cis15
advanced programming techniques, using c++
lecture # I.1
introduction

topics:

(0) introduction to the course

(1) to do

(2) review of c++

instructor:

- Arif Tuna Ozgelen, *ozgelen@sci.brooklyn.cuny.edu*

course web page:

- `http://www.sci.brooklyn.cuny.edu/~ozgelen/cis15`

(0) introduction to the course

- about this course
  - gives you more experience with C++
  - introduces advanced concepts, like recursion and object-oriented programming
  - introduces you to UNIX
- topics covered:

(I) Fundamentals
(II) Classes
(III) Specifications and Testing
(IV) Pointers and Memory
(V) Object-oriented Programming
(VI) Recursion
(VII) Templates

(0) course structure

- 7 **units**
- each unit has:
  - 1-2 **lectures**
  - 1-2 **labs**
  - 1 **assignment**
- the labs will be hands-on sessions using laptops
- your grade =
  - 7 assignments (53% total)
  - attendance and participation (7%)
  - midterm (15%)
  - final (25%)

(1) to do.

- get a copy of the textbook (C++ by Dissection, by Ira Pohl, published by Addison Wesley, 2001)
- Read chapters 1 & 2
- check out the class web page:
  `http://www.sci.brooklyn.cuny.edu/~ozgelen/cis15`

(2) review of c++.

(2.1) Basics

1. Programing Process
2. Compiling and Editing in UNIX
3. Algorithms and Style

(2.2) Language Fundamentals

1. Program Elements
2. Basic Input/Output
3. Types
4. Expressions
5. Control Flow

(2.1.1) Programming Process

- Computer programs are written to instruct machines to carry out a specific tasks or to solve specific problems. *Programming* is the activity of communicating to computers.
- Steps of programming in C++:
  1. *Editing* involves creating and modifying C++ codes (*source codes*). Source code files have extensions **.cpp* as in:

     `pgm.cpp`

     where *pgm* is the file name.
  2. *Compiling* refers to 3 separate events:
     - *preprocessor* includes other files to a copy of the source code
     - *compiler* translates the source code into object code
     - *linker* produces the executable file
  3. *Execution* refers to testing the program. If test fails source code has to be edited.

(2.1.3) Compiling and Editing in UNIX

- Many distributions and flavors of UNIX exist. We will use MacOS X.
- Editor: GNU EMacs, a powerful text editor for coding. Can be invoked from terminal.
- Compiler: `g++`, to invoke the compiler, type in the terminal window:

  ```
  > g++ pgm.cpp
  ```

  Once the compiler exits without error it will create an executable file

  ```
  > a.out
  ```

  This file will be overwritten every time the compiler is invoked. In order to avoid confusion, either rename or use `-o` option:

  ```
  > g++ -o pgm pgm.cpp
  ```

  This command will compile `pgm.cpp` and produce `pgm` rather than `a.out`. To run the program:

  ```
  > ./pgm
  ```

  If you want to interrupt the program at any point hit *control-c*

(2.1.2) Algorithms and Style

- **Algorithm**
  - al-Khwarizmi: arabic mathematician of the ninth century.
  - A step-by-step procedure that accomplishes a desired task is called an *algorithm*.
  - Algorithms should be precise, leaving no ambiguity and its instructions must terminate. It is desirable to design algorithms simple and effective.
- **Coding Style**
  - A good coding style is essential to the art of programming. It improves readability and maintenance of programs.
  - To do: Adopt a coding style and use it consistently!

```
#include <iostream>

 int main()
 {
     ...
 }
```

(2.2.1) Program Elements

- Program is composed of *tokens*, collections of characters.
  - Comments

```
// this is a single line comment
/*
 * this is a multiple
 * line comment
 */
```

  - Keywords - `for, while, void, using, double, break`, etc.
  - Identifiers - variable and function names, etc.
  - Literals - `5, 'a', "a", 5u, true`, etc.
  - Operators - arithmetic (`+, -, /, *, \%`), logical (`&&, ||`), assignment(`= , +=, *=`), etc.
  - Punctuators - `',', '\{'`, etc.

(2.2.2) Basic Input/Output

- Input/Output is not directly part of the C++ language, it is added in the standard library. This I/O library is `iostream` or `iostream.h`
- In order to access its types and routines, add the I/O library into your program simply by writing

```
#include <iostream>
```

  for input use >> operator and for output use << operator with streams:

```
cout    // standard out stream
cin     // standard in stream
```

  C++ standard library is in `namespace std` therefore the full name of the streams and their usage:

```
std::cout << "some text";
std::cin >> var_input    // string variable: var_input
```


(2.2.3) Simple Types

- Simple native types: `int, double, char, bool, wchar_t`
- Modifiers: `short, long, unsigned signed`
- Byte sizes increase from left to right: `bool, char, whcar_t, int, double` . Type sizes depend on the system, `sizeof()` operator is used to determine the storage size of type

```
cout << "int size = " << sizeof(int) << endl;
```

- Declaration and definition of a variable are conceptually separate things. Declaration refers to associating a variable name with type, but it does not allocate memory. Initialization does that by defining the variable

```
type id = expression
```


(2.2.3) Type Conversion

- Some operators can be applied to different types. When convertible, a copy of one of the operands is implicitly casted in to the type of the other operand.
- Rule of implicit conversion:
  - Any `bool, char, short, enum` is converted to `int`
  - The operand of the lower type is promoted to the higher type according to following hierarchy

```
int < unsigned < long < unsigned long
     < float < double < long double
```

  - Example:

```
int i;    short s;    double d;    char c;

c + 1.0
3 / 2
d * 3 - i
c - s / i
```

(2.2.3) Enumeration Types

- Used for declaring distinct integer type with a set of named integer constants called *enumerators*

```
enum suits { clubs, diamonds, hearts, spades } ;
// clubs = 0, diamonds = 1, hearts = 2, spades = 3
```

Enumerators can be initialized to constant expressions

```
enum ages { laura = 11, max, debra = 39, ira = debra + 7, robin }
// max = 12, ira = 46, robin = 47
```


(2.2.4) Expressions

- Precedence and Associativity: Operators have rules of precedence and associativity that determine how expressions are evaluated.

```
1 + 2 * 3
```

- Relational, equality and logical operators: C++ use `bool` values to direct the control of flow and these operators are used for this purpose

```
int a = 1, b = 2, c = 0;

a + 5 && b     or     ((a + 5) && b)    ?
!(a < b) && c    or    ((!(a < b)) && c)    ?
(a == b) || c    or    ((a == b) || c)     ?
```

Short circuit evaluation is an important feature ensuring to halt the evaluation as soon as the outcome is known.

```
expr1 && expr2
expr1 || expr2
```


(2.2.5) Control Flow

- if statement:

```
if (condition)
    statement;
```

- if-else

```
if(condition)
    statement_1;
else
    statement_2;
```

- else if

```
if(condition_1)
    statement_1;
else if (condition_2)
    statement_2;
else
    statement_3;
```


(2.2.5) Control Flow: break, continue, switch

- break: ends the control structure
- continue: ends the iteration
- switch

```
switch(condition) {
case constant_1:
    statement_1;
    break;    // optional: when used ends the switch statement
case constant_2:
    statement_2;
    break;    // optional
    ...
default:
    statement_n;
}
```

(2.2.5) Control Flow: Loops

- while loop

```
while(condition){
    statement;
}
```

- do - while

```
do{
    statement;
} while(condition)
```

- for loop

```
for ( init-statement; condition; expression )
    statement;
```