Class Notes

EECS 183
University of Michigan

Jinyan Miao
Fall 2021

Definition 0.0.0.0.1
*An **algorithm** is a sequence of steps to solve a problem. Algorithm is an ordered sequence of small tasks, or instructions, and instructions are well-defined and unambiguous. Everyone would perform the instruction the same way, and stops at some point when the problem is solve.*

Note that algorithm requires some human interpretation. In programming, it is usually expressed in pseudocode.

Definition 0.0.0.0.2
Program is a way expressing an algorithm in a way a computer can understand.

Definition 0.0.0.0.3
Programming is collaborative problem-solving between people and computers, giving a computer a list of instructions to follow.

Definition 0.0.0.0.4
A computer is a machine that executes algorithm.

Definition 0.0.0.0.5
Psedocode uses human language to describe an algorithm, structured like a programming language. It is words and phrases that people can understand.

Definition 0.0.0.0.6
Programming language offers a way for us to communicate algorithm to computer.

Computers can not interpret, so programming language needs to be unambiguous. For example, C++ is an programming language.

When we try to run a program, a source code, such as C++ code, expressed by a algorithm, will be passed to the IDE compiler, such as XCode and VS, which will then turn into executable, the application, that can be run in the CPU.

```
#include < iostream >                          Imports the library iostream
#include < string >                           Imports the library string for string
#include < cmath >                           Imports the library cmath for math functions
using namspace std;

int main(){                                  All C++ code starts with using the main() function
    cout << "Hello World";                   cout prints the string Hello World
    int x;                                   int x declares a variable x as an integer
    int y = 4;                               Declare variable y and set it to 4
    x = 3;                                   Assign 3 to the variable x
    x = y;                                   Permanently change the value of x to y=4
    y = "M";                                 Change the value of y to M, while x remains 4
    x = x + 2;                               Add 2 to x, now x is 6
    x + 2;                                   This line does nothing, but the program keeps running
    return 0;                                Ends the program
    cout << x;                               Prints the value of x, here it prints 6
}                                            Ends the main() function
```

Every time we use an assignment operation "=", we permanently change the value of the variable. If we did not assign a new value to a variable, the value of the variable will not change regardless how other variables change.

The 0 in the "return 0;" statement tells the operating system the program ended without error.

Compile errors are usually caused by grammar mistake in the code, the compiler, or IDE, got confused about the grammar of the code.

Floating points are stored as "double", integers are stored as "int" in the C++ language. A type conversion is a conversion of one data type to another, such as an int to a double. The compiler automatically performs several common conversions between int and double types, such automatic conversion known as implicit conversion. For an arithmetic operator like + or *, if either operand is a double, the other is automatically converted to double, and then a floating-point operation is performed. For assignments, the right side type is converted to the left side type. As a result, if we perform the calculation $z = x/y$, where x and y are both integers, and we have $x = 5$ and $y = 3$. If z is declared as an integer, then z will also be an integer and we have $z = 1$. If z is declared as a double, then z will be an float and $z = 1.0$.

For explicit conversion, the static_cast operator:

$$static\_cast < type > (expression)$$

converts the expression's value to the indicated type.
For example: If myIntVar is 7, then static_cast<double>(myIntVar) converts int 7 to double 7.0.

A variable of char type, as in char myChar;, can store a single character like the letter m. A character literal is surrounded with single quotes, as in myChar = 'm';.

A string is a sequence of characters. A string literal surrounds a character sequence with double quotes, as in "Hello", "52 Main St.", or "42", vs. an integer literal like 42 or character literal like 'a'. Various characters may be in a string, such as letters, numbers, spaces, or symbols like $, as in "$100 for Julia!!". Note that when using strings, we need to include $\#include < string >$ at the beginning of the program to import the string library.

Sometimes we might want to declare a constant, which is not allowed to change throughout the entire script. To declare a constant, we can use the following syntax:

$$const\ double\ X\ =\ 5.0; \qquad const\ int\ Y\ =\ 3;$$

A whitespace character is a character used to represent horizontal and vertical spaces in text, and includes spaces, tabs, and newline characters. Ex: "Oh my goodness!" has two whitespace characters, one between h and m, the other between y and g. When assigning a value to the string using *cin*, the *cin* only takes the input up to the next white space. That is, if the input is "Hello world" for $cin\ <<\ myString$, then we will have $myString$ = "$Hello$", the word "world" will be ignored, or left for the next *cin*. Sometimes a we want to get whitespace characters into a string, such as getting a user's input of the name "Franklin D. Roosevelt" into a string variable presidentName. For such cases, the language supports getting an entire line into a string. The function $getline(cin,\ stringVar)$ gets all remaining text on the current input line, up to the next newline character, which is removed from input but not put in stringVar. On another note, *cin* ignores all the whitespace and newline character, while $getline(cin,\ stringVar)$ does not. That is, if the input is " Hello World", *cin* will get 'Hello", while $getline(cin,\ stringVar)$ will get " Hello World". Moreover, if the input is "Hello\n World", and we first use *cin* to get the input "Hello", and then use $getline(cin,\ stringVar)$ to get the next input for variable stringVar, then stringVar will be an empty string, this is because $getline(cin,\ stringVar)$ does not skip newline character.

C++ function can only return one singular value of a single type. To declare a user-defined function, we need the following code:

$$
\begin{aligned}
&double\ FunctionName(string\ parameter1,\ int\ parameter2)\{\\
&\quad Something = 19.0;\\
&\quad return\ Something;\\
&\}
\end{aligned}
$$

Here the function return a type double variable "Something." Other return types are allowed, such as char, double, etc. A function can only return one item, not two or more. A return type of void indicates that a function does not return any value. We can have no parameter, or more parameters, when declaring the function, while notice that the type of the parameters also need to be included when declaring a function.

One might declare a function before defining the function. That is, at the beginning of the program, before the main() function, one might declare a function using the following:

$$void\ myFunction(int\ x);$$

and define the function at the end of the program, after the main() function:

$$
\begin{aligned}
&void\ myFunction(int\ x)\{\\
&\quad return;\\
&\}
\end{aligned}
$$

The scope of a variable starts at the declaration point of the function, and ends at the closing bracket of the enclosing block, which is the end of a user-defined function, the end of the main() function, etc.. Once the execution leaves the scope of a variable, by return statement or reaching the end of the scope, the variable gets de-allocated. On the other hand, the scope of the declared functions starts at their declaration points and ends at the end of the program. When not in the scope of a variable or function, one cannot write a statement using that variable or function