Lecture 2 Notes

Strings

string is a data type that represents a sequence of characters. In order to use this data type, you must insert this line at the beginning of your source code:

```
#include <string>
```

A literal value of a string is expressed in double quotes with an ‘s’ suffix.

Listing 1: Examples of string variable declarations

```
string teaching_assistant = "mark"s;
string course_name = "CSE 201"s;
string course_description = "Introduction to C++"s;
```

There are special characters which you cannot use inside a string’s literal value. Characters such as double quotes and line feeds will break the syntax of your code. To use these characters inside a string, you must use an “escape sequence”.

An escape sequence is a sequence of characters which represents another character in it’s place:

Character	Escape sequence
Beep	`\a`
Backspace	`\b`
Formfeed	`\f`
Newline	`\n`
Carriage Return	`\r`
Horizontal Tab	`\t`
Vertical Tab	`\v`
Backslash	`\\`
Single quotes	`\'`
Double quotes	`\"`

Here is an example of a string that contains multiple lines and double quotes:

```
"\"My name is Linus Torvalds and I am your god.\"\n- Linus Torvalds"s
```

The `string` data type can be used with the addition operator to combine strings:

Listing 2: Example of combining strings

```
string name;
string prefix_message;
string full_message;

name = "Mark"s;
prefix_message = "Happy birthday, "s;
full_message = prefix_message + name + "!"s;
```

The example above will produce a variable `full_message` containing “Happy birthday, Mark!”.

cout

cout is a variable that you can use to write messages to the terminal window. You can use it after inserting this statement at the beginning of your source code:

```
#include <iostream>
```

cout stands for “character out”, it’s data type is ostream. You can insert data into an ostream using the insertion operator:

```
cout << "hello world!\n"s;
```

The insertion operator returns the stream that you wrote to, therefore you can chain the insertion operator to insert multiple values:

```
cout << "You must be "s << 21 << " to enter\n"s;
```

cin

cin stands for “character in”, t is a variable that you can use to read data from the keyboard. It is useful for getting user input. Like cout, you must also include iostream to use it.

The data type of cin is istream. You can extract data from an istream using the extraction operator:

Listing 3: Example of reading user input

```
string firstname;
int age;
cout << "Input first name: "s;
cin >> firstname;
cout << "Input age: "s;
cin >> age;
```

The extraction operator requires a variable for it’s right-hand operand. The extraction operator extracts a value from it’s istream and stores it into any variable you specify.

Extracting data from an istream into a string only extracts non-whitespace characters, in other words: it only reads a single word. To read an entire line from an istream, you must use the getline function:

```
string full_name;
cout << "Enter first and last name: "s;
getline(cin, full_name);
cout << "Hello ,"s << full_name << '\n';
```

ostream manipulators

You can control the output formatting of an ostream by inserting ostream manipulators. To use ostream or istream manipulators, you must use this statement:

```
#include <iomanip>
```

Listing 4: Using ostream manipulators to control formatting of double values

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

int main()
{
   cout << 98.07 << '\n';
   cout << fixed << 98.07 << '\n';
   cout << scientific << 98.07 << '\n';
}
```

The above program outputs:

```
98.07
98.070000
9.807000e+01
```

The `fixed` and `scientific` manipulators provide alternative output formats for `double` values. The `defaultfloat` manipulator resets the `double` output formatting to default.

For `int`, you can control which numeric base to use for output:

Listing 5: Example of setting output numeric base for integers

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

int main()
{
   cout <<
      dec << 42 << '\n' <<
      hex << 42 << '\n' <<
      oct << 42 << '\n';
}
```

The above program outputs the integer 42 in decimal, hexadecimal, and octal.

The `setprecision` manipulator sets the maximum number of digits to display after the decimal point:

Listing 6: Setting output precision of a double

```
double pi = 3.141592653589793239;
cout << setprecision(10) << pi << '\n';
```

The code above would output: `3.141592654`. Notice that `setprecision` takes an additional value in parenthesis in order to operate, this additional value is known as an argument.

Most ostream manipulators produce permanent effects on the ostream. The `setw` manipulator only affects the next data insertion, it pads additional whitespace to the next insertion if necessary:

Listing 7: Example of whitespace padding using setw manipulator

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

int main()
{
   cout << left;
   cout << setw(10) << "Cups"s << setw(10) << "Oz"s << '\n';
   cout << setw(10) << 1 << setw(10) << 8 << '\n';
   cout << setw(10) << 2 << setw(10) << 16 << '\n';
   cout << setw(10) << 3 << setw(10) << 24 << '\n';
}
```

In this example, we used 10 as the argument for `setw`, making the next inserted value have padded whitespace until it's character width is 10. We also used the `left` manipulator to make the padded output left-aligned, the default is right-aligned. The result is:

```
Cups Oz
1 8
2 16
3 24
```

– *Mark Swoope*