1. Using Function 1

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **Function1.** Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// main.cpp
#include <iostream>
using namespace std;

/* define some external functions */
extern int func_a();
extern int func_b();

void main()
{
  int a = func_a();     // call function func_a() and return the value to a
  int b = func_b();     // call function func_b() and return the value to b
  cout << "Have FUN!!" << endl;
  cout << "a: " << a << " b: " << b << endl;
}
```

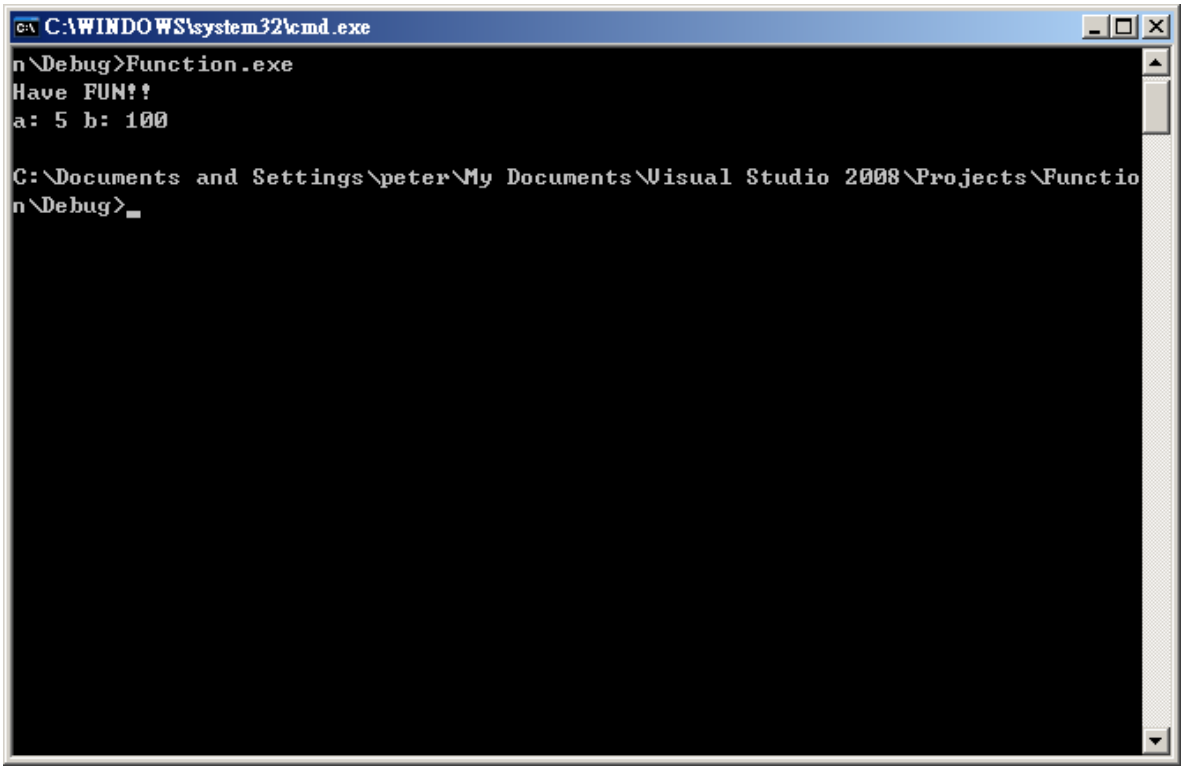
3. Create a **C++ File (.cpp)** and name it as **a.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside..

```
// a.cpp
int func_a()
{
  return 5;
}
```

4. Create a **C++ File (.cpp)** and name it as **b.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside..

