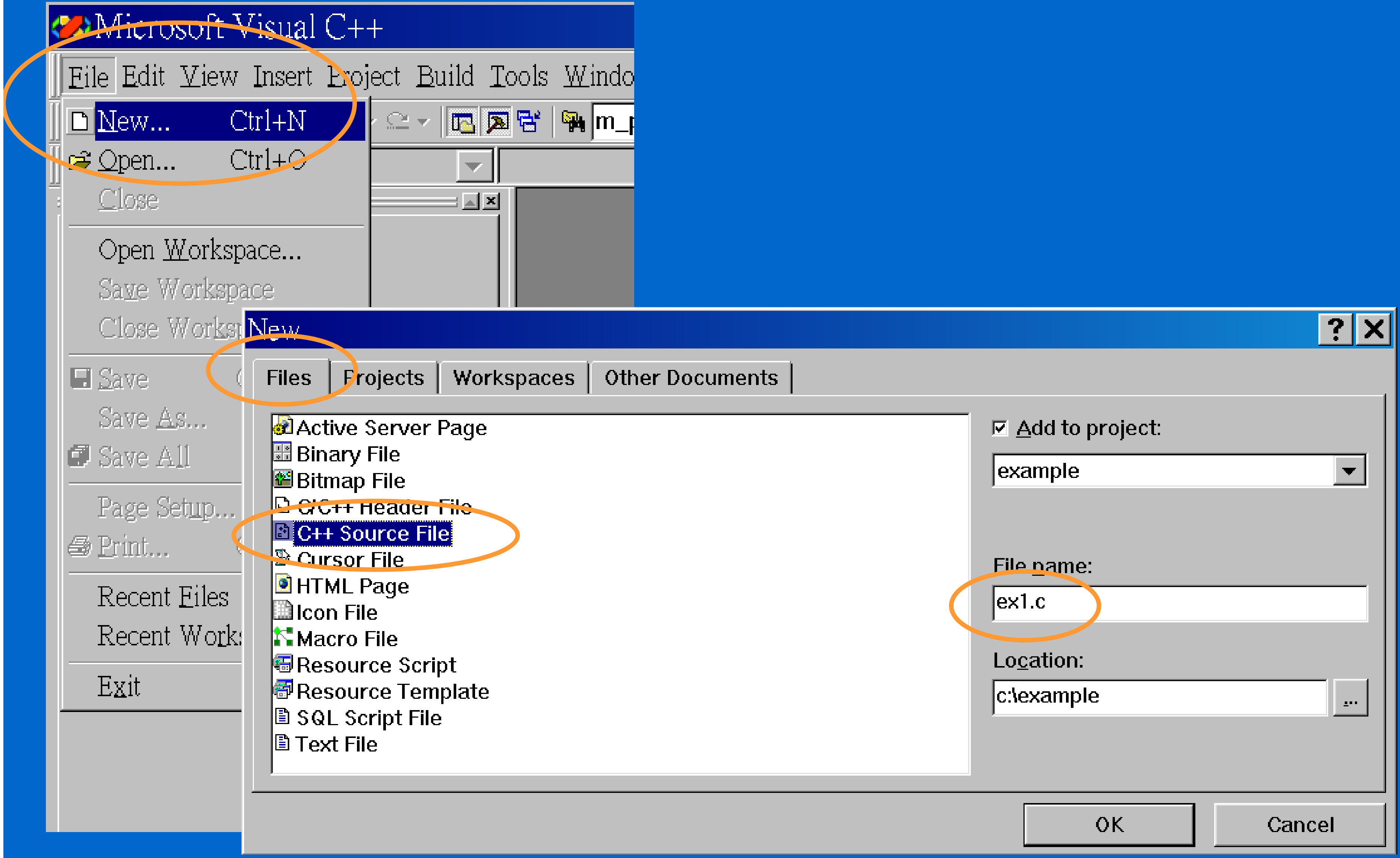
Project Directory:

c:\example

Visual C++ 6.0


example - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
m_pDC
(Globals)
(All global mem
(No members - Create New
example classes
ClassView
FileView

