Week 6

strings, Pass by Reference, 1-Dimensional Arrays

New Material

- **Pass By Reference**
- **1-Dimensional Arrays**

Plan for Today

- Indexing into strings
- Pass by Reference
  - Practice problems
- 1-Dim Arrays
- Review and open discussion for questions:
  - Material
  - Exam Review

In Class Exercise

- Write code to print each letter in the following string on its own line?

string my_string = “Anabanana”

- Write code to count the number of a’s in the string my_string

Pass By Reference vs. Pass By Value

- What's the difference?

Pass By Reference vs. Pass By Value

- What's the difference?
  - Passing by values makes copies
  - Passing by references passes around the actual variable to make changes
- Both apply to calling functions
  - Pass by reference saves time and memory!
  - The & means the address in memory of the variable that is passed in

Syntax for Pass by Reference

```
void sum(int x, int y, int& result);

int main(){
        int a = 3;
        int b = 4;
        int result = 0;
        sum(a, b, result);
        return 0;
}

void sum(int x, int y, int& result){
        result = x + y;
        x = 0; y = 0;
}
```

Syntax for Pass by Reference

```
void sum(int x, int y, int& result);

int main(){
        int a = 3;
        int b = 4;
        int result = 0;
        sum(a, b, result);
        return 0;
}

void sum(int x, int y, int& result){
        result = x + y;
        x = 0; y = 0;
}
```

x and y are passed by value
Result is passed by reference

-- Not all parameters must be the same
-- Some can be passed by value, and some can be passed by reference!

Syntax for Pass by Reference

x and y are passed by value
Result is passed by reference

```
void sum(int x, int y, int& result);

int main(){
        int a = 3;
        int b = 4;
        int result = 0;
        sum(a, b, result);
        return 0;
}

void sum(int x, int y, int& result){
        result = x + y;
        x = 0; y = 0;
}
```

What are the values of a, b, and result when the code is complete?

Syntax for Pass by Reference

x and y are passed by value
Result is passed by reference

```
void sum(int x, int y, int& result);

int main(){
        int a = 3;
        int b = 4;
        int result = 0;
        sum(a, b, result);
        return 0;
}

void sum(int x, int y, int& result){
        result = x + y;
        x = 0; y = 0;
}
```

What are the values of a, b, and result when the code is complete?

a is 3
b is 3

result is 7

Syntax for Pass by Reference

```
void sum(int x, int y, int& result);

int main(){
        int a = 3;
        int b = 4;
        int result = 0;
        sum(a, b, result);
        return 0;
}

void sum(int x, int y, int& result){
        result = x + y;
        x = 0; y = 0;
}
```

What are the values of a, b, and result when the code is complete?

a is 3
b is 3

result is 7

x and y are out of scope as soon as the function is done

Result is in scope for this entire program

New and important for Pass by Reference

- Types must be **identical**
  - No implicit casting can occur
    - A double cannot be passed into a function that takes an int as its pass-by-reference parameter
- **Must pass in a variable**, if parameter is passed by reference (can't be just a number or expression
- The & can be in different places, and still act the same way

These are all the same parameter:

(int &number)

(int& number)

(int & number)

Unchanged details

- **Variable names can still be different in main and in the function**

For example the following is just fine:

```
int main(){
        int a = 3, b = 4, result = 0;
        swap(a, b, result);
        cout << result; //prints 7
        return 0;
}

void swap(int x, int y, int& sum_in){
        sum_in = x + y;
}
```

Result will still be updated to 7 in main
because it was passed by reference!

