fit@hcmus

Fundamental Concepts

Basic Instructions

Allen Downey, in *How To Think Like A Computer Scientist*, writes:

*The details look different in different languages, but a few **basic instructions** appear in just about every language:*

- ***Input**: Gather data from the keyboard, a file, or some other device.*
- ***Output**: Display data on the screen or send data to a file or other device.*
- ***Arithmetic**: Perform basic arithmetical operations like addition and multiplication.*
- ***Conditional Execution**: Check for certain conditions and execute the appropriate sequence of statements.*
- ***Repetition**: Perform some action repeatedly, usually with some variation.*

Variables

Definition

- **Variables** are the *names* to computer *memory locations* used to **store** values.

- Some steps to use:
  - ***Create*** the variable with appropriate name.
  - ***Store*** value in the variable.
  - ***Retrieve*** and ***use*** the stored value from the variable.

Naming Variables

- Variable names are ***case sensitive***.
  `Hello` different from `hello`

- Contains only alphabetic letters, underscores or numbers.
- Should not start with a number.
- Cannot be any other keywords (`if, while, for, etc`).
- **Give your variables meaningful names!**

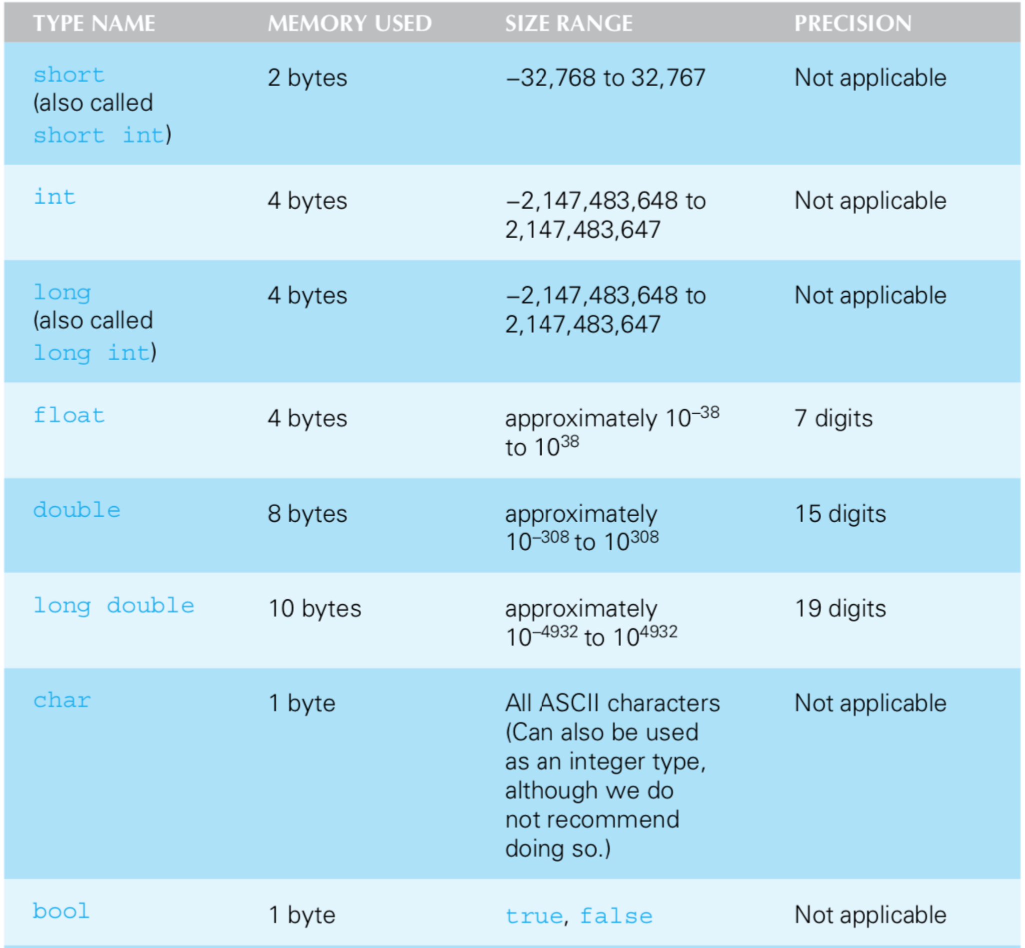
Data Types

- Data type: **set of values** together with a **set of operations**

- Different data types:
  - Simple (Number, Boolean, Character, etc)
  - Structured
  - Pointer

Data Types

- Integral data types
  - char
  - short
  - int
  - long
  - ***unsigned*** char
  - ***unsigned*** short
  - ***unsigned*** int
  - ***unsigned*** long
  - bool
- Floating-point number data ty



TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	–32,768 to 32,767	Not applicable
`int`	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

Arithmetic Operators

- Arithmetic operators:
  - `+` addition
  - `-` subtraction
  - `*` multiplication
  - `/` division
  - `%` modulus operator
- `+`, `-`, `*`, and `/` can be used with integral and floating-point data types
- Operators can be unary or binary

Variable Declaration

- Syntax (C/C++):

```
Type_Name Variable_Name_1, Variable_Name_2,..;
```

- Examples:
  - `int count, numberofDragons, numberofTrolls;`
  - `double distance;`

Variable Assignment

- Syntax (C/C++/Python/..):

```
Variable = Expression;
```

- Expression can be a variable, a number or a more complicated expression (made up of variables, numbers, operators, function invocations,..)

Variable Usage

- Examples

```
int num1, num2;
double sale;
char first;
num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
num2 = num1 + 27;
num2 = num1;
```

Simple Input - Output

Input

- Data must be loaded into main memory before it can be manipulated

- Storing data in memory is a two-step process:
  - Instruct computer to allocate memory
  - Include statements to put data into memory

Console Input/Output

- Using these objects: **std**::cin, **std**::cout, **std**::cerr of iostream

- Declaring before use:

```
#include <iostream>
//using namespace std;
```

Input Using `std::cin`

- `std::cin` is used with >> to gather input

```
std::cin >> variable1;
```

- The stream **extraction operator** is >>
- Using more than one variable in `std::cin` allows more than one value to be read at a time
- Examples:

```
std::cin  >> miles;
std::cin  >> numberofLanguages;
std::cin  >> dragrons >> trolls;
std::cin  >> dragrons
          >> trolls;
```

Output Using `std::cout`

- Any combinations of variables and strings can be output.

- `std::cout` is used with << to output.

```
    std::cout << expression or manipulator;
```

- The stream **insertion operator** is <<

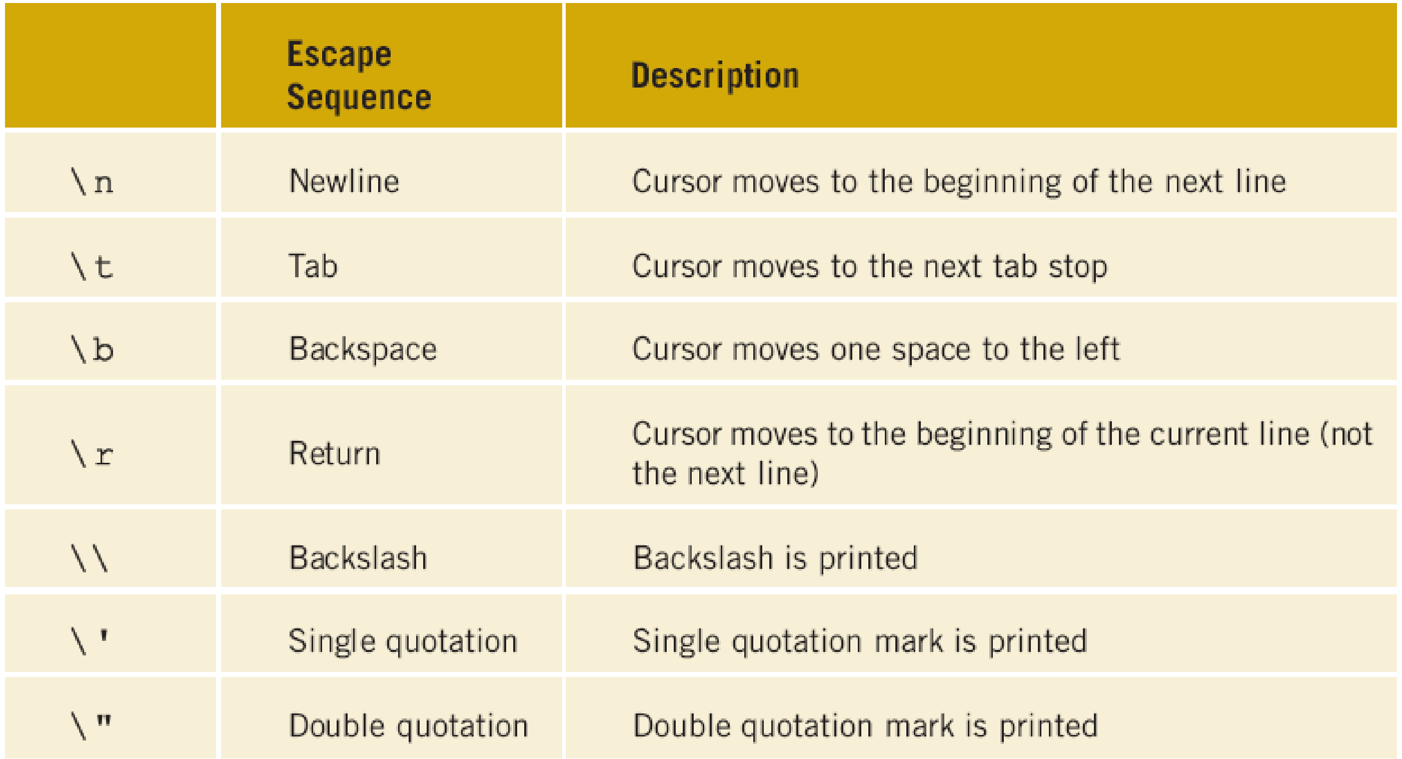
- Expression evaluated and its value is printed at the current cursor position on the screen.