Visual C++ 6.0

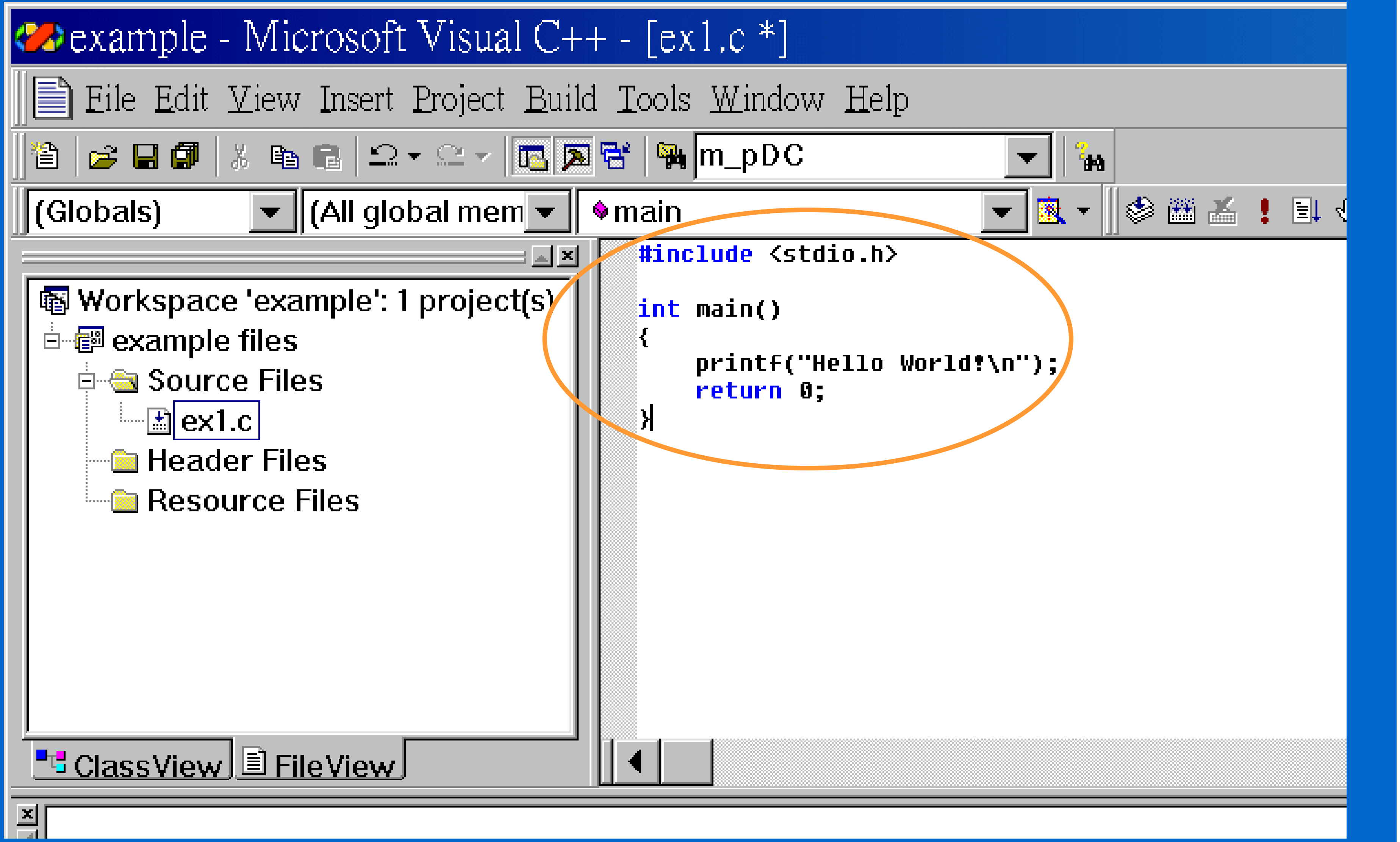


☑ Add to project: example

File name: ex1.c

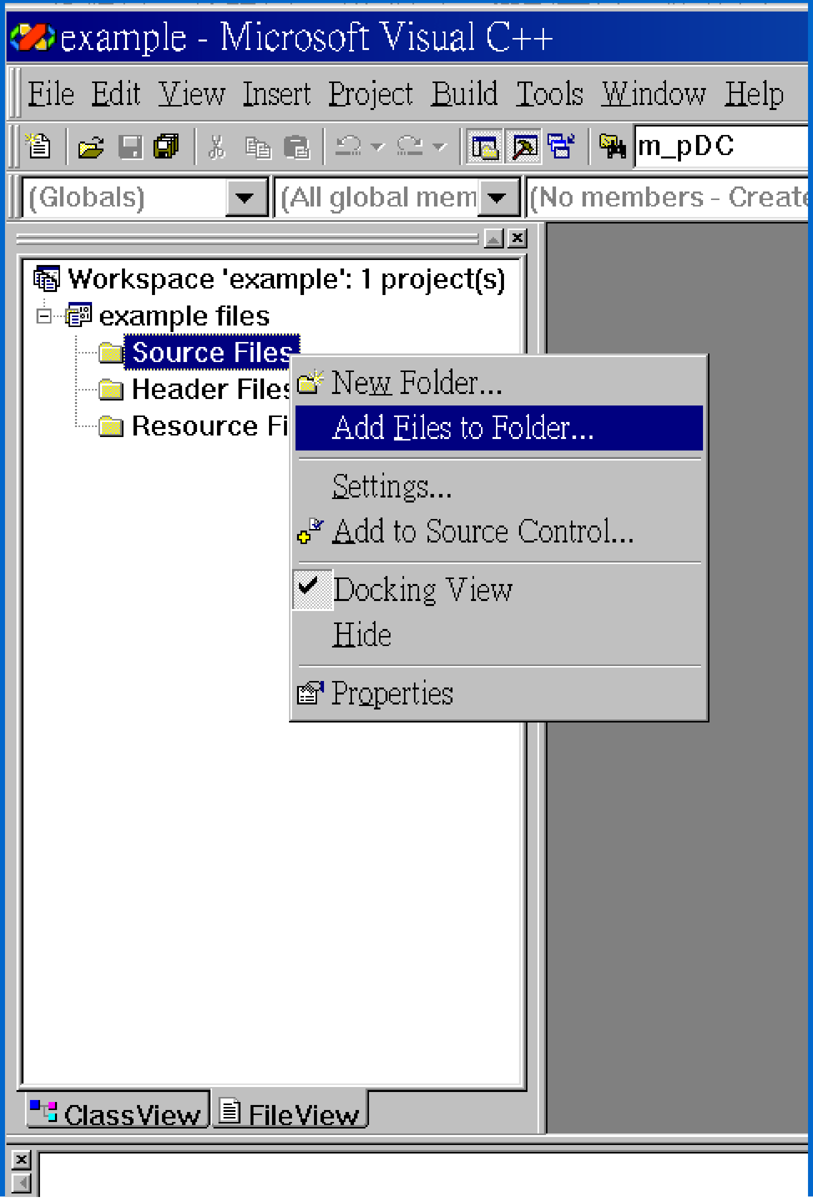
Location: c:\example

Visual C++ 6.0


example - Microsoft Visual C++ - [ex1.c *]
File Edit View Insert Project Build Tools Window Help
m_pDC
(Globals)
(All global mem
main
Workspace 'example': 1 project(s)
example files
Source Files
ex1.c
Header Files
Resource Files
ClassView
FileView

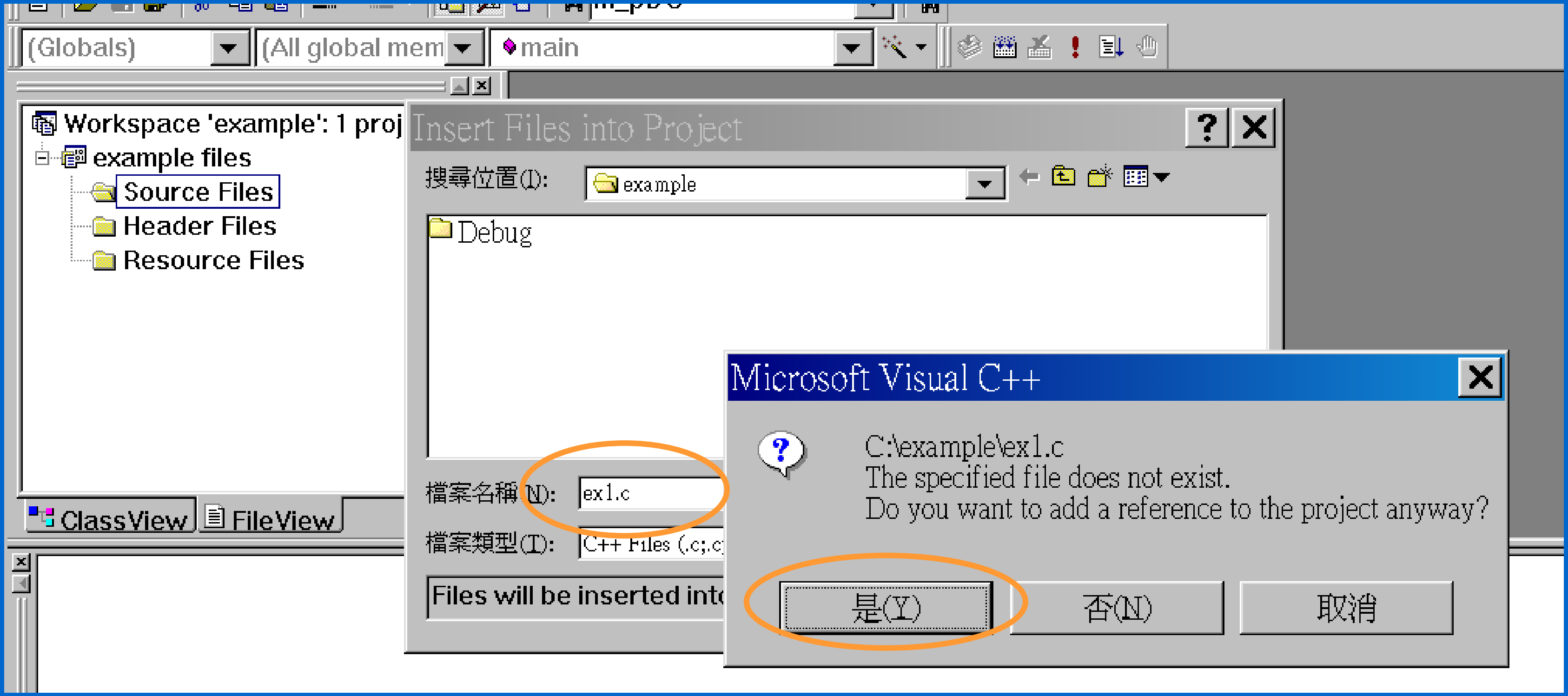
```
#include <stdio.h>

int main()
{
	printf("Hello World!\n");
	return 0;
}
```

Visual C++ 6.0



example - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
m_pDC
(Globals)
(All global mem
(No members - Create
Workspace 'example': 1 project(s)
example files
Source Files
Header Files
Resource Fi
New Folder...
Add Files to Folder...
Settings...
Add to Source Control...
Docking View
Hide
Properties
ClassView
FileView

