

LABORATORY 6

Pass it on

Objective

This week in the lab we will be examining parameter passing. Understanding parameterized functions will enable you to design functions that correctly compute and return values to your program. Misuse of parameters is a common problem among programmers—a problem that often results in unexpected errors.

Key Concepts

- Programmer-defined functions
- Invocation and flow of control
- Value parameters
- Reference parameters
- `const` parameters
- `return` statement
- Local objects

6.1 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 `lab06.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.

6.2 FUN WITH PARAMETERS

Up to this point your experience with functions has been fairly limited. However, you should not underestimate their usefulness in imposing logical structure upon programs. An important part of using functions entails the understanding of parameters. Parameters enable functions to be flexible. With parameters a single function can handle a variety of related tasks—the parameters will determine which tasks to do and which values to compute. Parameters have this important role because they are the primary interface between the function to be invoked and the calling program fragment. Parameters enable programmers to control the values or objects that pass in and out of a function. Without parameters, we would be forced to rely on global objects as the sole means of communicating values between the various functions. However, we know that using global objects to share information is hazardous. Global objects make programs hard to trace and debug, since changing one global object could have ramifications throughout many different functions. Programs whose functions use parameters to share information are more modular and are easier to understand.

For each of the problems in this laboratory, perform the following activities with your group.

- *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, then 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.

Helpful Hint: When you try to trace parameters by hand, it is useful to draw boxes for each object and then write the current value in the box. This method enables you to see what is going on with each parameter. It also enables you to cross out an old value and replace it with a new one every time an object is updated.

6.3 PROBLEM 1

```
#include <iostream>
using namespace std;
void MyFunc() {
    return;
}

int main() {
    int i = 10;
    int j = 20;

    MyFunc();

    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```



- What output is produced? ✓

6.4 PROBLEM 2

```
#include <iostream>
using namespace std;
void MyFunc(int i, int j) {
    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;

    return;
}

int main() {
    int i = 10;
    int j = 20;

    MyFunc(i, j);

    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```



- What output is produced? ✓

6.5 PROBLEM 3

```
#include <iostream>
using namespace std;
void MyFunc(int i, int j) {
    i = i + j;
    j = j + i;
    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;

    return;
}

int main() {
    int i = 10;
    int j = 20;

    MyFunc(50, j);
    MyFunc(i, 50);

    cout << "main: i = " << i << endl;
    cout << "main: j = " << j << endl;
    return 0;
}
```



- What output is produced? ✓

6.6 PROBLEM 4

```
#include <iostream>
using namespace std;
void MyFunc(int i, int j) {
    int temp;

    temp = i;
    i = j;
    j = temp;

    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;

    MyFunc(a, b);
    MyFunc(b, a);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    return 0;
}
```



- What output is produced? ✓

6.7 PROBLEM 5

```
#include <iostream>
using namespace std;
void MyFunc(int &i, int &j) {
    int temp;

    temp = i;
    i = 30;
    j = temp;
    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;

    MyFunc(a, b);
    MyFunc(b, a);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    return 0;
}
```

- What output is produced?
- Why would it be illegal to change the first call of function `MyFunc()` in the function `main()` to `MyFunc(i, j)`? ✓



6.8 PROBLEM 6


```
#include <iostream>
using namespace std;
void MyFunc(int &i, int j) {
    int temp;

    temp = i;
    i = 30;
    j = temp;
    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;

    MyFunc(a, b);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    return 0;
}
```

- What output is produced?
- Why should it be illegal to change the call of MyFunc() in the function main() to MyFunc(10, b)?
-  Why should it be legal to change the call of MyFunc() in the function main() to MyFunc(a, 20)? ✓

6.9 PROBLEM 7

```
#include <iostream>
using namespace std;
int MyFunc(int &i, int j) {
    int temp;

    temp = i;
    i = 40;
    j = temp;
    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    return j;
}

int main() {
    int a = 10;
    int b = 20;
    int c = 30;

    c = MyFunc(b, b);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    cout << "main: c = " << c << endl;
    return 0;
}
```



- What output is produced? ✓

6.10 PROBLEM 8

```
#include <iostream>
using namespace std;
void MyFunc(int &i, int j, int k) {
    int a = 100;
    int b = 200;

    i = k + a;
    j = k + i;
    k = a;

    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    cout << "MyFunc: k = " << k << endl;
    cout << "MyFunc: a = " << a << endl;
    cout << "MyFunc: b = " << b << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;

    MyFunc(b, a, b);
    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    return 0;
}
```



What output is produced? ✓

6.11 PROBLEM 9

```
#include <iostream>
using namespace std;
void MyFunc(const int i, int &j) {
    int a = 100;
    int b = 200;

    a = b + j;
    j = i + a;

    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    cout << "MyFunc: a = " << a << endl;
    cout << "MyFunc: b = " << b << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;

    MyFunc(b, a);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    return 0;
}
```

- What output is produced?
- Why would it be illegal to add the statement `i = a + b;` to the function `MyFunc()`?



6.12 PROBLEM 10

```
#include <iostream>
using namespace std;
void MyFunc(int i, int j = 50) {
    int a = 100;
    int b = 200;

    i = i + j + a + b;

    cout << "MyFunc: i = " << i << endl;
    cout << "MyFunc: j = " << j << endl;
    cout << "MyFunc: a = " << a << endl;
    cout << "MyFunc: b = " << b << endl;
    return;
}

int main() {
    int a = 10;
    int b = 20;
    int c = 30;

    MyFunc(a, b);
    MyFunc(c);
    MyFunc(0);

    cout << "main: a = " << a << endl;
    cout << "main: b = " << b << endl;
    cout << "main: c = " << c << endl;
    return 0;
}
```



What output is produced? ✓

6.13 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpp1ab`.
- Hand in your check-off sheet.