Output Using `std::cout`

- The new line character is `'\n’`. May appear anywhere in the string.
- `std::endl` causes insertion point to move to beginning of next line.

Output Using `std::cout`

- Commonly used escape sequences:



	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

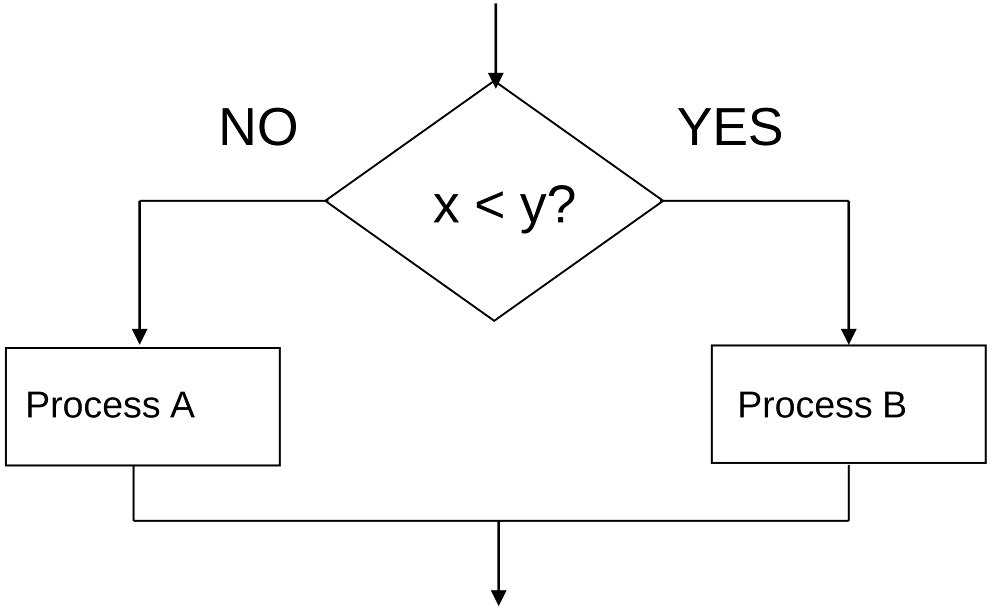
Condition Structures

Boolean Expression

- Boolean expression: an expression that is either ***true*** or ***false***.

- Comparison Operators: == , ! =, <, <=, > , >=

if-else Statements



if-else Statements

- Syntax:

```
if (Boolean_Expression)
   Yes_Statement


if (Boolean_Expression)
   Yes_Statement
else
   No_Statement
```

if-else Statements

- Syntax:

```
if (Boolean_Expression_1)
   Statement_1
else if (Boolean_Expression_2)
   Statement_2
...
else if (Boolean_Expression_n)
   Statement_n
else
   Statement_For_All_Other_Possibilities
```

switch Statement

```
switch (Controlling_Expression)
{
        case Constant_1:
                Statement_Sequence_1
                break;
        case Constant_2:
                Statement_Sequence_n
                break;
        ...
        case Constant_n:
                Statement_Sequence_n
                break;
        default:
                Default_Statement_Sequence
}
```

Repetition Structures

while Structure

- Single-statement body:

```
while (Boolean_Expression)
   Statement
```

- Multi-statement body:

```
while (Boolean_Expression)
{
   Statement_1
   Statement_2
   ...
   Statement_Last
}
```

do..while Structure

- Single-statement body:

```
do
   Statement
while (Boolean_Expression);
```

- Multi-statement body:

```
do
{
   Statement_1
   Statement_2
   ...
   Statement_Last
}while (Boolean_Expression);
```

for Structure

- The general form of the for statement is:

```
 for (Initialization_Action; Boolean_Expression;
Update_Action)
   Body_Statement
```

- The *Initialization_Action*, *Boolean_Expression*, and *Update_Action* are called for loop control statements
  - Initialization_Action usually initializes a variable (called the for **loop control**, or for **indexed**, **variable**)