```
// b.cpp
int func_b()
{
  return 10*10;
}
```

5. Use **Build Solution** to compile and build the solution, and then execute it.



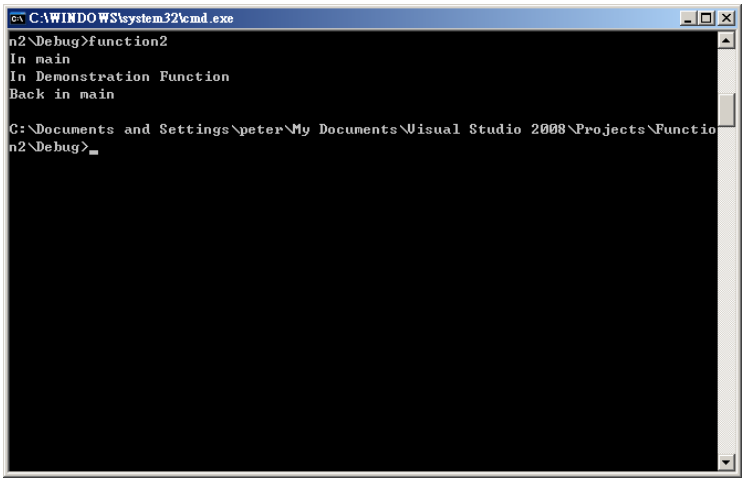
2. Using Function 2

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **Function2**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// Program demonstrating function call
#include <iostream>
using namespace std;

// function DemonstrationFunction: prints out a useful message
void DemonstrationFunction() {
        cout << "In Demonstration Function\n";
}

// function main - prints out a message, then calls DemonstrationFunction,
// then prints out a second message.
void main()
{
       cout << "In main\n" ;
       DemonstrationFunction();
       cout << "Back in main\n";
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



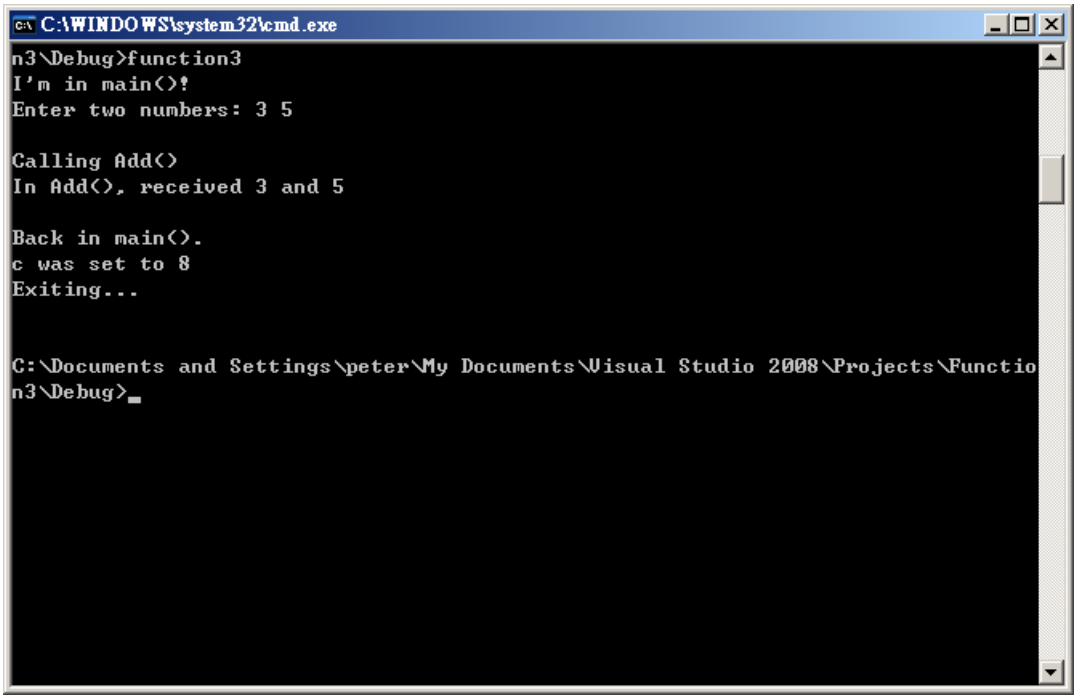
3. Using Function 3

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **Function3**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// Program demonstrating function call
#include <iostream>
using namespace std;


int Add(int a, int b);// function prototype(definition)

void main() {
        int a, b, c;

        cout << "I'm in main()!\n";
        cout << "Enter two numbers: ";
        cin >> a;  // Get the user input to variable  “a” ;
        cin >> b;  // Get the user input to variable  “b” ;

        cout << "\nCalling Add()\n";

        c = Add(a,b);  // Call the function  “Add()”
        cout << "\nBack in main().\n";
        cout << "c was set to " << c;
        cout << "\nExiting...\n\n";
}

int Add (int x, int y)
{
      cout << "In Add(), received " << x << " and " << y << "\n";
      return (x+y);
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it. What happens if you delete the line "***int Add(int a, int b);***"? Try it!



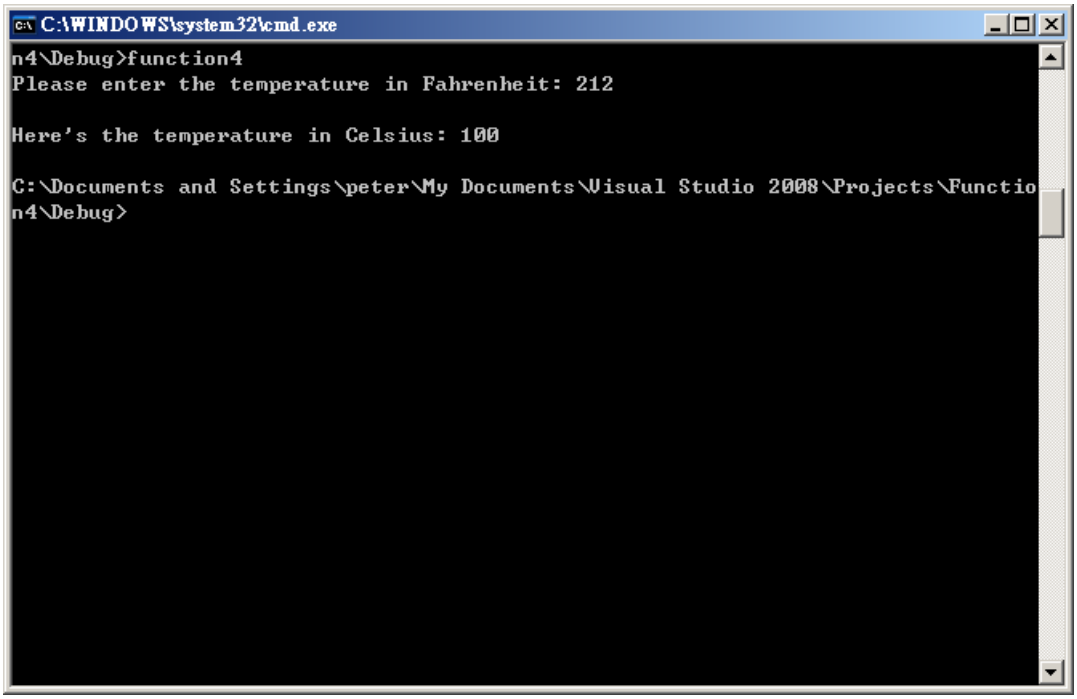
4. Using Function 4

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **Function4**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// Demonstration of use of local variables and parameters.
#include <iostream>
using namespace std;

float Convert(float);  // function definition

void main() {
        float temp_fer;
        float temp_cel;

        cout << "Please enter the temperature in Fahrenheit: ";
        cin >> temp_fer;

