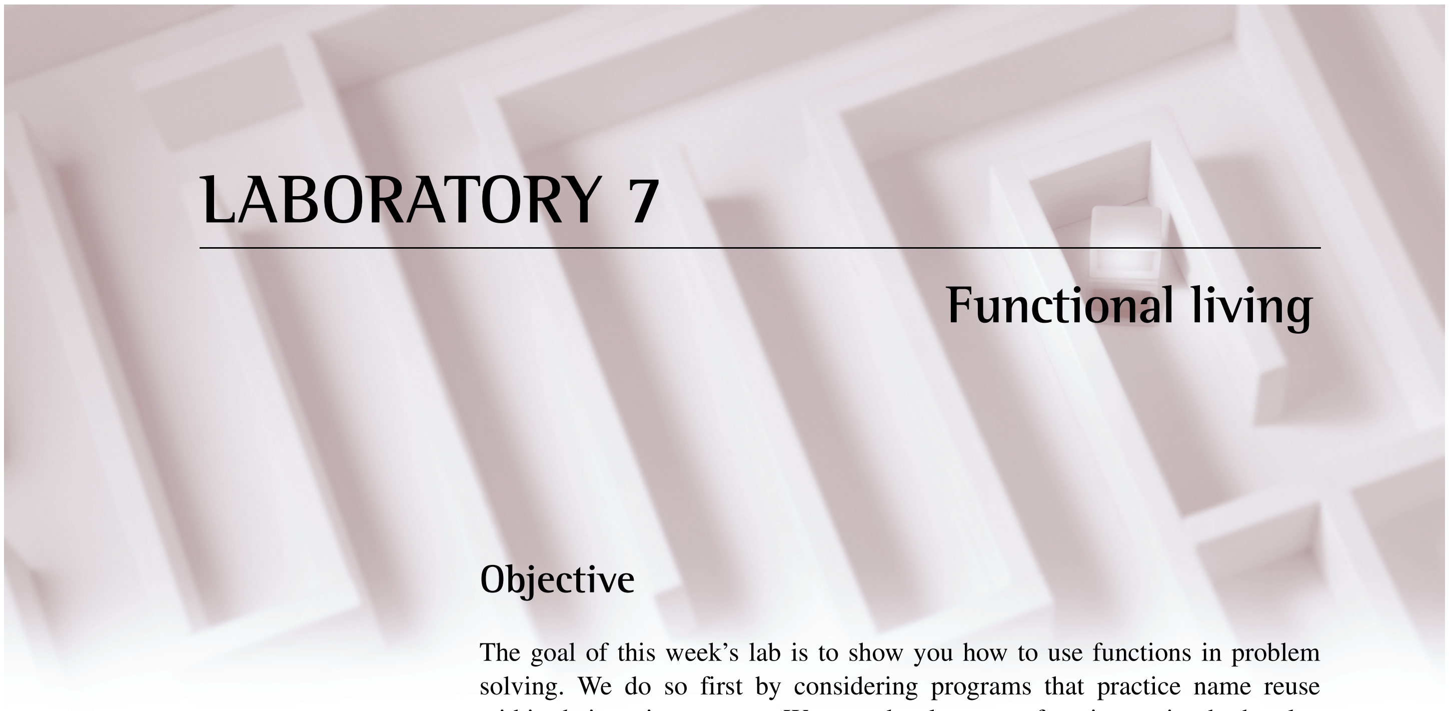


LABORATORY 7

Functional living

Objective

The goal of this week's lab is to show you how to use functions in problem solving. We do so first by considering programs that practice name reuse within their various scopes. We next develop some functions using both value and reference parameters. One of the functions makes use of recursion.

Key Concepts

- Local scope
- Global scope
- Recursion
- User-defined functions
- Program ordering

7.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab07.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

7.2 FUN WITH SCOPE

In last week's lab, you considered parameter passing behavior by analyzing various programs and determining their output. You will now perform a similar analysis to explain name reuse as it applies to some different scopes that can appear in a program.