Visual C++ 6.0



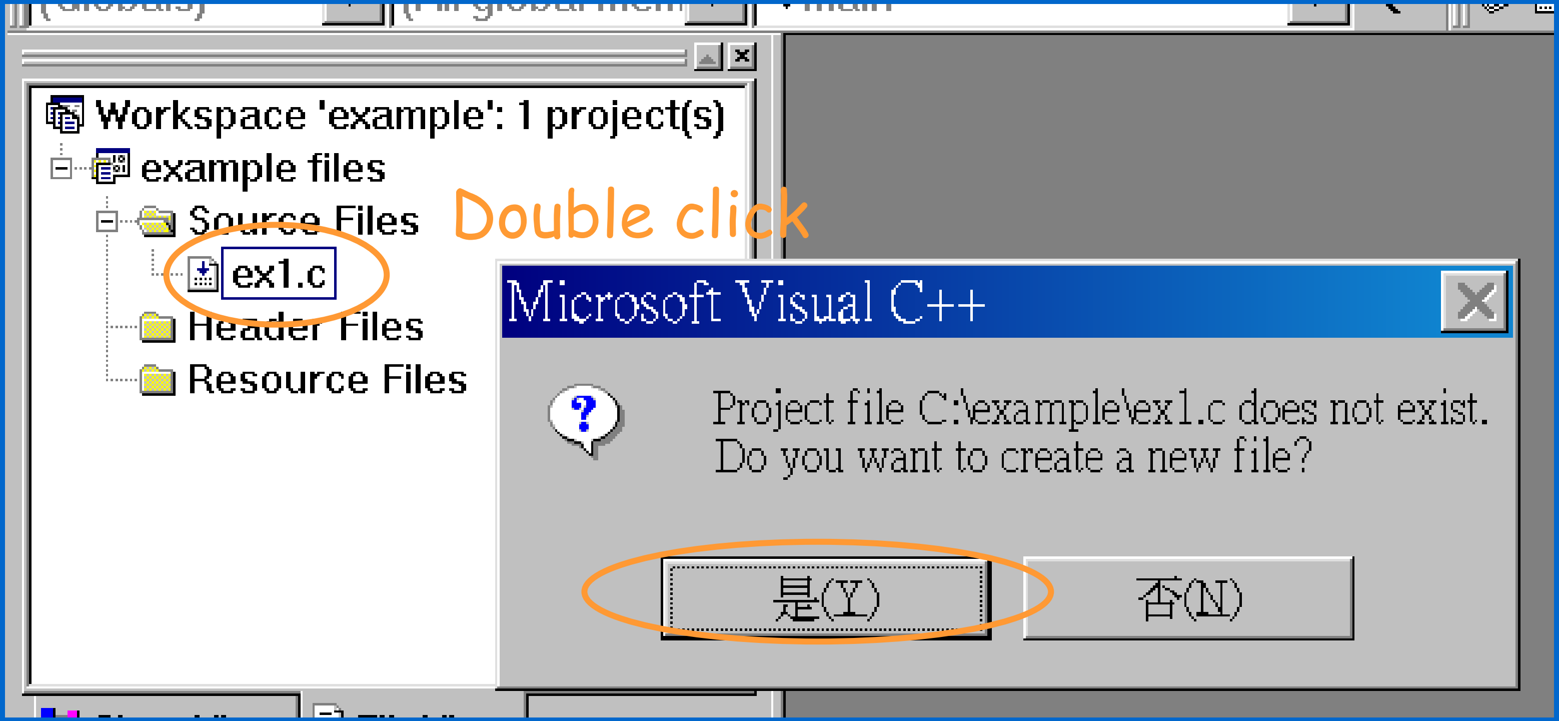
檔案名稱(N): ex1.c

Files will be inserted int

Microsoft Visual C++

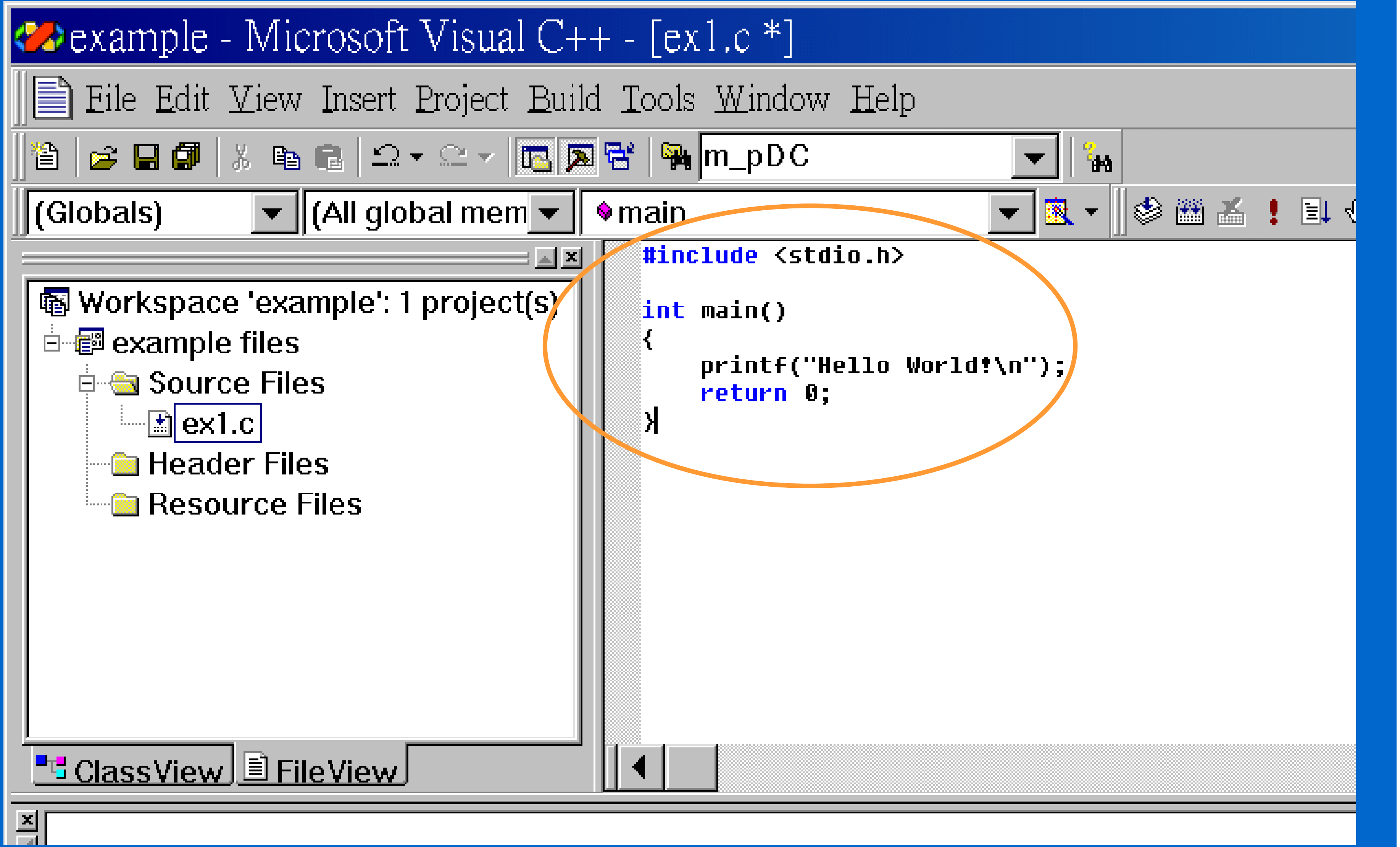
C:\example\ex1.c
The specified file does not exist.
Do you want to add a reference to the project anyway?

Visual C++ 6.0



Project file C:\example\ex1.c does not exist.
Do you want to create a new file?

Visual C++ 6.0

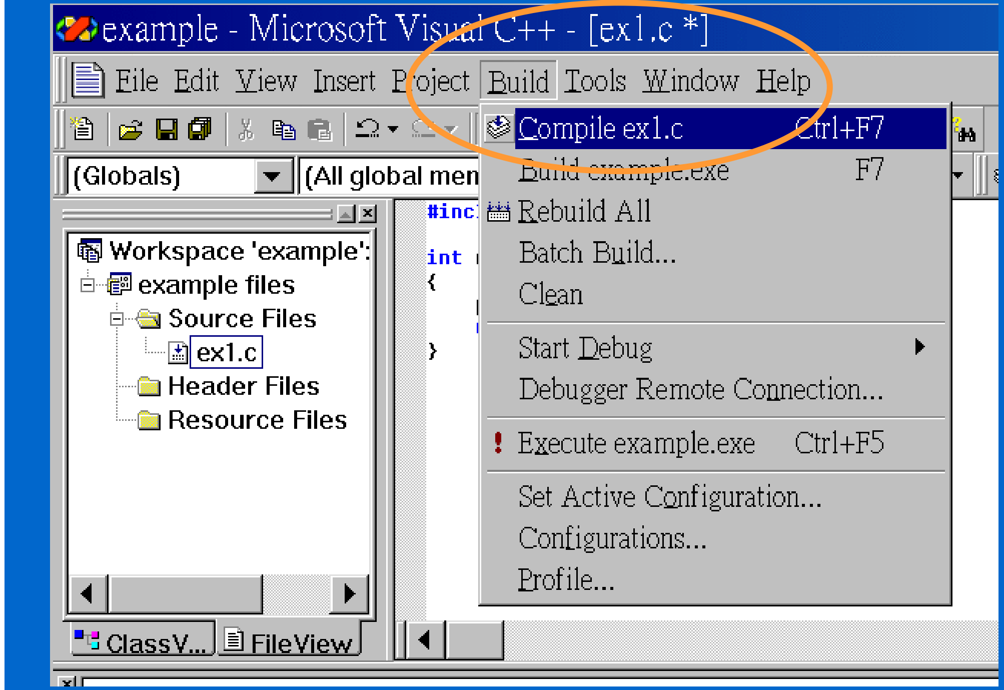


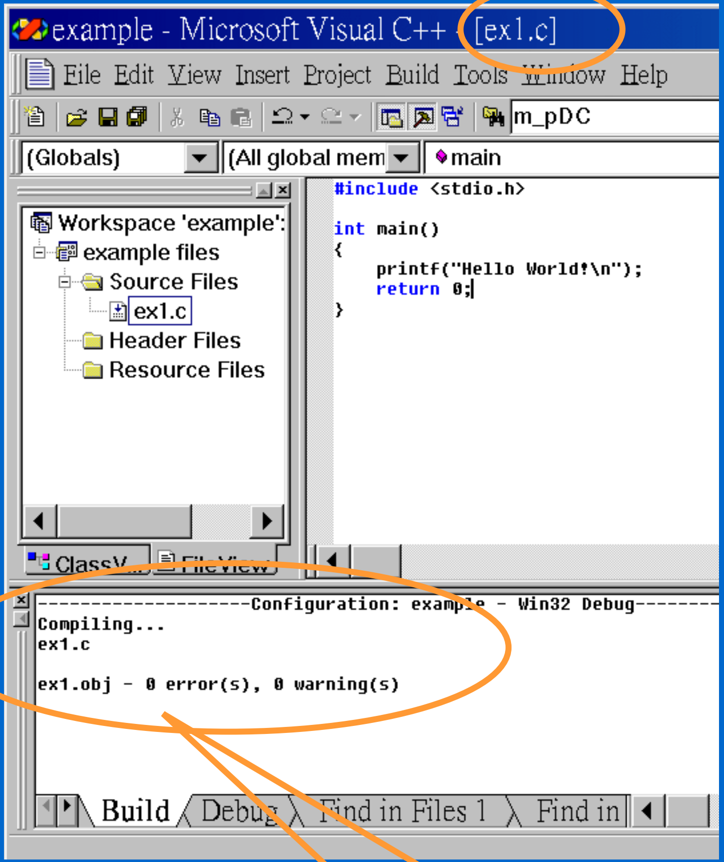
example - Microsoft Visual C++ - [ex1.c *]
File Edit View Insert Project Build Tools Window Help
m_pDC
(Globals)
(All global mem
main
Workspace 'example': 1 project(s)
example files
Source Files
ex1.c
Header Files
Resource Files
ClassView
FileView