        temp_cel = Convert(temp_fer);
        cout << "\nHere's the temperature in Celsius: ";
        cout << temp_cel << endl;
}

float Convert(float temp_fer) {
        float temp_cel;
        temp_cel = ((temp_fer - 32) * 5) / 9;
        return temp_cel;
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



4. Program Explanation

- On lines 8 and 9, two float variables are declared, one to hold the temperature in Fahrenheit and one to hold the temperature in degrees Celsius. The user is prompted to enter a Fahrenheit temperature on line 11, and that value is passed to the function Convert(). Execution jumps to the first line of the function Convert() on line 18, where a local variable, also named temp_cel, is declared. Note that this local variable is not the same as the variable temp_cel on line 9. This variable exists only within the function Convert(). The value passed as a parameter, temp_fer, is also just a local copy of the variable passed in by main().
- This function could have named the parameter FerTemp and the local variable CelTemp, and the program would work equally well. You can enter these names again and recompile the program to see this work.
- The local function variable temp_cel is assigned the value that results from subtracting 32 from the parameter temp_fer, multiplying by 5, and then dividing by 9. This value is then returned as the return value of the function, and on line 13 it is assigned to the variable temp_cel in the main() function. The value is printed on line 15. The program is run three times. The first time, the value 212 is passed in to ensure that the boiling point of water in degrees Fahrenheit (212) generates the correct answer in degrees Celsius (100). The second test is the freezing point of water. The third test is a random number chosen to generate a fractional result.

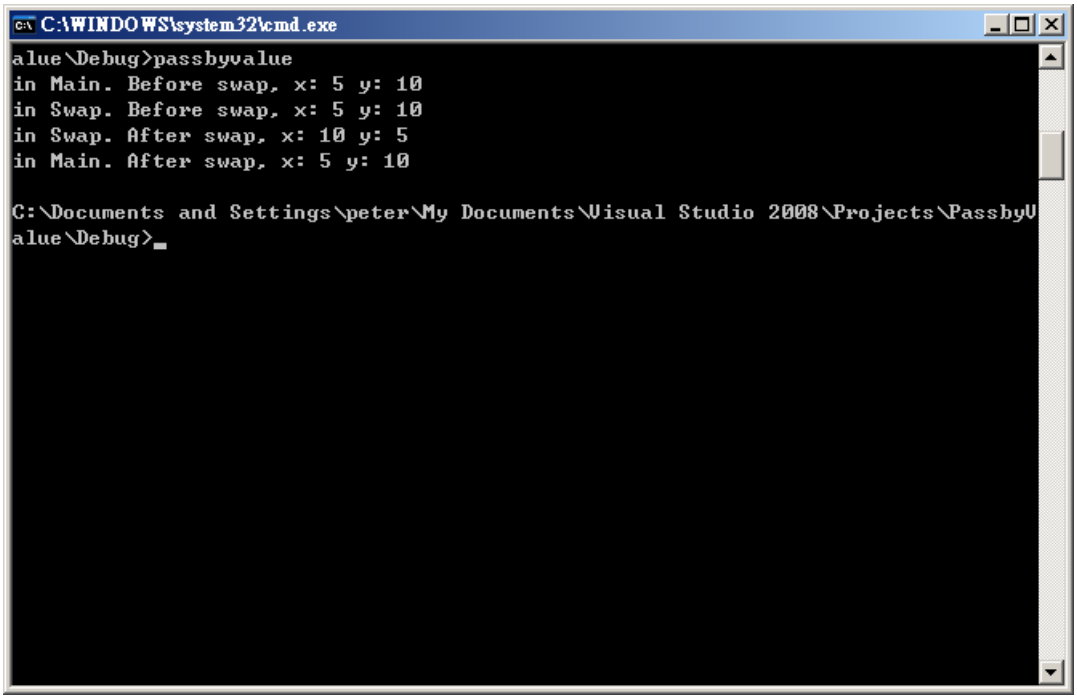
5. Demonstrates Passing by Value

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **PassByValue**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// demonstrates passing by value
#include <iostream>
using namespace std;

void swap(int x, int y); // function prototype

void main()
{
    int x = 5, y = 10;

    cout << "in Main. Before swap, x: " << x << " y: " << y << "\n";
    swap(x,y);
