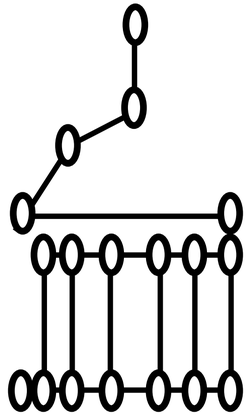
Lecture: Functions

ENGR 2730: Computers in Engineering



Five-Step Problem-Solving Methodology

1. State the problem clearly.
2. Describe the input and output.
3. Work hand examples.
4. Develop a solution/algorithm.
5. Test your solution.

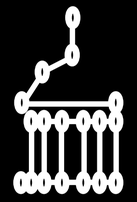
Pass-by-Value vs. Pass-by-Reference

- Pass-by-value: The function makes a copy of the input parameter.
  - Changing the copy does not change the original source.
- Pass-by-reference: The address of the input parameter is passed to the function.
  - When the function changes the value stored at that address, the value changes both in the function and in the original source.

Principle of Least Privilege

- Definition: Only give enough access to data to accomplish a specified task, but no more.
- Example: Only give a function enough access to the data in its parameters to accomplish its specified task, but no more.
- Example: Do not allow ave() to change the values stored in the data array or in count.

```
double ave(const double data[], const int count){
    double sum=0;
    for(int i=0; i<count; i++){
        sum += data[i];
    }
    return sum/count;
}
```



Write a function that swaps two numbers.

```
void swap(int & num1, int & num2);

int main() {
    int x1=3, x2=1;
    cout << "x1 = " << x1 << ", x2 = " << x2 << endl;

    swap(x1, x2);

    cout << "x1 = " << x1 << ", x2 = " << x2 << endl;

    return 0;
}

void swap(int & num1, int & num2){
    int tmp = num1;
    num1 = num2;
    num2 = tmp;
}
```

Note: Can't tell if parameters are pass-by-value or pass-by-reference when calling the function.

Note: & means parameters are pass-by-reference. Changing values in function change values in main().

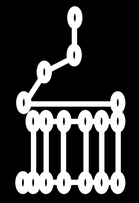
Passing arrays vs vectors to functions

```
#include <iostream>
#include <vector>

using namespace std;

double averageArray(const double data[], int size);
double averageVector(const vector<double> & data);

int main() {
    double values[] = {1.0, 3.5, 2.0, 4.3};
    vector<double> values2 = {1.0, 3.0, 4, 5};

    cout << "average1 = " << averageArray(values, 4) << endl;
    cout << "average2 = " << averageVector(values2);

    return 0;
}
```

Must pass size of array

In general, pass objects pass-by-alias

```
double averageArray(const double data[], int size){
    double sum=0;
    for(int i = 0; i < size; i++){
        sum += data[i];
    }
    return sum/size;
}

double averageVector(const  vector<double> & data){
    double sum=0;
    for(int i = 0; i < data.size(); i++){
        sum += data[i];
    }
    return sum/data.size();
}
```



CQ: What is the output of the following function?

```
void doSomething()
{
    int hist[5] = {0};
    int num_bins = 5;
    int indices[10] = {0, 0, 2, 3, 3, 3, 3, 4, 4, 4};
    int num_indices = 10;

    for (int i=0; i < num_indices; ++i)
    {
        hist[ indices[i] ]++;
    }

    for (int i=0; i < num_bins; ++i)
    {
        cout << hist[i] << " ";
    }
    cout << endl;
}
```

A. 0 0 0 0 0

B. 1 1 1 1 1

C. 2 0 1 4 3

D. 0 0 2 3 3

E. 2 3 4 4 4

C++ functions

default arguments

function overloading

In C++, within a function prototype, you can specify that the rightmost parameters have default arguments.

```
double distanceFromOrigin(double x, double y = 0, double z = 0);

double distanceFromOrigin(double x, double y, double z)
{
    return sqrt(x*x + y*y + z*z);
}
```

Example call: `distanceFromOrigin(1.0, 2.0)`

`0.0` will be used for third argument

Example of default arguments

```
// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1,
                       unsigned int width = 1,
                       unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1,
                       unsigned int width = 1,
                       unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify that the parameter length has a default argument - a default value (1) to be passed to length.

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1,
                       unsigned int width = 1,
                       unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify that the parameter width has a default argument - a default value (1) to be passed to length.

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1,
                       unsigned int width = 1,
                       unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify that the parameter height has a default argument - a default value (1) to be passed to length.

Example of default arguments

Defaults specified in declaration prototype!

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Default arguments must be specified with the first occurrence of the function name—typically, in the function prototype.

Defaults *NOT* specified in definition

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

No arguments — use default arguments for all parameters.

boxVolume(1, 1, 1)

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify `length`; default `width` and `height`.

`boxVolume(10, 1, 1)`

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify `length` and `width`; default `height`.

`boxVolume(10, 5, 1)`

Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

Specify all arguments.
boxVolume(10, 5, 2)

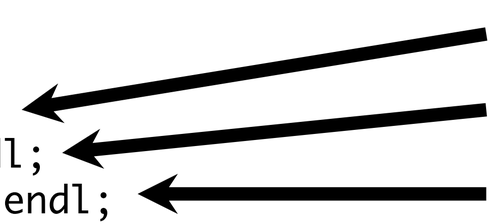
Example of default arguments

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout <
    cout <
    cout <
    cout <
}

// functio
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```

When a program omits an argument for a parameter with a default argument in a function call, the compiler rewrites the function call and inserts the default value of that argument.

Default arguments *must* be the rightmost (trailing) arguments in a function's parameter list

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

// function prototype that specifies default arguments
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

int main ()
{
    cout << boxVolume() << endl;
    cout << boxVolume(10) << endl;
    cout << boxVolume(10, 5) << endl;
    cout << boxVolume(10, 5, 2) << endl;
}

// function boxVolume calculates the volume of a box
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
    return length * width * height;
}
```



boxVolume(1, 1, 1)

boxVolume(10, 1, 1)

boxVolume(10, 5, 1)