For the next several problems, do the following:

- *Read the program, but do not run it!*
- Trace through it by hand and determine what the results of the program will be.
- Whenever you are asked to explain or describe something, write down your answers in the space provided.
- Discuss your results with your group and come to a consensus on the answer. When you have reached a consensus, open the file containing that program and run it. (You will need to create a default workspace). The programs are named for the respective problem—for example, Problem 1 corresponds to `prob01.cpp`.
- If your answer disagrees with the results produced when you ran the program, go back and see why your answer was incorrect. If you cannot figure out a problem, ask a laboratory instructor for assistance. Remember to get a check-off for each solution.

7.3 PROBLEM 1

```
#include <iostream>
using namespace std;

int i = 0;

int I = 1;

void f() {

   cout << "i: " << i << endl;

   cout << "I: " << I << endl;

   i = 10;

   I = 20;

}

int main() {

   int i = 2;

   int I = 3;

   cout << "i: " << i << endl;

   cout << "I: " << I << endl;

   f();

   cout << "i: " << i << endl;

   cout << "I: " << I << endl;

   return 0;
}
```

✎ ▪ What output is produced? ✓

7.4 PROBLEM 2

```
#include <iostream>
using namespace std;

int counter = 0;

void f() {

   ++counter;

}

void g() {

   f();
   f();

}

void h() {

   f();

   g();

   f();

}

int main() {

   f();

   cout << counter << endl;

   counter = 0;

   g();

   cout << counter << endl;

   counter = 0;

   h();

   cout << counter << endl;

   return 0;
}
```

✎ ▪ What output is produced? ✓

7.5 PROBLEM 3

```
#include <iostream>
using namespace std;

int i = 0;

int main() {

    int i = 1;

    cout << "i: " << i << endl;

    {

        cout << "i: " << i << endl;

        int i = 2;

        cout << "i: " << i << endl;

        {

            cout << "i: " << i << endl;

            int i = 3;

            cout << "i: " << i << endl;
        }

        cout << "i: " << i << endl;
    }

    cout << "i: " << i << endl;

    cout << "i: " << ::i << endl;

    return 0;
}
```

✎ ▪ What output is produced? ✓

7.6 PROBLEM 4

First determine whether the following program is correct. If it is not, indicate why. If it is correct, determine its output.

```
#include <iostream>
using namespace std;

void f(int a) {

    cout << "int a: " << a << endl;

    return;
}

void f(char a) {

    cout << "char a: " << a << endl;

    return;
}
int main() {

    int  i = 1;

    char c = 'c';

    f(i);

    f(c);

    return 0;
}
```

- What is your analysis (and output if any)?

✎

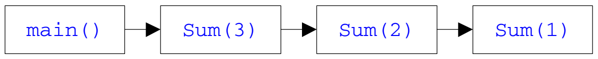
- Suppose `main()` included the invocation `f(2.5)`. Would the program still compile? Why? ✓

7.7 PROBLEM 5

The following program makes use of a recursive function `Sum()`. In C++, a function can invoke other functions including itself to do its task.

```
#include <iostream>
using namespace std;

int Sum(int number) {

   if (number == 1) {
      return 1;

   }
   else {
      int n = number - 1;

      int result = Sum(n);

      return number + result;
   }
}

int main() {

   int Answer = Sum(3);

   cout << "Answer: " << Answer << endl;

   return 0;
}
```

- To assist in your determination of the program output, fill in the depictions of the activation records on the following page. Remember *each* invocation of a function gets a new activation record. When a function makes a recursive invocation (as in other function invocations), the current invocation is suspended and the new invocation begins. When the new invocation finishes, its return value is brought back to the now reactivated invocation. The program you are considering will at one point in its execution suspend `main()` and invocations of `Sum(3)` and `Sum(2)` while it executes the invocation of `Sum(1)`.



Activation tree

- `main()`

Answer

- Activation record for invocation of `Sum(3)`.

`number`
`n`
`result`

- Activation record for invocation of `Sum(2)`.

`number`
`n`
`result`

- Activation record for invocation of `Sum(1)`.