    cout << "in Main. After swap, x: " << x << " y: " << y << "\n";

}

void swap (int x, int y)
{
    int temp;

    cout << "in Swap. Before swap, x: " << x << " y: " << y << "\n";

    temp = x;
    x = y;
    y = temp;

    cout << "in Swap. After swap, x: " << x << " y: " << y << "\n";

}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



4. Program Explanation
   - This program initializes two variables in main() and then passes them to the swap() function, which appears to swap them. When they are examined again in main(), however, they are unchanged! The variables are initialized on line 9, and their values are displayed on line 11. swap() is called, and the variables are passed in.
   - Execution of the program switches to the swap() function, where on line 21 the values are printed again. They are in the same order as they were in main(), as expected. On lines 23 to 25 the values are swapped, and this action is confirmed by the printout on line 27. Indeed, while in the swap() function, the values are swapped.
   - Execution then returns to line 13, back in main(), where the values are no longer swapped.
   - As you've figured out, the values passed in to the swap() function are passed by value, meaning that copies of the values are made that are local to swap(). These local variables are swapped in lines 23 to 25, but the variables back in main() are unaffected.

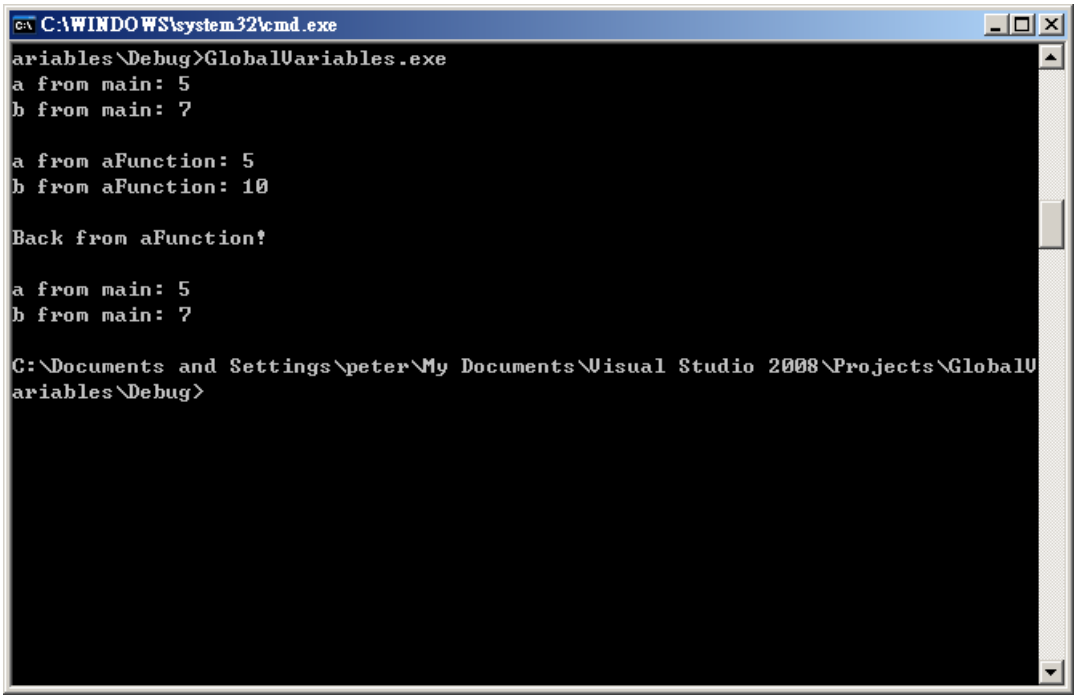
6. Global Variables

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **GlobalVariables**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// Demo of Global Variables
#include <iostream>
using namespace std;

void aFunction();           // prototype

int a = 5, b = 7;            // global variables

void main()
{
	cout << "a from main: " << a << "\n";
	cout << "b from main: " << b << "\n\n";

	aFunction();

	cout << "Back from aFunction!\n\n";
	cout << "a from main: " << a << "\n";
	cout << "b from main: " << b << "\n";
}

void aFunction()
{
	int b = 10;

	cout << "a from aFunction: " << a << "\n";
	cout << "b from aFunction: " << b << "\n\n";
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



4. Program Explanation

- This simple program illustrates a few key, and potentially confusing, points about local and global variables. On line 7, two global variables, a and b, are declared. The global variable a is initialized with the value 7, and the global variable b is initialized with the value 7.