Default arguments *must* be the rightmost (trailing) arguments in a function's parameter list

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

                                                                    1);
    return length * width * height;
}
```

Good Programming Practice 6.4

Using default arguments can simplify writing function calls. However, some programmers feel that explicitly specifying all arguments is clearer.

C++ functions

default arguments

function overloading

In C++, you can have multiple functions with the same name, as long as the sets of parameters (number, order, and/or types) are different.

```
int getMaximum(int a, int b);

double getMaximum(double a, double b);
```

Basics of function overloading

```
1  // Fig. 6.20: fig06_20.cpp
2  // Overloaded square functions.
3  #include <iostream>
4  using namespace std;
5
6  // function square for int values
7  int square(int x) {
8     cout << "square of integer " << x << " is ";
9     return x * x;
10 }
11
12 // function square for double values
13 double square(double y) {
14    cout << "square of double " << y << " is ";
15    return y * y;
16 }
17
18 int main() {
19    cout << square(7); // calls int version
20    cout << endl;
21    cout << square(7.5); // calls double version
22    cout << endl;
23 }
```

Fig. 6.20 | Overloaded `square` functions. (Part 1 of 2.)

Good Programming Practice 6.6

Overloading functions that perform closely related tasks can make programs more readable and understandable.

Basics of function overloading

```
1  // Fig. 6.20: fig06_20.cpp
2  // Overloaded square functions.
3  #include <iostream>
4  using namespace std;
5
6  // function square for int values
7  int square(int x) {
8     cout << "square of integer " << x << " is ";
9     return x * x;
10 }
11
12 // function square for double values
13 double square(double y) {
14    cout << "square of double " << y << " is ";
15    return y * y;
16 }
17
18 int main() {
19    cout << square(7); // calls int version
20    cout << endl;
21    cout << square(7.5); // calls double version
22    cout << endl;
23 }
```

Several functions of the same name can be defined with different number, types or order of parameters.

Used to create several functions of the *same* name that perform similar tasks, but on different data types.

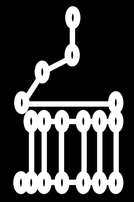
Fig. 6.20 | Overloaded `square` functions. (Part 1 of 2.)

Basics of function overloading

```
1  // Fig. 6.20: fig06_20.cpp
2  // Overloaded square functions.
3  #include <iostream>
4  using namespace std;
5
6  // function square for int values
7  int square(int x) {
8     cout << "square of integer " << x << " is ";
9     return x * x;
10 }
11
12 // function square for double values
13 double square(double y) {
14    cout << "square of double " << y << " is ";
15    return y * y;
16 }
17
18 int main() {
19    cout << square(7); // calls int version
20    cout << endl;
21    cout << square(7.5); // calls double version
22    cout << endl;
23 }
```

The compiler selects the proper function to call based on the number, types and order of the arguments in the call.

Fig. 6.20 | Overloaded `square` functions. (Part 1 of 2.)



Caution with Overloading

Common Programming Error 6.9

Creating overloaded functions with identical parameter lists and different return types is a compilation error.

Common Programming Error 6.10

A function with default arguments omitted might be called identically to another overloaded function; this is a compilation error. For example, having a program that contains both *a function that explicitly takes no arguments and a function of the same name that contains all default arguments results in a compilation error when an attempt is made to use that function name in a call passing no arguments. The compiler cannot determine which version of the function to choose.*

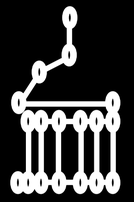
Clicker questions

CQ: Do you think having the following two function prototypes would be allowed in the same C++ program?

```
int doSomething(int a);

float doSomething(int b);
```

A. Yes

B. No



CQ: What is printed to the screen?

```
int mystery(int a, int &b, int c=1);

int main()
{
    int x = 2;
    int y = 2;

    int z = mystery(x,y);

    cout << x << ", " <<y << ", " << z << endl;

    return 0;
}

int mystery(int a, int &b, int c)
{
    a = a + 1;
    b = b + 1;
    c = c + 1;
    return a + b + c;
}
```

A. 2, 2, 7

B. 2, 2, 8

C. 2, 3, 7

D. 2, 3, 8

ProjectEuler.net, Problem 1. Multiples of 3 and 5

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Write a function that finds the sum of all the multiples of 3 or 5 below and input integer N.
Make the default value for N equal to 1000.

ProjectEuler.net, Problem 6. Sum square difference

The sum of the squares of the first ten natural numbers is,

$1^2 + 2^2 + ... + 10^2 = 385$

The square of the sum of the first ten natural numbers is,

$(1 + 2 + ... + 10)^2 = 55^2 = 3025$

Hence, the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 − 385 = 2640.

Write a function that takes a positive integer N as input and returns the difference between the sum of the squares of the first N natural numbers and the square of the sum. Test your function for N=100.