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Visual C++ 6.0

- Compile a single source file





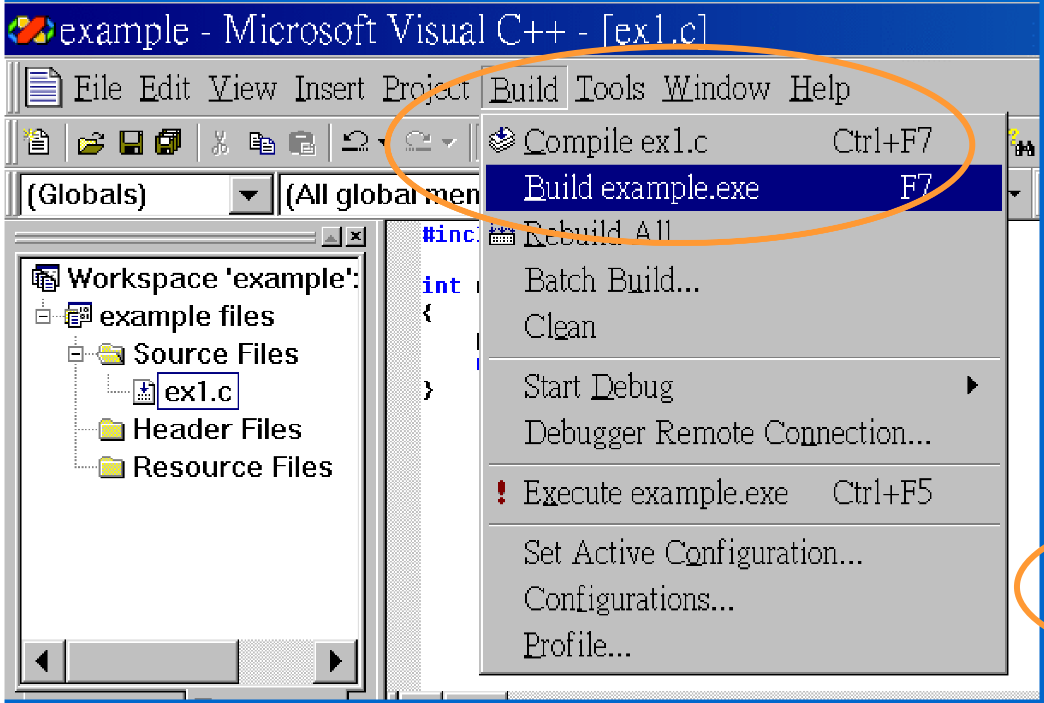
```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

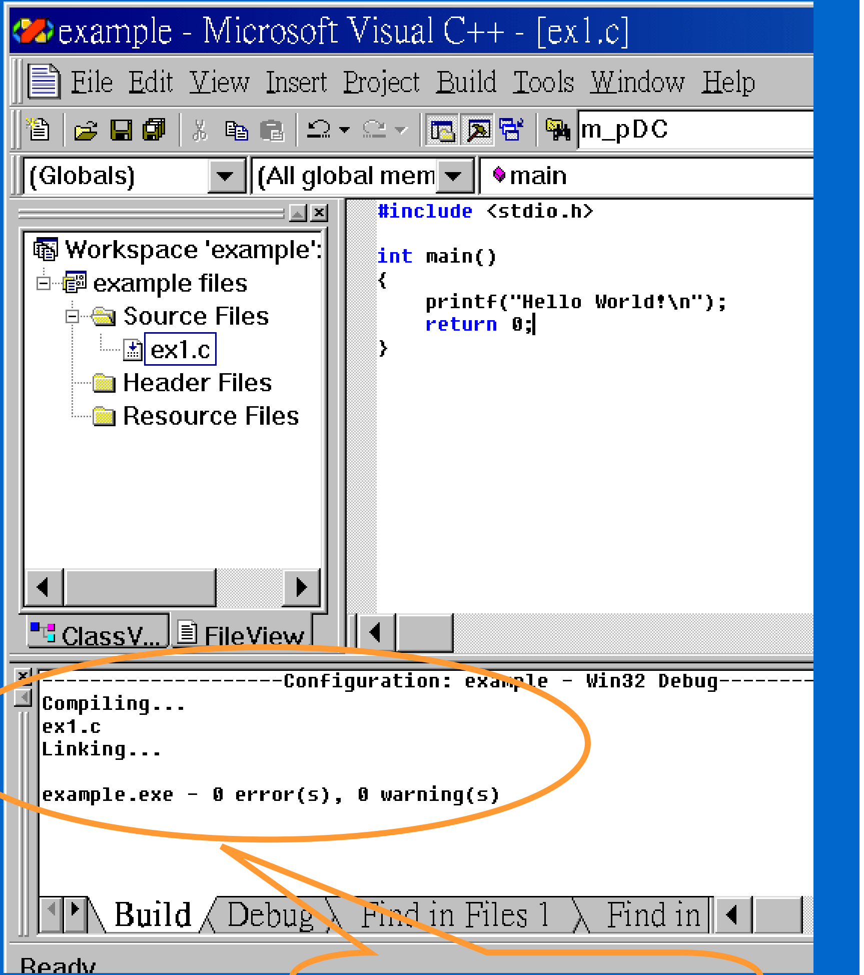
```
--------------------Configuration: example - Win32 Debug--------
Compiling...
ex1.c

ex1.obj - 0 error(s), 0 warning(s)
```

Warning and error messages if any

Visual C++ 6.0

- Build the whole project





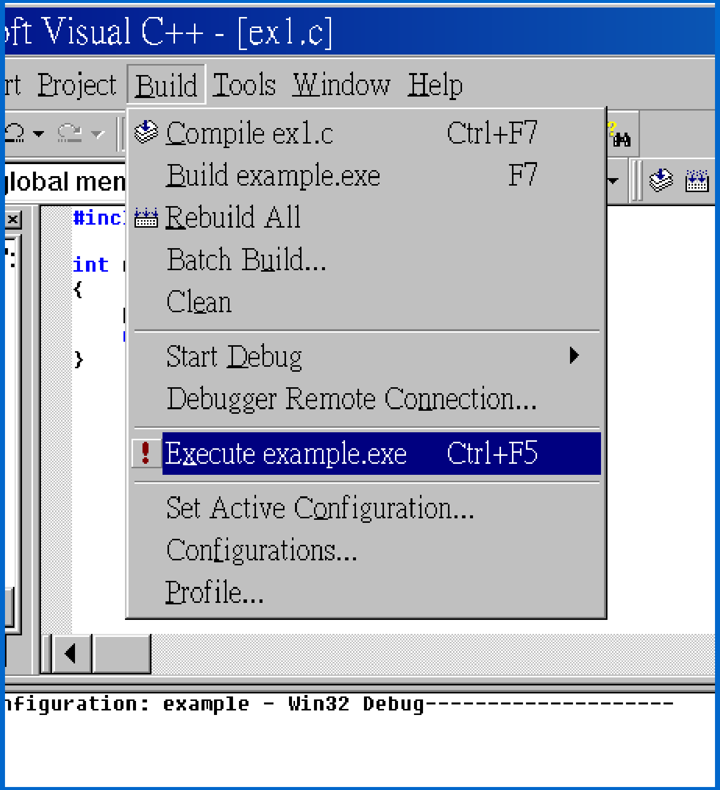
```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
```

```
--------------------Configuration: example - Win32 Debug--------------------
Compiling...
ex1.c
Linking...

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

First compile then link

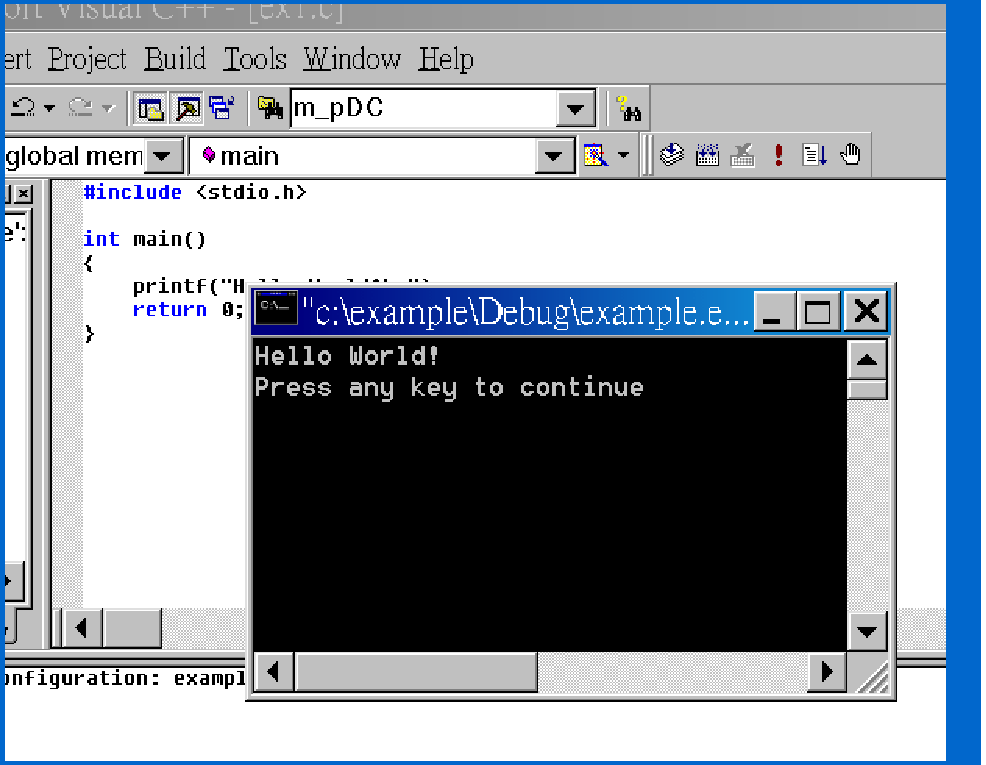
Visual C++ 6.0

- Execute



```
#inc
int
{
}
```

```
nfiguration: example - Win32 Debug--------------------
```


oft Visual C++ - [ex1.c]
ert Project Build Tools Window Help
m_pDC
global mem
main
"c:\example\Debug\example.e...

```
#include <stdio.h>

int main()
{
    printf("H
    return 0;
}
```

```
Hello World!
Press any key to continue
```

```
onfiguration: exampl
```