- On lines 11 and 12 in the function main(), these values are printed to the screen. Note that the function main() defines neither variable; because they are global, they are already available to main().
- When aFunction() is called on line 14, program execution passes to line 21, and a local variable, b, is defined and initialized with the value 10. On line 23, aFunction() prints the value of the variable a, and the global variable a is used, just as it was in main(). On line 26, however, when the variable name b is used, the local variable b is used, hiding the global variable with the same name.
- The function call ends, and control returns to main(), which again prints the values in the global variables. Note that the global variable b was totally unaffected by the value assigned to aFunction()'s local b variable.

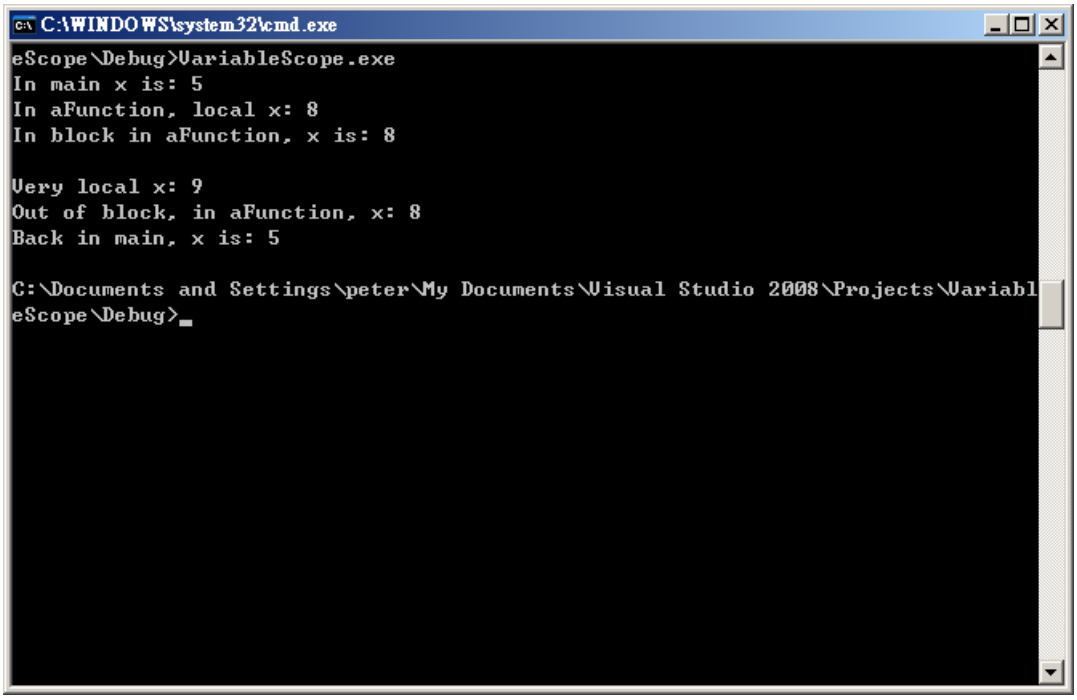
7. Variables Scope

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **VariablesScope**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// demonstrates variables scoped within a block
#include <iostream>
using namespace std;

void aFunction();


void main()
{
    int x = 5;
    cout << "In main x is: " << x << endl;

    aFunction();

    cout << "Back in main, x is: " << x  << endl;

}

void aFunction()
{

    int x = 8;
    cout << "In aFunction, local x: " << x << endl;

    {
        cout << "In block in aFunction, x is: " << x  << endl;

        int x = 9;

        cout << "\nVery local x: " << x;
    }

    cout << "\nOut of block, in aFunction, x: " << x << endl;
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



4. Program Explanation
   - This program begins with the initialization of a local variable, x, on line 10, in main(). The printout on line 11 verifies that x was initialized with the value 5. aFunction() is called, and a local variable, also named x, is initialized with the value 8 on line 22. Its value is printed on line 23.
   - A block is started on line 25, and the variable x from the function is printed again on line 26. A new variable also named x, but local to the block, is created on line 28 and initialized with the value 9.
