Introduction to Numerical Methods SS 23 Lecture Notes

Igor Dimitrov

Apr 26, 2023

CONTENTS

Introduction to Numerical Methods Uni Heidelberg summer semester 23 Lecture Notes.

Part I

Programming Tutorial

CHAPTER ONE

INTRODUCTION

C++ programming tutorial notes

Some links:

- hedgedoc link
- git repo for hdNUM likn

CHAPTER

TWO

HELLO WORLD

Libraries are included with the `#include` directive.

- `<iostream>`: standard library for standard input/output (keyboard and screen)
- `<vector>`: library for dynamic arrays, i.e. arrays whose size changes during run-time. Implemented as amortized arrays (rather than linked lists, contrary to what one would naively assume).
- `"hdnum.hh"`: header file to include the custoim `hdnum` library.

Note: standard libraries are enclosed with angle brackets, progammer-defined header files are enclosed with double quotes

```
#include <iostream>
#include <vector>
#include "../../../hdnum/hdnum.hh"
```

and print out some messages to standard outpout:

```
std::cout << "hello" << std::endl;
std::cout << "1 + 1 = " << 1 +  1 << std::endl;
```

```
hello
```

```
1 + 1 = 2
```

Full program text:

```
#include <iostream>
#include <vector>
#include "../../../hdnum/hdnum.hh"

int main()
{
    std::cout << "hello" << std::endl;
    std::cout << "1 + 1" << 1 + 1 << std;endl;

    return 0;
}
```

To compile a single program called `hello.cpp` enter the command:

```
$ g++ -o hello -I../../../ hello.cc
```

where `-I../../` is the include option indicating where the header files are located. In this case the headers are located in a folder **three levels above** the folder where `hello.cc` is located and is being compiled.

To run the compiled program enter `./hello` at the CLI

CHAPTER

THREE

VARIABLES INTRODUCTION

A **domain** is a collection or a range of possible values exhibiting a pattern. Objects belonging to the same domain have the same **type**. If an object x belongs to a domain D this is written mathematically as:

$$x \in D \quad \text{(Set theoretical notation)}$$

or equivalently as

$$x : D \quad \text{(Type theoretical notation)}$$

In C++:

```
D x;
```

This is called the **declaration** of **variable** x of the **data type** D.

A (mathematical) domain can be finite, countably infinite, or uncountanbly infinite.

Note: A **mathematical structure** is a domain equipped with distinguished **special elements**, **relations** and **operations**. Relations and operations are usually binary.

Example: Domain of integeres $\mathbb{Z}$ along with the special elements 0 and 1, order relation $<$, arithmetic operations $+, \times$.

$\mathbb{Z}$ **countably infinite**.

computers are finite, and physical components (the arithmetic and logic unit) performing the operations are capable of holding finitely many distinct representations of objects.

Therefore the mathematical structures realized by computer hardware are not $\langle \mathbb{Z}, 0, 1, +, \times$ and $\langle \mathbb{R}, 0, 1, +, \times \rangle$ but their finite approximations $\langle \tilde{\mathbb{Z}}, 0, 1, \oplus, \otimes \rangle$ and $\langle \mathbb{F}, 0, 1, \oplus, \otimes \rangle$, where the basic axioms are not always satisfied.

3.1 Basic Data Tytpes in C++

basic(atomic) data types of a programming langauge are the types directly provided by the language, as opposed to the data types defined by the programmer using the mechanisms of the langauge.

Some high-level programming langauges provide basic numerical data types that correspond to the ideal mathematical types $\mathbb{R}$ and $\mathbb{Z}$. But C++ provides only low level basic data types that are directly represented by the computer and directly operated on by the ALU. This data types are `int`, `float` and `double` correspong to $\tilde{\mathbb{Z}}$ and $\mathbb{F}$, respectively.

Type	Range	Implements	Represents
`int`	[-2^31^, 2^31^-1]	IEEE int	$\mathbb{Z}$
`unsigned int`	[0, 2^32^ - 1]	-	$\mathbb{N}$
`float`	[-3.4e38, 3.4e38]	IEEE float	$\mathbb{R}$
`double`	[-1.80e+308, 1.80e+308]	IEEE double	$\mathbb{R}$
`char`	ASCII characters	ASCII characters	letters and others
`string`	strings of ASCII	-	-

Table: List of Basic Data Types in C++

3.2 Variable Declarations

variables can be declared unitialized. Usually some default value like 0 is assigned to them.

```
unsigned int i;
std::cout << i << std::endl;
```

```
0
```

or initialized

```
double x(3.14);
float y(1.0);
short j(3);
std::cout << x << " " << y << " " << j << std::endl;
```

```
3.14 1 3
```

full program text:

```
#include <iostream>

int main()
{
    unsigned int i;
    double x(3.14);
    float y(1.0);
    short j(3);
    std::cout << x << " " << y << " " << j << std::endl;

}
```

3.3 Statements and Expressions

3.3.1 Statements

An object of a certain type can take different values during its existense. This transformation of values is called **change of state** of the object.

Computers can change the sate of an object residing in memory. This is called an **action**. The **instructions** to peform actions in a given programming language are called **statements**.

The change of variables state caused by a statement is said to be the **effect** of the statement.

3.3.2 Expressions

Expressions do not transform the state of the variables, but denote **values**.

Expressions are formed according to the rules of some formal notation (like the language of arithemtic)

Some expressions: `1 - (3 / 6)`, `1 + x * (y % 10)`

Thus:

- expressions **denote values**,
- statements **have effects**

Note: Understanding this dichotomy between statements and expessions is fundamental.

3.3.3 Assignment

The most basic statement is the **assignment**. It has the **effect** of updating the value of the variable to the value denoted by the **expression** on the right-hand side of the assignment operator. In C++:

```
#include <iostream>

int x(1);  //declare & initialize x
int y = x; //declare y and assign the value denoted by the expression "x" to y.

std::cout << x << " " << y << " " << std::endl;
```

```
1 1
```

The expression on the right hand side of the assignment can contain the variable being updated. Expression is evaluated before assignment:

```
y = (y * 3) + x; //assign the value denoted by (y * 3) + x to y.
std::cout << y << std::endl;
```

```
4
```

Side Effects

In some programming languages like Pascal the world of statements and the world of expressions are completely distinct. A given syntactical entity is either a statement or an expression.

But in C/C++ a syntactical entity can be both a statement (have an effect) and an expression (denote a value).

The action that such an expression performs is called the **side effect** of the expression.

Example: `j--` is both a statement and an expression. Its effect is to assign to the variable `j` the value of the expression `j - 1`. I.e. it is equivalent to `j = j - 1`. As an expression it denotes the value `j` before assignment.

```
int j = 10;

std::cout << j-- << std::endl; //side effect: decrement j.
std:: cout << j << std::endl;
```

```
10
```

```
9
```

Havinhg expression with side effects is academically less clean, but it practical benefits as it allows shorter programming constructs and succinct idioms like:

```
int i = 10;
while (i--){
    std::cout << i << " ";
}
```

```
9 8 7 6 5 4 3 2 1 0
```

Part II

Numerical Methods

CHAPTER

FOUR

INTRODUCTION

Introduction to Numerical Methods Notes.

INDEX