- .exe file is located in the “Debug” directory in debug configuration
- .exe file is located in the “Release” directory in release configuration

Visual C++ Command-Line Compiler

- Download at:
  - http://msdn.microsoft.com/visualc/vctoolkit2003/
- Install the toolkit
- Configure environment:
  - Set PATH=<the toolkit directory>\bin;%PATH%
  - Set INCLUDE=<the toolkit directory>\include;%INCLUDE%
  - Set LIB=<the toolkit directory>\lib;%LIB%

Visual C++ Command-Line Compiler

- Compile and Build

```
> cl foo.c
or
> cl foo1.c foo2.c  –OUT:foo.exe
```

- Compile

```
> cl –c foo.c
```

- Link

```
> link foo1.obj foo2.obj –OUT:foo.exe
```

Contents

- C Development Environment
- **Basic Procedural Programming Concepts**
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

Basic Programming Concepts

- Controlling the CPU+Memory+I/O to obtain your computational goals
- Memory: provides storages for your data
  - Constants: 1, 2, 'A', "a string"
  - Variables: int count;
- CPU: provides operations to data
  - Data movement: count = 1;
  - Arithmetic or Boolean expressions: 2 * 4
  - Testing and control flow: if statement, for loop, while loop, function
- I/O: FILE, stdin, stdout, printf(), scanf(), getc(), …

Programming Concepts (cont'd)

Procedural programming basics

- ✧ **Step 1**: represent your data in terms of variables
  basic types: char, int, float, double
  user defined types: struct…link lists, trees,…

(Here are what you learned in **Data Structure**)

- ✧ **Step 2**: figure out how to transform the original data to the desired result that you want to see with the primitive operations a computer provides: ex. search, sort, arithmetic or logic computations,…

(Here is what you learned in **Algorithm**)

Programming Concepts (cont’d)

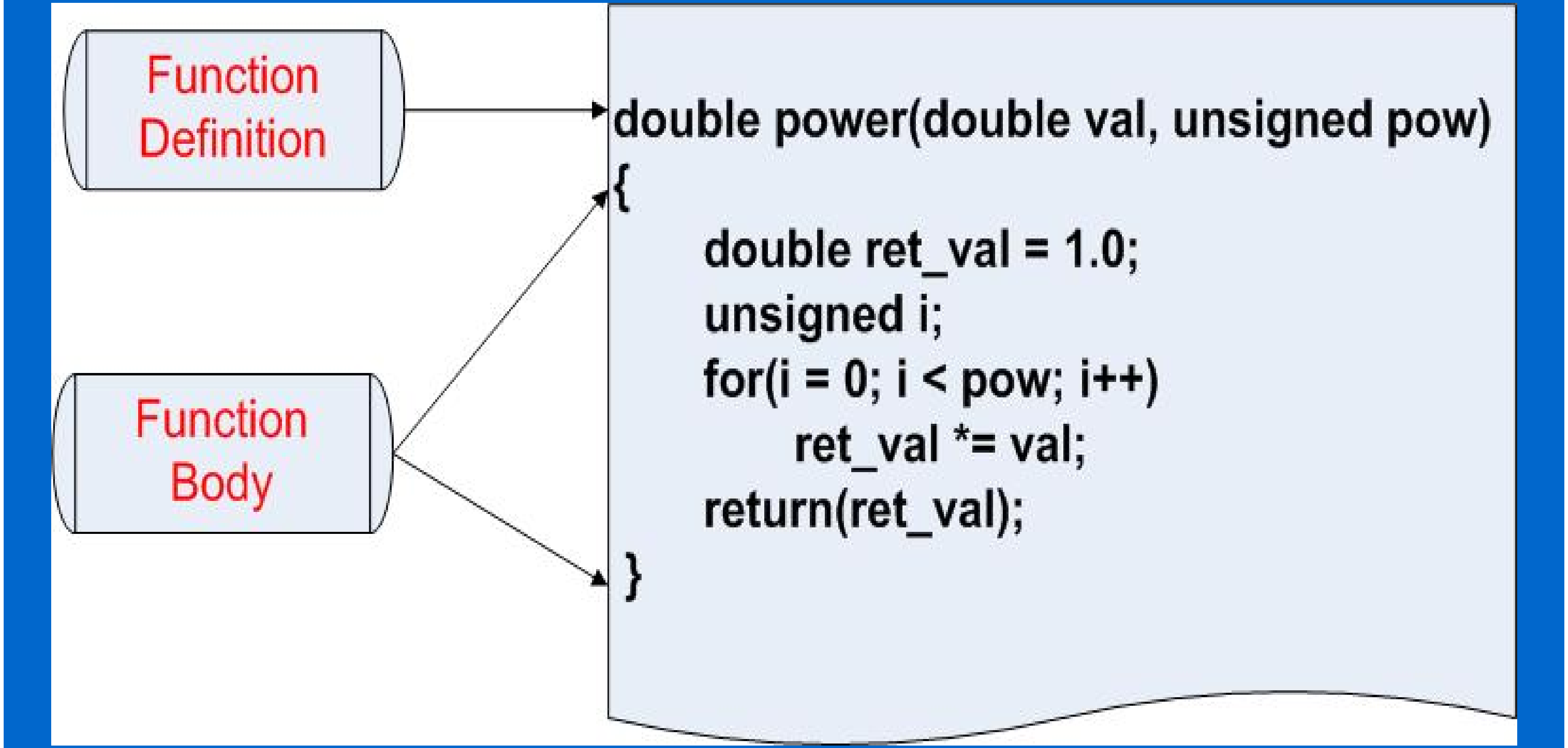
- ✧ Additional Requirements
  - ✶ **Structural Programming**: if statement, switch-case statement, iteration structure, function, block … (forbidden commands: goto, break…)
  - ✶ **Modularization**: function and file
  - ✶ **Functional testing / Unit testing**: assertion, unit testing routines, functional testing routines

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- **Functions**
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

Function Basic

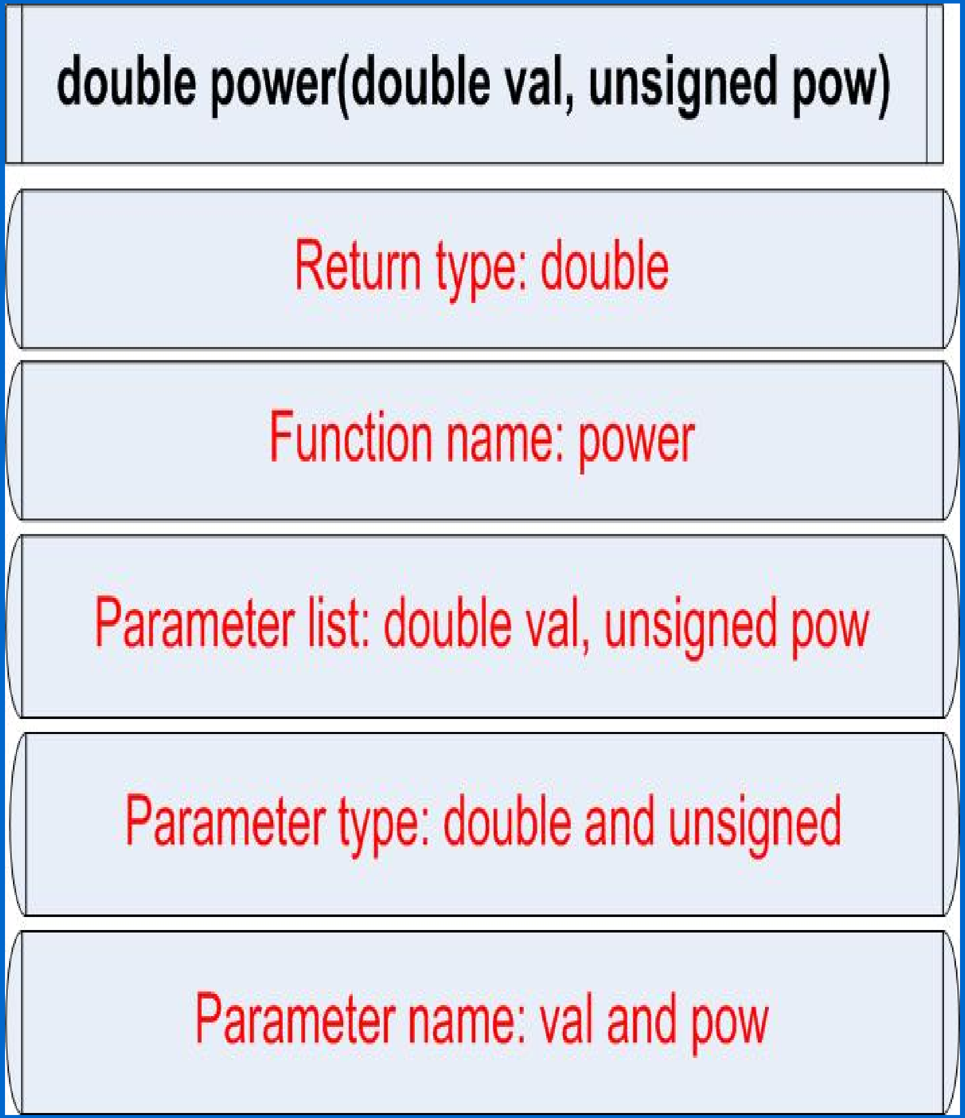
- A simple function compute the value of val^{pow}



```
double power(double val, unsigned pow)
{
        double ret_val = 1.0;
        unsigned i;
        for(i = 0; i < pow; i++)
                ret_val *= val;
        return(ret_val);
}
```

