CS 101: Computer Programming and Utilization

Jan-Apr 2017

Sharat (piazza.com/iitb.ac.in/summer2017/cs101iitb/home)

Lecture 8B: Numbers (Continued)

About These Slides

- Based on Chapter 3 of the book
 An Introduction to Programming Through C++ by Abhiram Ranade (Tata McGraw Hill, 2014)
- Original slides by Abhiram Ranade

 First update by Varsha Apte
 Second update by Uday Khedker
 Third update by Sunita Sarawagi

Data Representation

What happens when you say float x = 23.2 double y = 1.3E27

Concluding Remarks

- Key idea 1: Current/charge/voltage values in the computer circuits represent bits (0 or 1).
- Key idea 2: Use numerical códes to represent non numerical entities
 - letters and other symbols: ASCII code
 - In fact, even the program written in "English" gets converted to numbers. So we have operations to perform on the computer and operation codes
- Key idea 3: Radix based system
 - Integers can be represented using sequence of bits. In a fixed number of bits you can represent positive integers in a fixed range.
 - If you dedicate a bit to representing the sign, the range of representable numbers changes.

Concluding Remarks

- Key idea 4:
 - Real numbers are represented approximately.
 - Because we need very large numbers and very small numbers, we cannot have a fixed location for the "decimal point" (or "binary point"). If you want more precision or greater range, you need to use larger number of bits.



Some Data Types Of C++

- unsigned int : Used for storing integers which will always be positive
 - 1 word (32 bits) will be allocated
 - Ordinary binary representation will be used
- char : Used for storing characters or small integers
 - 1 byte will be allocated
 - ASCII code of characters is stored
- float : Used for storing real numbers
 - 1 word will be allocated
 - IEEE FP representation, 8 bits exponent, 24 bits significand
- double : Used for storing real numbers
 - 2 words will be allocated
 - IEEE FP representation, 11 bits exponent, 53 bits significand

Variable Declarations

•Okay to define several variables in	
same statement	

- •The keyword long : says, I need to store bigger or more precise numbers, so give me more than usual space.
- •long unsigned int: Likely 64 bits will be allocated
- long double: likely 96 bits will be allocated

unsigned int telephone_number;

float velocity;

float mass, acceleration;

long unsigned int crypto_password;

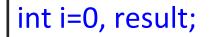
long double more_precise_vaule;

Variable Initialization

 Initialization - an INITIAL value is assigned to the variable

the value stored in the variable at the time of its creation

- -Variables i, vx, vy are declared and are initialized
- -2.0e5 is how we write 2.0*10⁵
- 'f' is a character constant representing the ASCII value of the quoted character
 result and weight are declared but not initialized



```
float vx=1.0,
vy=2.0e5,
weight;
```

```
char value = 'f';
```

Const Keyword

const double pi = 3.14;

The keyword const means : value assigned once cannot be changed

Useful in readability of a program

area = pi * radius * radius;

reads better than

area = 3.14 * radius * radius;

Reading Values Into Variables (1)

- Can read into several variables one after another
- If you read into a char type variable, the ASCII code of the typed character gets stored
- If you type the character 'f', the ASCII value of 'f' will get stored

```
cin >> noofsides;
cin >> vx >> vy;
char command;
cin >> command;
```

Reading Values Into Variables (2)

Some rules:

- User expected to type in values consistent with the type of the variable into which it is to be read
- Whitespaces (i.e. space characters, tabs, newlines) typed by the user are ignored.
- newline/enter key must be pressed after values are typed

An Assignment Statement

Used to store results of computation into a variable. Form: *variable_name = expression;*

Example:

s = u*t + 0.5 * a * t * t;

Expression : can specify a formula involving constants or variables, almost as in mathematics

- If variables are specified, their values are used.
- operators must be written explicitly
- multiplication, division have higher precedence than addition, subtraction
- multiplication, division have same precedence
- addition, subtraction have same precedence
- operators of same precedence will be evaluated left to right.
- Parentheses can be used with usual meaning

Arithmetic Between Different Types Allowed

int x=2, y=3, z, w;

float q=3.1, r, s;

r = x; // representation changed

// 2 stored as a float in r "2.0"

z = q; // store with truncation

// z takes integer value 3

s = x*q; // convert to same type,

// then multiply

// Which type?

Evaluating varA op varB e.g. x*q

- if varA, varB have the same data type: the result will have same data type
- if varA, varB have different data types: the result will have more expressive data type
- int/short/unsigned int are less expressive than float/double
- shorter types are less expressive than longer types

Rules for storing numbers of one type into variable of another type

- •C++ does the "best possible".
- int x; float y;
- x = 2.5;

y = 123456789;

•x will become 2, since it can hold only integers. Fractional part is dropped.

•123456789 cannot be precisely represented in 24 bits, so something like 1.234567 e 8 will get stored.

Compound Assignment

The fragments of the form sum = sum + expression occur frequently, and hence they can be shortened to sum += expression

Likewise you may have *=, -=, ...

Example

int x=5, y=6, z=7, w=8;

x += z; // x becomes x+z = 12

y *= z+w; // y becomes y*(z+w) = 90

Blocks and Scope

•Code inside {} is called a block.

•Blocks are associated with repeats; may create them otherwise too. You may declare variables inside any block.

•The variable **term** is defined close to where it is used, rather than at the beginning. This makes the program more readable.

```
// The summing
program
// written differently.
main_program{
  int s = 0;
 repeat(10){
    int term;
    cin >> term;
    s = s + term;
 cout << s << term <<
endl;
```

How definitions in a block execute

Basic rules

•A variable is defined/created every time control reaches the definition.

•All variables defined in a block are destroyed every time control reaches the end of the block.

•"Creating" a variable is only notional; the compiler simply starts using that region of memory from then on.

- •Likewise "destroying" a variable is notional.
- •New summing program executes exactly like the old, it just reads different (better!).

Shadowing and scope

- Variables defined outside a block can be used inside the block, if no variable of the same name is defined inside the block.
- If a variable of the same name is defined, then from the point of definition to the end of the block, the newly defined variable gets used.
- The new variable is said to "shadow" the old variable.
- The region of the program where a variable defined in a particular definition can be used is said to be the scope of the definition.