Introduction to Software Development – CS 6010
Lecture 8 – References

Master of Software Development (MSD) Program

Varun Shankar

Fall 2022

Notes from Yesterday…

- How to return multiple pieces of data from a function?
  - 1) Use a vector if all the pieces of data being returned are the same type.
  - 2) Use a struct if the data is of different types (or if it makes sense).
    - For example, if you wanted to return the x and y values of a point, return a *Point*:

```
struct Point {
        float x;
        float y;
}
```

Lecture 8 – References

- Topics
  - References
  - Code Review – Deck of Cards
  - Homework – Poker

Issues We Want To Solve

- for(char c : myString) // Change the letters in myString
- void swap(int a, int b) // swap the values of a and b such that the caller sees them swapped
  - Take a few minutes to write the *swap()* function in class.

```
int main()
{
    int x = 1;
    int y = 2;
    swap( x, y );
    assert( x == 2 );
    assert( y == 1 );
}
```

Pass By Value vs Pass By Reference

- By Value
  - The value of the parameter is copied into the function.
  - Changing the value of the parameter inside the function does not effect the value outside the function.
- ByReference
  - The parameter just refers to (references) the variable that was passed into the function.
  - If the parameter's value is changed in the function, the value outside the function changes too.

Example of Pass By Value Parameters

```
int main() {
        int x = 2;
        int y = 4;
        swap( x, y );
        cout << “x is: “ << x << “, and y is: “ << y
}
```

```
void swap( int a, int b ) {
    int temp = a;
    a = b;
    b = temp;
}
```

- What are the values of x and y at this point?
- They stay the same – only the *values* were passed into the function
  - In other words, *a* received a copy of *x* (and thus the value 2). But when *a* changes, *x* remains the same.

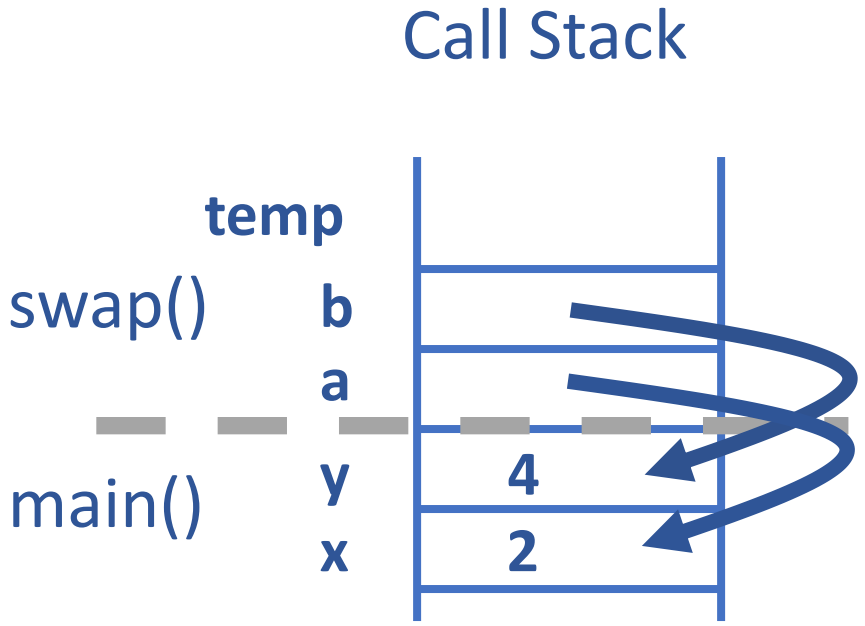
References (New Datatype)

- Reference variables are just another name for an existing variable/value
  - The do not make a copy of the original
  - As far as the programmer is concerned, they are the original
- Syntax: Use the '&' after the type to get a "reference to that type"
  - References must always "reference" another variable
  - int i; // What we've been doing – Creates an integer variable
  - int & i; // NEW – Create a *reference to an integer* variable

Example of Pass By Reference Parameters

```
int main() {
        int x = 2;
        int y = 4;
        swap( x, y );
}
```

```
void swap( int & a, int & b ) {
    int temp = a;
    a = b;
    b = temp;
}
```



- Now what are the values of *x* and *y*?
  - They have been swapped because inside *swap()* the variable *a* refers to the variable *x*, so when *a* is modified, so is *x*. The same happens for *y*.
  - Use reference parameters to modify variables in some other function's stack frame.

When parameters are passed by reference, it means: The function intends to change the value of that variable.

shuffleDeck()

- How do we get the shuffled deck back to the caller of this function?

```
vector<Card> shuffleDeck( vector<Card> deck );

deck = shuffleDeck( deck ); // Send in the old deck, the new deck is returned
                            // (And overwrites the old deck)
```

- How many decks exist?
  - During *shuffleDeck(),* **TWO decks exist.**

- By using references, we can just *modify* the deck that is passed into the function:

```
void shuffleDeck( vector<Card> & deck );  // deck is modified directly, nothing is returned
```

References and For Each Loops

```
vector<int> numbers = { 3, 5, 9, 15, -5 };
for( int i : numbers ) { … }
```

- The values in *numbers* are **copied** into the variable *i* (one at a time). Changing *i* does not change the value in numbers.
- Reference version:

```
for( int & i : numbers ) { // Only a subtle change… the & was added.
        i = 0;                    // What happens to numbers?
}
```

- Now all the *numbers* are set to 0: { 0, 0, 0, 0, 0 }

Constant *Variables / Parameters*

- The keyword *const* is used to specify that a variable (or parameter) cannot change.
- The compiler will enforce this rule for you.
- *const* works well with reference parameters
  - It allows you to pass large amount of data into a function without copying the data
  - And at the same time, guarantees that the function will not change the data

Const *Parameters* and Data Copying

void print(vector<string> sentences);

Using print():

vector<string> shakespeare = … all the works of shakespeare

print(shakespear); // **Copies** all the data in *shakespeare* into the print() function

void print(vector<string> & sentences);

This version of print will only *reference* the data in sentences!

But it would be valid for the *print()* function to change some of those sentences!

void print(const vector<string> & sentences);

This is the winner! No copying, and the function is not allowed to modify the data.

Today’s Assignment(s)

- Code Reviews – String Analyzer
- Homework – Poker