Function Definition

- The first line of the function, contains:
  - Return data type
  - Function name
  - Parameter list, for each Parameter, contains:
    - Parameter data type
    - Parameter name



double power(double val, unsigned pow)

Return type: double

Function name: power

Parameter list: double val, unsigned pow

Parameter type: double and unsigned

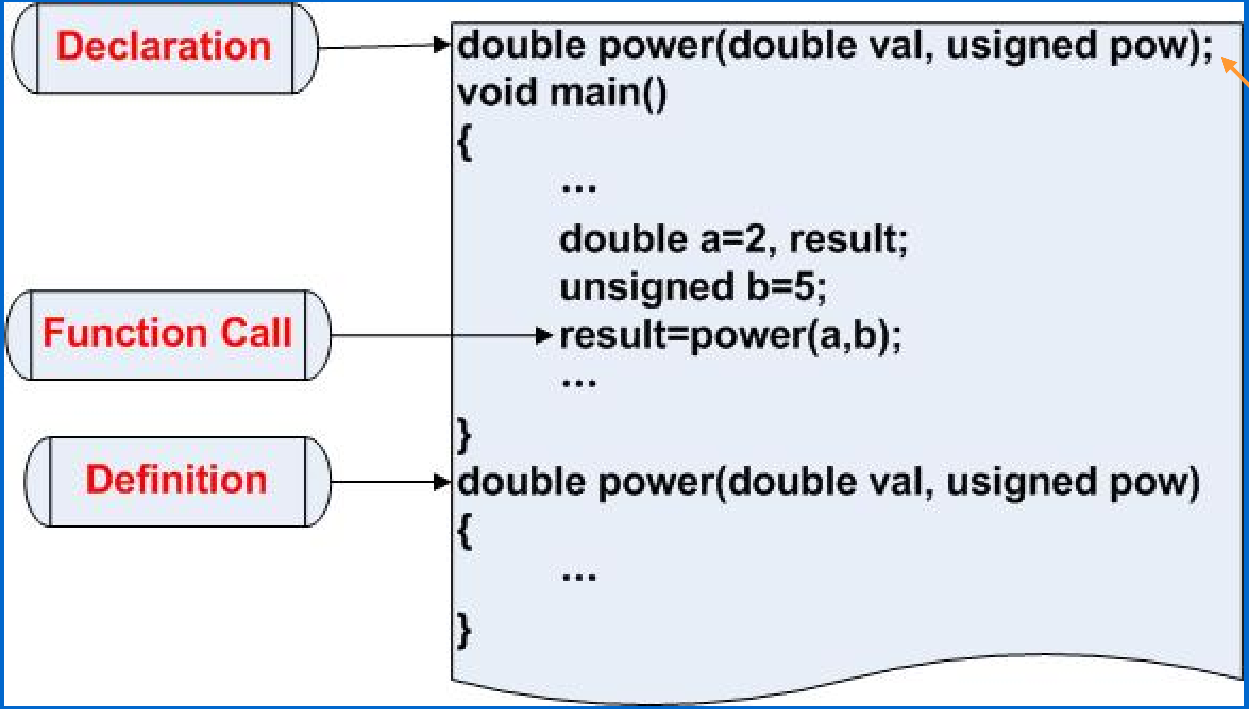
Parameter name: val and pow

Function Body

- Function Body is bounded by a set of curly brackets
- Function terminates when:
  - “return” statement is reached or
  - the final closing curly bracket is reached.
- Function returns value by:
  - “return(ret_val);” statement, the ret_val must be of the same type in function definition;
  - Return automatically when reaching the final closing curly bracket , the return value is meaningless.

Function Declaration & Function Call

- ✧ Function can be called only after it is declared, a simple skeletal program:



```
double power(double val, usigned pow);
void main()
{
        ...
        double a=2, result;
        unsigned b=5;
        result=power(a,b);
        ...
}
double power(double val, usigned pow)
{
        ...
}
```

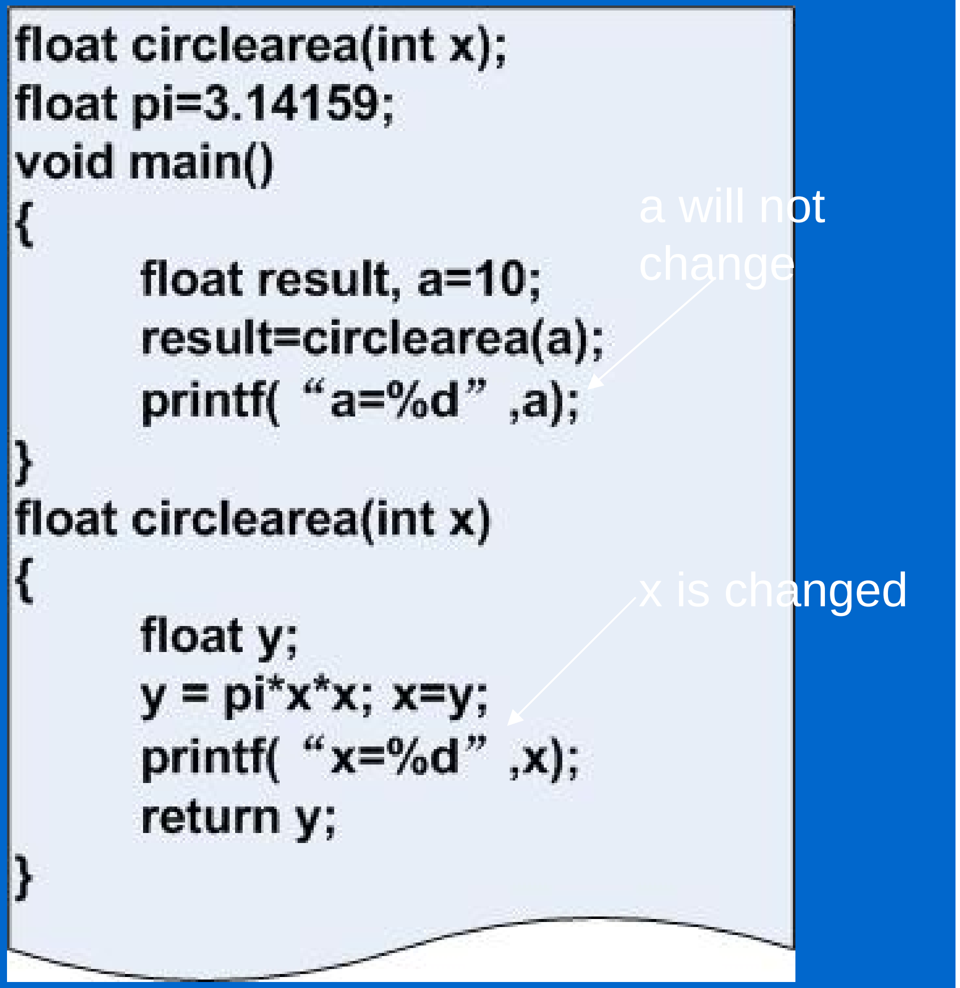
Semicolon

Function Call

- ✧ Function can be called at any part of the program after the declaration:
  - ✶ The return value of a function can be assigned to a variable of the same type.
  - ✶ Example: result = power(2, 5);
    - ✡ Compute the value of 2^5 =32 and assign the value to the variable “result”, equals to “result=32”.

