Programming Languages and Tools: Programming with C++ CS:3210:0003

Lecture/Lab #8

Passing Function Arguments by Reference

- Until now we have been passing arguments by value:
  - function opens a local scope for argument
- A reference is an alias to a variable
- Syntax:

```
varType varName = Value;                    //variable
varType& refName = varName;        //reference to variable
```

- To pass a variable by reference, in the function header:

```
returnType functionName (varType& refName, …)
```

- Passing by reference helps for large values, or to return multiple values
- Constants can’t be referenced

Passing Arrays

- To pass arrays to functions, pass the array and the length separately:
  returnType funcName (type arrayName [], int arrayLen)
- This is **not** a pass-by-value
- Arrays cannot be passed by value
- When passed this way, the value *decays* to a pointer to the beginning of the array
- These are called C-style (static) arrays
- More recent versions of C++ has a better array type that can be passed by value and by reference

C-Style Strings

- Special case of an array of characters. Ex:
  cout << “Hello World” << endl;
- The above is equivalent to:
  char sayHello[] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘ ’, ‘W’, ‘o’, ‘r’, ‘l’, ‘d’, ‘\0’};
  std::cout << sayHello << std::endl;
- The character ‘\0’ is the null character
  - Tells the compiler that the string has ended
- For string literals, compiler implicitly adds the null character

std::string

- For string literals, C-style strings are fine to use (ex: with cout)
- For string variables, std::string is a safer type to use
- Lives in the <string> header
- std::string is dynamic (unlike C-style strings since they are just C-style static arrays)
- Use getline instead of cin if you want to input space-separated input
- Implemented as a class

Strings

C-Style String	std::string
Null('\0')-terminated char arrays	Separate type for strings in std
Static	Dynamic
No call-by-value (decays to pointer)	Can call by value or reference
Use with literals	Use with variables and computations
Ex: char cstr[] = {‘E’, ‘x’, ‘\0’};	Ex: std::string stdstr = “Ex”

Arrays

C-Style Arrays
Static
No call-by-value (decays to pointer)
C++ has better array types
Int intArray[3] = {0, 1, 2}

Switch Statement

- Condition is evaluated to produce a value
- If value is equal to the value after any of the case labels, executes statements after matching case label are executed
- If no match, default label statements are executed
- Compiler will go through every case, even if a match is found
  - Use break to avoid this
- Switch cases can have multiple statements without braces ({ })
- A switch block (everything between its braces) has a single scope that all variables inside it share

Switch Statement

- When switch finds a match and executes matching statements and continues sequentially unless:
  1. The end of the switch block is reached
  2. Another control flow statement (like break or return) causes switch/function to exit
- Switch will fallthrough to next case without break/return

Loops

- Nested loops – loop inside loop
- Inner loops completes all iterations for each iteration of outer loop

Activity

Write a program nestedn.cpp that inputs some positive integer n as the input and counts from 1 to n incrementally in n lines. For example, for n == 3, output:

1
1 2
1 2 3

for Loops

- Most common form of C++ loops
- Syntax:
  for (init-statement; condition; end-expression)
    body
- Init-statement is executed once when loop is initiated
  - Variables initialized have loop scope
- Condition is checked at the beginning of each iteration
- End-expression is evaluated at the end of each iteration
- Any or all of init-statement, condition, end-expression can be empty
- When you have a counter, use for loops, otherwise use while

do while Loops

- Body of loop executes at least once
- Syntax:

```
do
   body
while (condition);
```

- Variable tracking exit criteria should be declared before loop begins
- Least common form of C++ loops

break and continue

- Break
  - In a switch, used at the end of the case to prevent fallthrough
  - In a loop, used to exit loop early

- Continue
  - End current iteration of loop without terminating the loop
  - Execution jumps to bottom of loop
  - In while/do while loops, continue might skip over update statement