   - The value of the newest variable x is printed on line 30. The local block ends on line 31, and the variable created on line 28 goes "out of scope" and is no longer visible.
   - When x is printed on line 33, it is the x that was declared on line 22. This x was unaffected by the x that was defined on line 28; its value is still 8.
   - On line 34, aFunction() goes out of scope, and its local variable x becomes unavailable. Execution returns to line 15, and the value of the local variable x, which was created on line 10, is printed. It was unaffected by either of the variables defined in aFunction().
   - Needless to say, this program would be far less confusing if these three variables were given unique names!

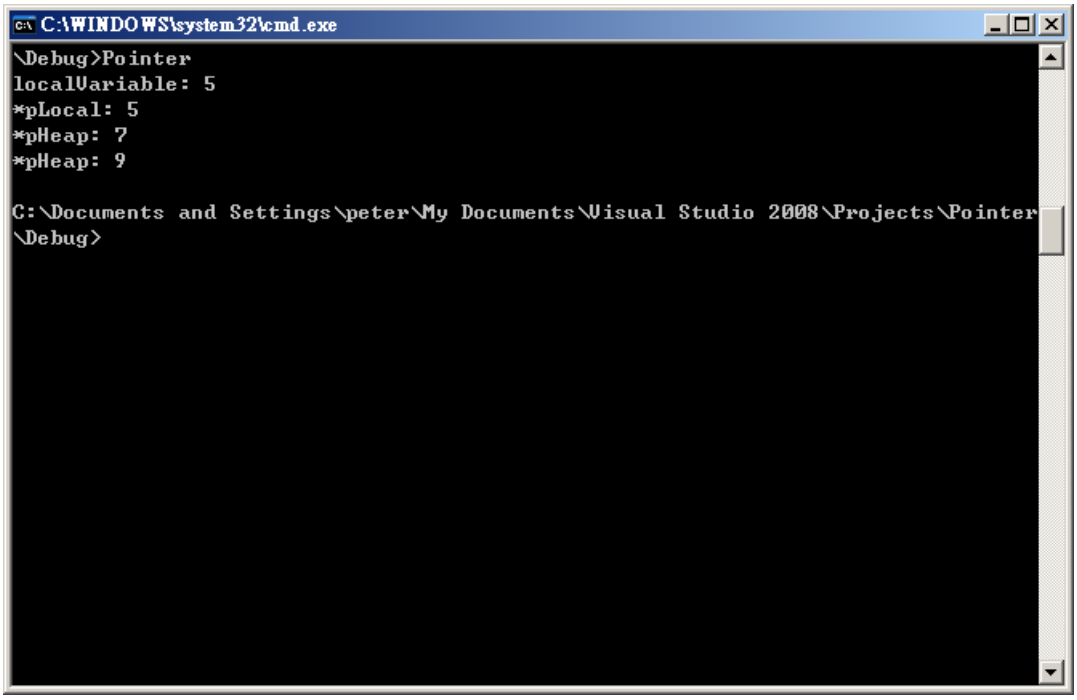
8. Allocating, Using, and Deleting Pointers

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **Pointer**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// Allocating and deleting a pointer
#include <iostream>
using namespace std;

void main()
{
    int localVariable = 5;
    int * pLocal= &localVariable;
    int * pHeap = new int;
    *pHeap = 7;
    cout << "localVariable: " << localVariable << "\n";
    cout << "*pLocal: " << *pLocal << "\n";
    cout << "*pHeap: " << *pHeap << "\n";
    delete pHeap;
    pHeap = new int;
    *pHeap = 9;
    cout << "*pHeap: " << *pHeap << "\n";
    delete pHeap;
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



4. Program Explanation
   - Line 7 declares and initializes a local variable. Line 8 declares and initializes a pointer with the address of the local variable. Line 9 declares another pointer but initializes it with the result obtained from calling new int. This allocates space on the free store for an int. Line 10 verifies that memory was allocated and the pointer is valid (not null). If no memory can be allocated, the pointer is null and an error message is printed.
   - To keep things simple, this error checking often won't be reproduced in future programs, but you must include some sort of error checking in your own programs.
   - Line 10 assigns the value 7 to the newly allocated memory. Line 11 prints the value of the local variable, and line 12 prints the value pointed to by pLocal. As expected, these are the same. Line 13 prints the value pointed to by pHeap. It shows that the value assigned in line 10 is, in fact, accessible.