Function Parameter

- C is “called by value”
  - The function receives copies of values of the parameters
  - Example:
    - Print “a=10” and “x=314.159”



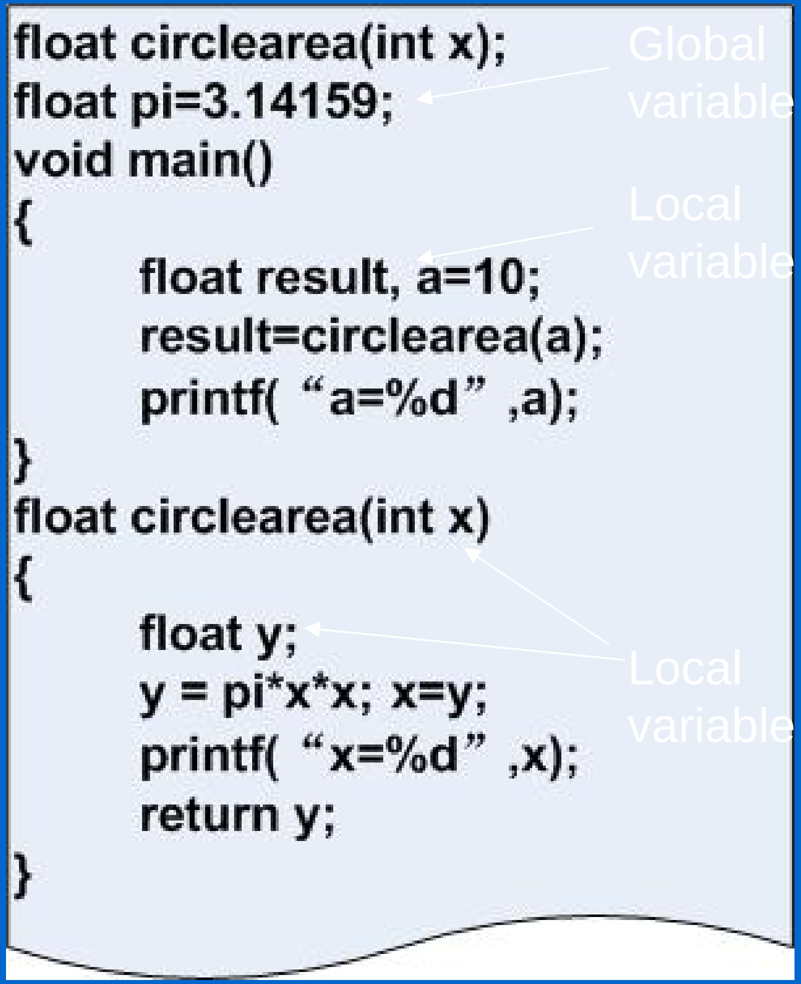
```
float circlearea(int x);
float pi=3.14159;
void main()
{
        float result, a=10;
        result=circlearea(a);
        printf( “a=%d” ,a);
}
float circlearea(int x)
{
        float y;
        y = pi*x*x; x=y;
        printf( “x=%d” ,x);
        return y;
}
```

a will not change

x is changed

Function Variable Scope

- Limited in the function
- Created each time when called
- Example,
  - pi: whole program
  - result, a: main
  - x,y: circlearea



```
float circlearea(int x);
float pi=3.14159;
void main()
{
        float result, a=10;
        result=circlearea(a);
        printf( “a=%d” ,a);
}
float circlearea(int x)
{
        float y;
        y = pi*x*x; x=y;
        printf( “x=%d” ,x);
        return y;
}
```

Global variable
Local variable
Local variable

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- **Pointers and Arrays**
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

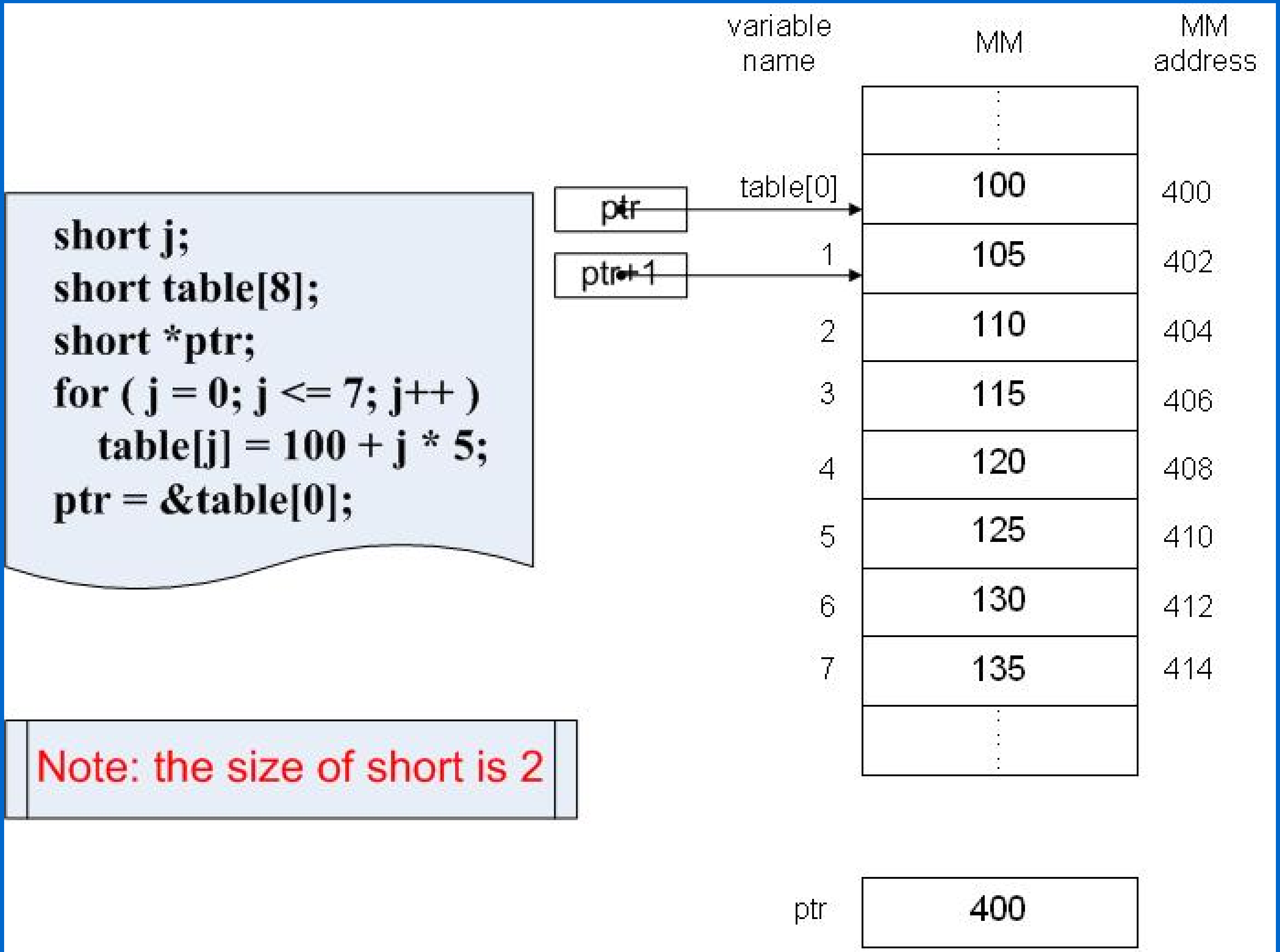
Basic Pointer Operations

- Declaration: with asterisk *.
  - int *ip; (declare a variable of integer address type)
- Generation: with “address-of” operator &.
  - int i = 5; ip = &i; (ip points to the address of i)
- Retrieve the value pointed to by a pointer using the “contents-of” (or “dereference”) operator, *.
  - printf("%d\n", *ip); (equals to “printf("%d\n", i); ”)
  - *ip=10; (equals to “i=10”)

Pointers and Arrays

- Pointers do not have to point to single variables. They can also point at the cells of an array.
  - int *ip; int a[10]; ip = &a[3];
- An array is actually a pointer to the 0-th element of the array
  - int *ip; int a[10]; ip = a; (equals to “ip = &a[0]”)
  - a[5]=10; is equivalent to *(a+5)=10;
- Pointers can be manipulated by “+” and “-”.
  - int *ip; int a[10]; ip = &a[3];
  - The pointer “ip-1” points to a[2] and “ip+3” points to a[6];

Pointers and Arrays: Example



```
short j;
short table[8];
short *ptr;
for ( j = 0; j <= 7; j++ )
    table[j] = 100 + j * 5;
ptr = &table[0];
```

Note: the size of short is 2

variable name	MM	MM address
	⋮	
table[0]	100	400
1	105	402
2	110	404
3	115	406
4	120	408
5	125	410
6	130	412
7	135	414
	⋮	

ptr	400	

Additional Information

- ✧ Pointer is a variable too, the content of a pointer is the address of the memory.
- ✧ Pointers can also form arrays, and there can be a pointer of pointer.

```
int * pt[10];
int ** ppt;     (viewed as     int * * ppt; )
ppt = &pt[0] (or ppt = pt);
```

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- **Strings**
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

String basic

- Strings in C are represented by arrays of characters.
- The end of the string is marked with the *null character*, which is simply the character with the value 0. (Also denoted as '\0');
- The string literals:
  - char string[] = "Hello, world!";
  - we can leave out the dimension of the array, the compiler can compute it for us based on the size of the initializer (including the terminating \0).