`number`
`n`
`result`

✎
- What is the output? ✓

7.8 YOUR VERY OWN FUNCTIONS

In this activity you will write some small text-processing functions. The first is a function `IsEndOfSentence()` that determines whether its formal parameter `c` is either a period, a question mark, or an exclamation point. If the formal parameter is such a value, the function returns true; otherwise, the function returns false. Although there are various ways to implement this function, we shall consider only a single implementation.

- In designing a function, you must be concerned with its interface. Because `IsEndOfSentence()` returns a value, its return type cannot be **`void`**. What should the return type be?

return type

- Is formal parameter `c` a value or reference parameter? To answer this question, consider whether you expect or want the actual parameter to permanently change?

`c`'s parameter type

- Given these choices, how should you define the interface of the function?

- The body of the function is relatively simple. Some case analysis must be performed. If the value of `c` is a period, question mark, or exclamation point, the function returns true; if the value of `c` is something different, the function returns false. Such actions suggest an **`if`**-**`else`** statement with a test expression that is true if `c` is a period, question mark, or exclamation point. If the test expression is true, a **`return true`** is executed; otherwise, a **`return false`** is executed. Because the test expression can be made true in one of three ways, test expression needs three terms that are combined in a disjunctive manner, that is the terms are joined using the *or* operator. That way if any of the terms are true, the test expression as a whole is true. Each term will test whether `c` has a particular value. What should the test expression be?

✎ - Open the file `text.cpp` and add function `IsEndofSentence()` *after* function `main()`. ✓

The next function to consider is a function `LineSpace()` that has an optional parameter `n`. If `n` is positive, the function inserts `n` copies of the newline character (`'\n'`) to the standard output stream `cout`. If `n` is negative or not present, the function inserts one copy of the new-line character to the standard output stream `cout`.

- Should our function return a value? What should the return type then be?

return type

- Is formal parameter `n`, a value or reference parameter?

`n`’s parameter type

- Because an invocation of function `LineSpace()` does not require that an actual parameter be supplied, the specification of formal parameter `n` must include a default value. What should this value be? Given this and your previous choices, how should you define the interface of the function?

- Every invocation of function `LineSpace()` must insert at least one new-line character, so our function body begins with such an insertion.

```
cout << '\n';
```

- An iterator `i` will be used to control a loop that displays any additional new-lines. The iterator will represent the number of new-lines that have been inserted so far. How should `i` be defined so that it is properly initialized?

- The loop should iterate as long as more new-line characters need to be inserted. What should the test expression of the loop be?

- The basic action to be accomplished by the loop is an insertion of the new-line character.

```
cout << '\n';
```

  Once the new-line is inserted, one additional action needs to be performed to prepare for the next testing/iteration of the loop. What is this action?

✎ ▪ Implement `LineSpace()` *before* function `main()` in `text.cpp`. ✓

The next function to consider is a function `Update()` that has two parameters `counter` and `val`. Parameter `counter` is a reference parameter; parameter `val` is a value parameter. Function `Update()` determines whether `val` is an end-of-sentence punctuation character. If it is, `counter` is incremented by one; otherwise, `counter` is not modified.

- In general, functions with reference parameters are **`void`** functions—it is simpler to understand a function that brings information back to the invoking function in a single way rather than through both the return value and the parameter list. How should the interface of our function be defined?

- Our function is to modify `counter` only if `val` is an end-of-sentence punctuation character. Therefore an **`if`** statement is appropriate where `counter` is incremented only if the test expression is true. What should this test expression be?

✎ ▪ Implement function `Update()` *after* function `IsEndOfSentence()` in `text.cpp`. ✓

- Save your changes to `text.cpp`. This file implements a program that counts the number of sentences in a user-specified text file.

- Run the program in `text.cpp` using the data file `example.txt` and single spacing, that is `LineSpace()` should be invoked using its default parameter capability. (You will need to create a default workspace). Does the program produce the correct answer? If it does not, correct your functions so that they are properly implemented.

✎ ▪ Run the program a second time using double-spacing, that is function `LineSpace()` should be invoked with an actual parameter of 2. ✓

✎ ▪ Suppose function prototypes are not used. Where must functions `Update()` and `IsEndOfSentence()` be located in the program file relative to function `main()` and to each other? ✓

7.9 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.