   - In line 14, the memory allocated in line 9 is returned to the free store by a call to delete. This frees the memory and disassociates the pointer from that memory. pHeap is now free to point to other memory. It is reassigned in lines 15 and 16, and line 17 prints the result. Line 18 restores that memory to the free store.
   - Although line 18 is redundant (the end of the program would have returned that memory) it is a good idea to free this memory explicitly. If the program changes or is extended, having already taken care of this step will be beneficial.

9. Reverse String

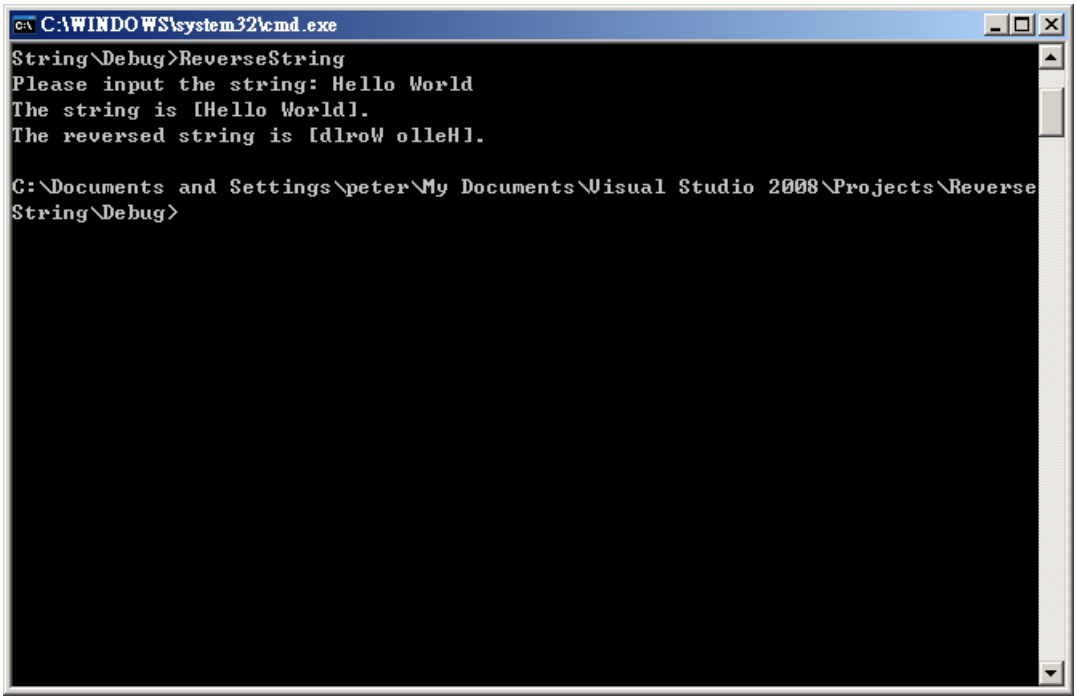
1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **ReverseString**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
// program that reverse a string in place,
// without any duplication of characters
#include <iostream>
using namespace std;

void reverse(char *);

void main() {
    char string[80];

    cout << "Please input the string: ";
    cin.getline(string, 80);
    cout << "The string is [" << string << "].\n";

    reverse(string);
    cout << "The reversed string is [" << string << "].\n";
}

/* Function: reverse()                                          */
/* it first loctes the end of the string.                       */
/* Then it swaps the first character with the last character, */
/* the second character with the second last character, etc.  */
void reverse(char* s) {
    char *end;
    char temp;

    // find end of s, could use strlen() instead of for loop here
    for (end = s; *end; end++) {
      ; //done nothing
    }
```

```
    while (s < end-1) {
        temp = *(--end);
        *end = *s;
        *s++ = temp;
    } // end while
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it.



10. Dynamic Pointer

1. Start the Microsoft Visual Studio and start a new Visual C++ Project. Select **Win32 Console Application** and name it as **DynamicPointer**. Remember to select **Application Setting** in the **Win32 Application Wizard**, then select **Console Application** and select **Empty Project**.

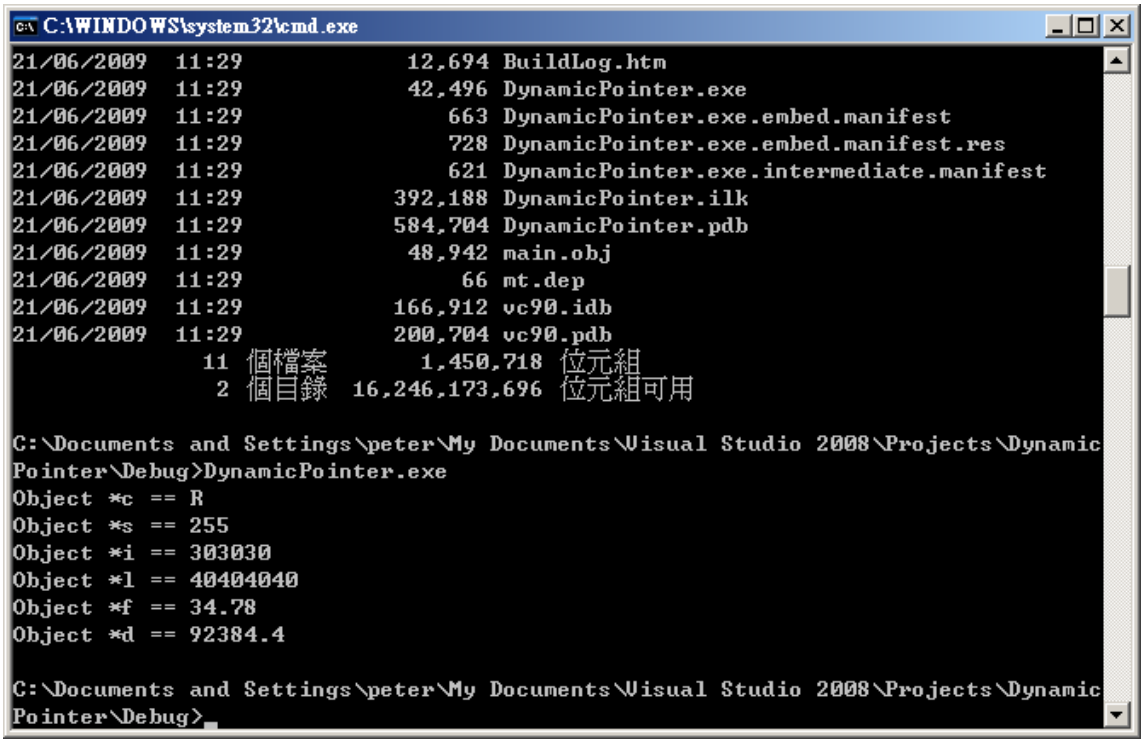
2. Create a **C++ File (.cpp)** and name it as **main.cpp** by selecting **Project ➔ Add New Item**, and then type the following code inside.