Note:

```
char string[];                      is illegal
string = "Hello, world!";           is illegal
```

String handling

- Standard library <string.h>
- For details, please refer to manual: such as MSDN

strcat,strncat	Append string
strchr,strrchr	Find character in string
strcpy,strncpy	Copy string
strcmp, strncmp	Compare string
strlen	Return string length
strstr	Find substring

A Review of C Language

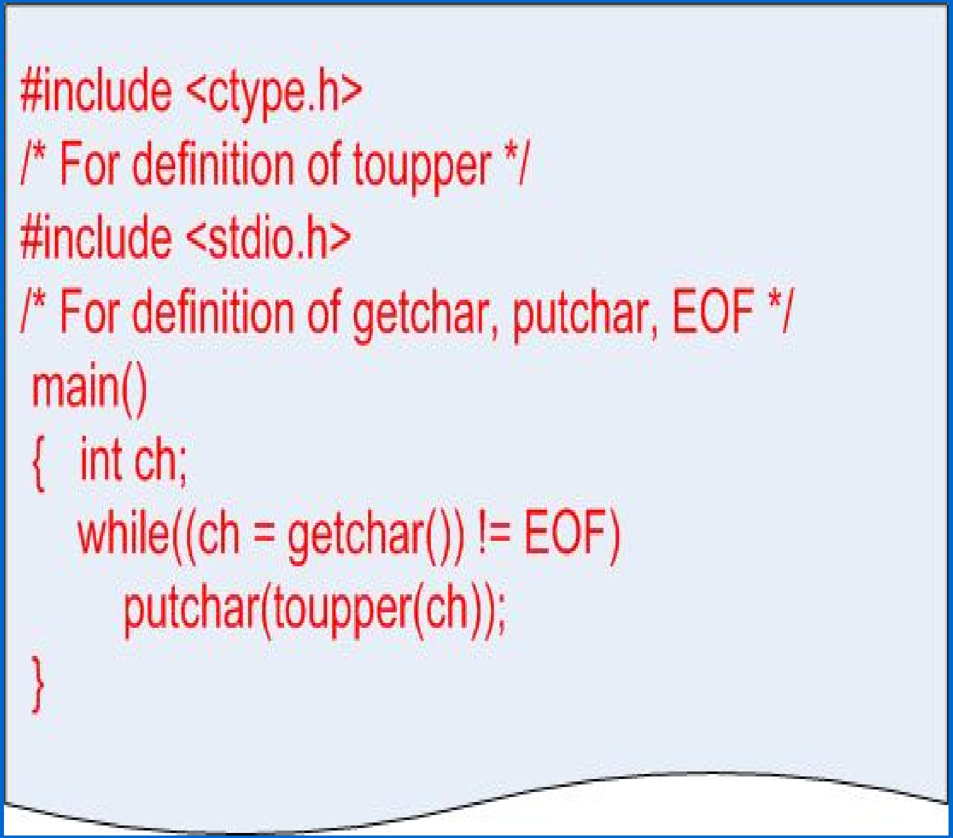
- C Development Environment
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- Reading the Command Line

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- Strings
- **Basic I/O**
- Memory Allocation
- File Operation
- Reading the Command Line

Char I/O

- “getchar”: getchar returns the next character of keyboard input as an int.
- “putchar”: putchar puts its character argument on the standard output (usually the screen).



```
#include <ctype.h>
/* For definition of toupper */
#include <stdio.h>
/* For definition of getchar, putchar, EOF */
 main()
 {   int ch;
     while((ch = getchar()) != EOF)
         putchar(toupper(ch));
 }
```

String I/O

- “printf”: Generates output under the control of a *format string*
- “scanf”: Allows *formatted reading* of data from the keyboard.

Format Specification

- Basic *format specifiers* for printf and scanf:
  - %d print an int argument in decimal
  - %ld print a long int argument in decimal
  - %c print a character
  - %s print a string
  - %f print a float or double argument
  - %o print an int argument in octal (base 8)
  - %x print an int argument in hexadecimal (base 16)

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- **Memory Allocation**
- File Operation
- Reading the Command Line

Allocating Memory with “malloc”

- Is declared in <stdlib.h>
  - void *malloc(size_t <u>size</u>);
- Returns a pointer to *n* bytes of memory
  - *char *line = (char *)malloc(100);*
- Can be of any type;
  - Assume “date” is a complex structure;
  - *struct date *today =*
    *(struct date *)malloc(sizeof(struct date));*
- Return null if failed

Freeing Memory

- ✧ Memory allocated with *malloc* lasts as long as you want it to.
- ✧ It does not automatically disappear when a function returns, but remain for the entire duration of your program.
- ✧ Dynamically allocated memory is deallocated with the *free* function.
  - ✶ *free(line); free(today);*
  - ✶ fail if the pointer is null or invalid value

Reallocating Memory Blocks

- Reallocate memory to a pointer which has been allocated memory before (maybe by *malloc*)
  - void *realloc(void *memblock, size_t size);
  - *today_and_tomorrow = realloc(today, 2*sizeof(date));*

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- **File Operation**
- Reading the Command Line

File Pointers

- C communicates with files using a extended data type called a file pointer.
  - FILE *output_file;
- Common file descriptors:
  - “stdin”: The standard input. The keyboard or a redirected input file.
  - “stdout”: The standard output. The screen or a redirected output file.
  - “stderr”: The standard error. The screen or a redirected output file.

Open and Close

- Using *fopen* function, which opens a file (if exist) and returned a file pointer
  - fopen("output_file", "w");
- Using *fclose* function, which disconnect a file pointer from a file
- Access character:
  - “r”: open for reading;
  - “w”: open for writing;
  - “a”: open for appending.

File I/O

- Standard library <stdio.h>
- For details, please refer to manual: such as MSDN

putchar, putc	Put a character to a file
getchar, getc	Get a character from a file
fprintf	Put formatted string into a file.
fscanf	Take data from a string of a file.
fputs	Put a string into a file
fgets	Get a string from a file

Contents

- C Development Environment
- Basic Procedural Programming Concepts
- Functions
- Pointers and Arrays
- Strings
- Basic I/O
- Memory Allocation
- File Operation
- **Reading the Command Line**

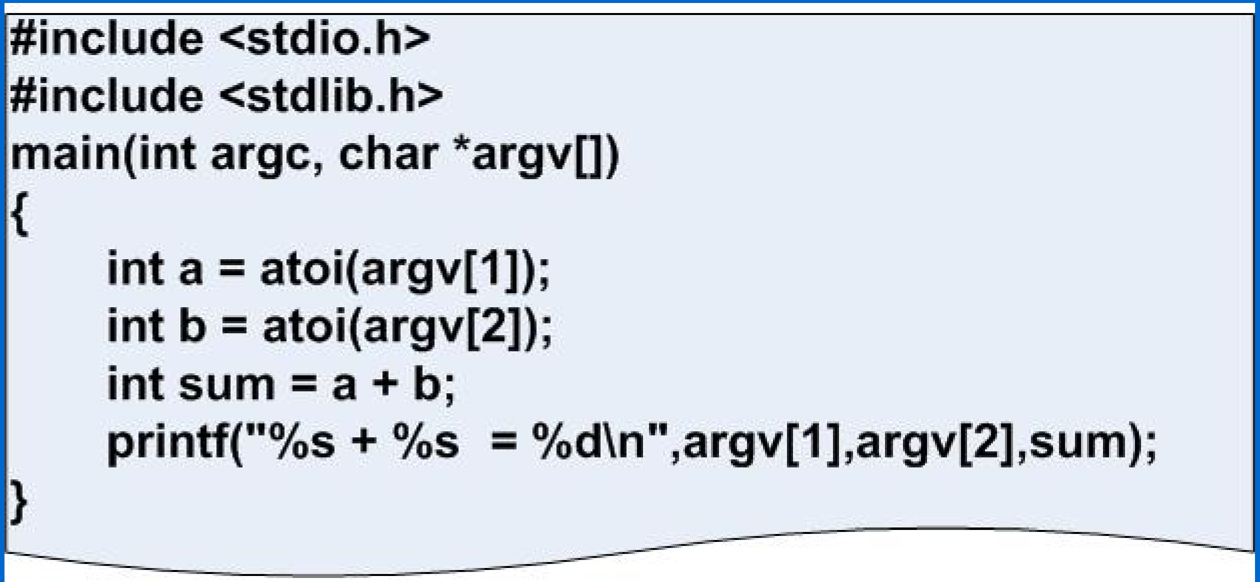
Input From the Command Line

- C's model of the command line of a sequence of words, typically separated by whitespace.
- A program with command arguments:
  - int main(int argc, char *argv[]) { ... }
  - “argc” is a count of the number of command-line arguments.
  - “argv” is an array (“vector”) of the arguments themselves.

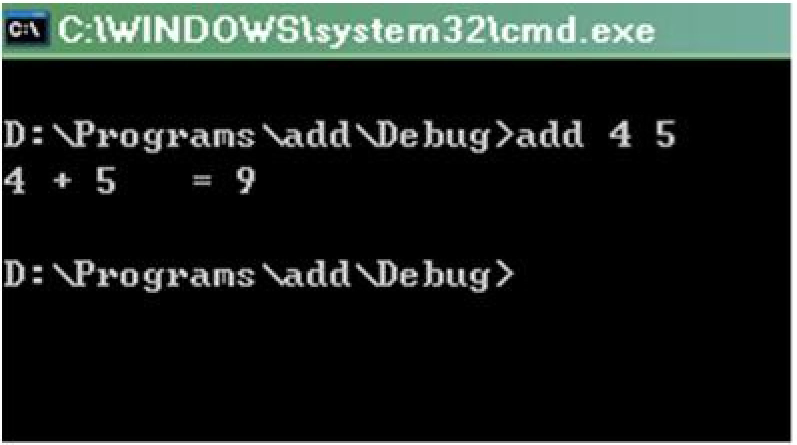
Ex.

```
sort file1 file2 file3
```

Example



```
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
        int a = atoi(argv[1]);
        int b = atoi(argv[2]);
        int sum = a + b;
        printf("%s + %s  = %d\n",argv[1],argv[2],sum);
}
```



```
D:\Programs\add\Debug>add 4 5
4 + 5    = 9

D:\Programs\add\Debug>
```

argc = 3

argv[0] = “add”

argv[1] = “4”

argv[2] = “5”