Functions

Functions

- Allow complicated programs divided into manageable pieces.
- Some advantages of functions:
  - A programmer can focus on just that part of the program
    - construct, debug, and perfect it.
  - Different people can work on different functions simultaneously
  - Can be re-used (even in different programs)
  - Enhance program readability

Functions

- Other names:
  - Procedure
  - Subprogram
  - Method

- Types:
  - Pre-defined functions
  - User-defined (Programmer-defined) functions

Functions

```
<value returned/void> FunctionName (Parameter_List)
```

- `void` function: Function does not produce a value.
- Argument list: comma-separated list of parameters/arguments.
  - Can be empty

Pre-defined Functions

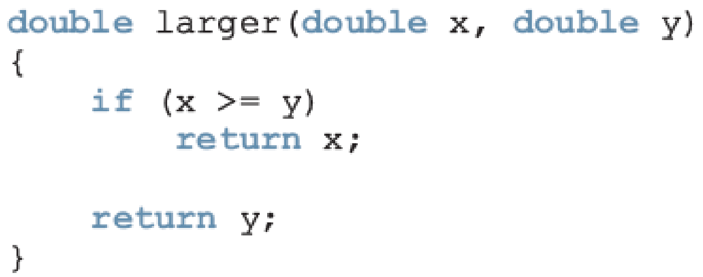
- Predefined functions are organized into separate libraries
  - I/O functions are in `iostream` header
  - Math functions are in `cmath` header
  - Some functions are in `cstdlib` header.
- Some of the predefined functions:

`sqrt``(x), cmath`: square root of `x`
`pow``(x, y), cmath`: `x` to the power of `y`
`floor``(x), cmath`: floor (round down) number `x`
`cos``(x),cmath`: cosine of angle `x`
`abs``(`**`x`**`), cstdlib`: absolute value of `x` (int)
`tolower``(c),cctype`: lowercase of `c`
`toupper``(c),cctype`: UPPERCASE of `c`

User-defined Functions

```
void FunctionName (Parameter_List)
{
…
}


<type> FunctionName (Parameter_List)
{
        …
        return expression;
}
```



```
double larger(double x, double y)
{
    if (x >= y)
        return x;

    return y;
}
```

Value vs Reference Parameters

- **Call-by-Value parameter:** a formal parameter that receives **a copy of the content** of corresponding actual parameter.
    - Can be variables or expressions.

- **Call-by-Reference parameter:** a formal parameter that receives **the location (memory address)** of the corresponding actual parameter.
    - Only be variables.

Call-by-Reference Parameters

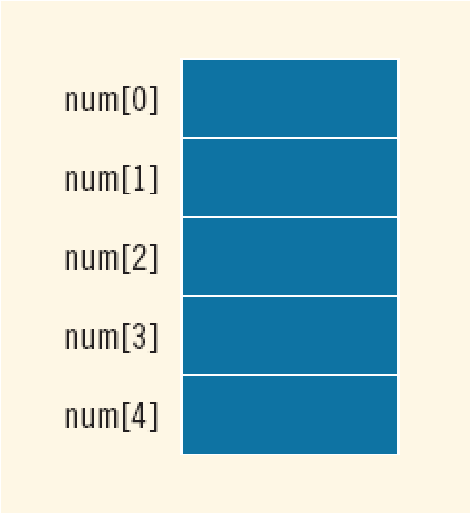
- Indicating the call-by-reference parameters by attaching the ampersand sign **&** at the of the type name in formal parameter list.

- Example:
  - `void getInput(double& N);`
  - `void sum(int N, int& s);`

Arrays

Arrays

- An array is a collection of items stored at **contiguous** memory locations.
- Elements can be **accessed randomly** using **indices** of an array.
- All elements must be the same data type.



- Used to represent **many instances** in one variable.

Arrays

- One-dimensional arrays
- Two-dimensional arrays
- Multi-dimensional arrays

One-dimensional Arrays

- Declaration:

```
        Data_Type ArrayName[ArraySize];
```

- Examples:

```
int numbers[10];
float grades[100];
```

- Usage:

```
numbers[1] = 2;
numbers[0] = 3 * numbers[1];
grades[8] = numbers[0] * 10/3.0;
```

Two-dimensional Arrays

- Declaration syntax:

```
Data_Type ArrayName[ROWSIZE][COLSIZE];
```

`ROWSIZE, COLSIZE:` positive integer values specify the **number of rows** and the **number of columns** in the array

- Examples:

```
int Array[8][10];
int Matrix[3][2] = {{1, 5}, {2, 4}, {3,9}};
```

- Usages:

```
Matrix[2][3] = Matrix[0][0]*7 + 2;
std::cout << Matrix[0][1];
```

Questions and Answers