```
    ////////////////////////////////////////////////////
//  Module  : Dynamic Pointer
//
//  Purpose : Demonstrates creating, using, and
//                  destroying dynamic variables.
////////////////////////////////////////////////////
#include <iostream>
using namespace std;
const char NL = '\n';

// Global pointers for dynamic allocation
//
char   *c = 0;
short  *s = 0;
int    *i = 0;
long   *l = 0;
float  *f = 0;
double *d = 0;

// Function prototypes
//
void CreateObjects();
void DestroyObjects1();
void InitializeObjectValues();

////////////////////////////////////////////////////
// CreateObjects()

void CreateObjects()
{
    // Allocate the objects associated with the pointers
    //
    c = new char;
    s = new short;
```

```
    i = new int;
    l = new long;
    f = new float;
    d = new double;
}

////////////////////////////////////////////////////
// DestroyObjects1()

void DestroyObjects1()
{
    // Kill the objects associated with the pointers
    //
    delete c;
    delete s;
    delete i;
    delete l;
    delete f;
    delete d;
}

////////////////////////////////////////////////////
// InitializeObjectValues()

void InitializeObjectValues()
{
    // Initialize the objects to some value
    //
    *c = 'R';
    *s = 255;
    *i = 303030;
    *l = 40404040;
    *f = 34.78f;
    *d = 92384.42958;
}

////////////////////////////////////////////////////
// main()

void main()
{
    CreateObjects();
    InitializeObjectValues();

    cout << "Object *c == " << *c << NL;
    cout << "Object *s == " << *s << NL;
```

```
    cout << "Object *i == " << *i << NL;
    cout << "Object *l == " << *l << NL;
    cout << "Object *f == " << *f << NL;
    cout << "Object *d == " << *d << NL;

    DestroyObjects1(); // OK, objects are allocated
    DestroyObjects1(); // Oops! Objects already dead!
}
```

3. Use **Build Solution** to compile and build the solution, and then execute it. Please note that you will encounter an error when destroy a destroyed object in last line!!



4. Program Explanation
   - Lines 33-38: Dynamically allocate the six objects
   - Lines 48-53: Dynamically deallocate the six objects
   - Lines 63-68: Initialize the six objects with some values
   - Lines 76-77: Call helper functions to create and initialize the six objects
   - This program begins by declaring six global pointers to six built-in C++ data types (lines 13 through 18). Lines 22 though 24 give the function declarations for the three helper functions CreateObjects(), DestroyObjects1(), and InitializeObjectValues(). As their names suggest, these functions do the actual work of creating, initializing, and destroying the dynamic objects.
   - The CreateObjects() function runs from lines 29 through 39, and dynamically allocates the six types of objects by using the new operator for each object. The corresponding DestroyObjects1() function runs from lines 44 through 54, deallocating the six objects by using the delete operator. The initializeObjectValues() function runs from lines 59 through 69, initializing each of the objects with a value appropriate to its type.
   - The main() function ties everything together; it begins by calling the helper functions CreateObjects() and InitializeObjectValues() to create and initialize the six objects. Lines 79 through 84 simply display the values of the objects, and line 86 calls DestroyObjects1() to destroy the objects. Line 87 again calls DestroyObjects1(), attempting to delete the objects again.
   - Because the objects being pointed to have already been destroyed, the pointers are now pointing to the memory addresses where each object used to be. Deallocating this memory again can cause a problem—not always immediately, but eventually.
   - Imagine the following scenario: The objects are deleted, releasing their memory back into the free store. Each pointer still points to the starting address of the dead object. Now suppose that one or more of the memory addresses that the six pointers point to is grabbed up by some other dynamic allocation; later on in your code, the DestroyObjects1() function is called again. The function would attempt to delete memory that's now in use by another object! Depending on the actual situation, your implementation of C++, your hardware, and your compiler, any of several disastrous things could result from a bungle like this. This is why you should always set a pointer equal to zero after you delete its associated object.
   - You may wonder why the deallocation function is named DestroyObjects1(). This is because I wrote two versions of the function. The second version is called DestroyObjects2(); the second version of this function checks for null pointers before wantonly deallocating memory.