Practice Question 1

- Is anything wrong with this? If so, what is it?

```
int main(){
        int x = 1;
        int y = 2;
        product(x * y);
        return 0;
}
```

```
void product(int& a){

        a = a * a;
}
```

Practice Question 1

- Is anything wrong with this? If so, what is it?

```
int main(){
        int x = 1;
        int y = 2;
        product(x * y);
        return 0;
}
```

```
void product(int& a){

        a = a * a;
}
```

Compile error

Can't pass in an expression where a variable is expected

Practice Question 2

- Is anything wrong with this? If so, what is it?

```
int main(){
        double x = 3.0;
        product(x);
        cout << x;
        return 0;
}
```

```
void product(int& a){

        a = a * a;
}
```

Practice Question 2

- Is anything wrong with this? If so, what is it?

```
int main(){
        double x = 3.0;
        product(x);
        cout << x;
        return 0;
}
```

```
void product(int& a){

        a = a * a;
}
```

Compile error

Can't pass in a double type when int type is expected – no implicit type casting!

Practice Question 3

- What does this print?

```
int main(){
        double x = 3.0;
        product(x);
        cout << x;
        return 0;
}
```

```
void product(double& a){

        a = a * a;
}
```

Practice Question 3

- What does this print?

```
int main(){
        double x = 3.0;
        product(x);
        cout << x;
        return 0;
}
```

```
void product(double& a){

        a = a * a;
}
```

OUTPUT:

9

Practice Question 4

- Is anything wrong with this? If so, what is it?

```
int main(){
        double x = 3.0;
        cout << product(x);
        return 0;
}
```

```
void product(double& a){

        a = a * a;
}
```

Practice Question 4

- Is anything wrong with this? If so, what is it?

```
int main(){
        double x = 3.0;
        cout << product(x);
        return 0;
}
```

```
void product(double& a){

        a = a * a;
}
```

Compile error

Can't print the value of a void function

Practice Question 5

- What is the output after this code runs?

```
int main(){
        int x = 1, y = 2;
        product(y, x);
        cout << x;
        cout << endl;
        cout << y;
        return 0;
}
```

```
void product(int& a, int x){

        a = a * a;
        x = a * 2;
}
```

Practice Question 5

- What is the output after this code runs?

```
int main(){
        int x = 1, y = 2;
        product(y);
        cout << x;
        cout << endl;
        cout << y;
        return 0;
}
```

```
void product(int& a, int x){

        a = a * a;
        x = a * 2;
}
```

OUTPUT:

1
4

Function Parameters - Summary

Value Parameters	Reference Parameters
Receives a copy	Receives a **reference** to the memory location
Single value, or nothing returned	**Multiple values** may be passed back *in effect*
Type coercion is allowed	**Types must match** parameter declaration
Can be a variable, constant, or expression	Must be a **variable**

1-Dimensional Arrays

- Think of an array as a container that can hold many items of a certain data type
- An array is a collection of items of the same data type stored **consecutively in memory** under one name
- You can access elements in an array by indexing just like we did with strings

Array Declaration and Initialization

```
int my_array[5];

int my_array[] = {1, 2, 3, 4, 5};

int my_array[5] = {1, 2, 3, 4, 5};

int my_array[8] = {1, 2, 3, 4, 5}; //compiler will insert 0's
                                   after 5th elt, usually
```

ALL VALID

Accessing Array Elements

```
int my_array[] = {1, 2, 3, 4, 5};
```

How would we print only the 3?

Accessing Array Elements

```
int my_array[] = {1, 2, 3, 4, 5};
```

How would we print only the 3?

```
cout << my_array[2];
```

- Many times you will iterate through an array using a loop to access or initialize elements

Types of Arrays

- Arrays can be of any data type, as long as all elements are of the same data type
- For example:

```
string my_array[6];
double your_array[8];
bool our_array[4];
```

- **Arrays are always 'passed by reference' – no & needed**

Iterating through an array

```
const int arr_size = 5;
int my_array [arr_size] = {1, 2, 3, 4, 5};

for (int i = 0; i < arr_size; ++i){
        cout << my_array[i];
}
```

Iterating through an array

```
const int arr_size = 5;
int my_array [arr_size] = {1, 2, 3, 4, 5};

for (int i = 0; i < arr_size; ++i){
        cout << my_array[i];
}
```

- Careful not to go off the end of the array
- For instance, if the condition was I <= arr_size, the program would try to access my_array[5] on the last iteration, which doesn't exist

Discussion open for questions!