Lecture 02 - Input – Process – Output

0. Casual Discussion, Warm-up Questions, and Lecture Demo Exercises

I. The Temperature Program

```c++
/* Program written by Helena - 06/09/2008
Convert temperature in Celsius to Fahrenheit */
#include <iostream>
using namespace std;

void main()
{
    double C_temp, F_temp;
    //input
    cout << "Type in a temperature in Celsius: ";
    cin >> C_temp;
    F_temp = C_temp*9/5 + 32; //processing
    //output
    cout << "Temperature in Fahrenheit is: ";
    cout << F_temp << endl;
}
```

Comments:
```
/* .. */  ,    // ..
```

Variable definition:
```
Double C temp, F temp;    // ..
```
- allocates sufficient memory to hold values.

Input statement:
```
cin >> C temp;
```
- extracts a value from the input stream (ie. cin) and stores it in a variable (here C_temp).

Advanced knowledge:
1. There are many operators, eg. +, -, *, /.
   >> is also a kind of operator: input (or extraction) operator.
2. cin is an object, known as the standard input stream. Like the standard output stream (cout), both are defined in <iostream>
   (You will learn more about >> and objects in later part of this course.)

To input 2 or more values in 1 input statement:
```
char c1, c2;
cin >> c1 >> c2;
```
When the user strikes <enter>.
- the cursor will be brought to the next line, and;
- what he has typed will enter the input stream for extraction (>>) to proceed.
Assignment statement: \[ F_{\text{temp}} = C_{\text{temp}} \times \frac{9}{5} + 32; \]

- \[ C_{\text{temp}} \times \frac{9}{5} + 32 \] is an expression.
- \[ F_{\text{temp}} = \ldots \] The computed value is stored in \( F_{\text{temp}} \).
- Here \( = \) is called the Assignment operator.
  - It changes \( F_{\text{temp}} \) (left of \( = \)) to the given value (right of \( = \)).
    - It doesn’t make sense to write \( 3 = x + 15; \) (Reason: you can’t change 3 !!)
  - Left of \( = \) must be l-value (ie. a variable, modifiable memory location).

II. Variables, Constants, Operators & Expressions

Variables

- To store data (e.g. user input, intermediate result).
- Basic data types in C++:
  - `int`
  - `double`
  - `char`
  - `void`: no value
  - `bool`: Boolean values (true/false)

- Each variable must have a name.

An example: swap (exchange) 2 integers

```c++
#include <iostream>
using namespace std;

void main()
{
    int value1, value2;
    int original_value1;
    cout << "Input value1 and value2: ";
    cin >> value1 >> value2;
    original_value1 = value1;
    value1 = value2;
    value2 = original_value1;
    cout << "After swapping: " << value1 << " " << value2 << endl;
}
```

An expression is any meaningful combination of values and operators.
Example 1: \( 3 + 4 \) (but not \( + 3 \ 4 \))
Example 2: \( 200 \) (just a value)

An expression can be evaluated to get a resultant value.
Eg. the result of \( 3 + 4 \) is \( Z \).
the result of \( Z \) is \( Z \).

Given an integer \( x \), which one below gives compilation error?
(i) \( x = 3 + 4; \)
(ii) \( 3 + 4 = x; \)
Why is it wrong?

Data Type Ranges (Visual C++ Language Reference):

Different Data Types have different memory sizes, those can be system-dependent.

For curious students:
The name “double” is strange.
Why is it called “double”?
- See the answer at the course web (Useful Links + Notes)

A similar example:

Apple juice
Orange juice

Swap the drinks above
- need another empty cup as buffer
Identifiers
- An identifier is the name of a variable, (or other items)
  - eg. _C_temp, _F_temp, _main, _cout, _int, _if

**keywords or reserved words**
- Predefined identifiers in a programming language.
  - E.g.: _main, _cout, _int, _if

**User defined identifiers**
- Syntax:
  - first character: a letter or `_`
s   - remaining: letters, digits, or `_`
  - cannot be a keyword
- Legal e.g.: _x, _value1, nSubjects, tot_subjects, totSubjects
- Illegal e.g.: _9to5, _a&b

**Typical Encoding Schemes for different data types**
- **Int**
  - Nowadays mostly 4 bytes (32 bits)
  - Range: \(-2^{31}\) to \(2^{31}-1\) (ie. \(-2147483648\) to \(2147483647\) \(^{**}\))
    (If beyond this range => overflow error.)
  - Try _cout \ll 50000^5000_ and _cout \ll 50000^5000_.
- **Double** (a real number which may have a fractional part)
  - Known as **floating point number** \(^{##}\)
  - 8 bytes
  - Wide range: approx. \(-1.7\times10^{308}\) to \(+1.7\times10^{308}\) \(\heartsuit\)
  - But limited accuracy (eg. can't represent 0.7 exactly) \(^{^^}\)
- **char**
  - ASCII (American Standard Code for Information Interchange)
  - 1 byte

<table>
<thead>
<tr>
<th>Characters</th>
<th>'0'</th>
<th>'1'</th>
<th>...</th>
<th>'9'</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>48</td>
<td>49</td>
<td>...</td>
<td>57</td>
</tr>
<tr>
<td>Characters</td>
<td>'A'</td>
<td>'B'</td>
<td>...</td>
<td>'Z'</td>
</tr>
<tr>
<td>ASCII</td>
<td>65</td>
<td>66</td>
<td>...</td>
<td>90</td>
</tr>
<tr>
<td>Characters</td>
<td>'a'</td>
<td>'b'</td>
<td>...</td>
<td>'z'</td>
</tr>
<tr>
<td>ASCII</td>
<td>97</td>
<td>98</td>
<td>...</td>
<td>125</td>
</tr>
</tbody>
</table>

- Each variable must be defined with a type before use. eg. _int i_
  - The contents in these locations will be interpreted according to the type.
  - Example:

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

void main()
{
    char c;
    int i;

    c='a';
    i=97;
    cout \ll c \ll endl;
    cout \ll i \ll endl;
}
```

- Naming: _C_temp, x1, x2, c, ..._
  - A matter of taste?
- First requirement:
  - You yourself can quickly recognize what it means -- when you read the code in the future (eg. 1 year later).
- Advice to Students:
  - Think: Can the name help others to understand your code?
  - But NOT too long, like _int total_number_of_courses_in_this...

**Test your curiosity:**
1. **What if an integer value which is beyond this range?** Any other data type suitable?
   - Answer: the data type named _long long has a wider range_
2. **Named "floating point" because of the nature of the encoding scheme. (Learn later.)**
3. How does the computer represent _negative numbers_?
   - Answer: Signed number representation scheme: Two's complement
4. **Consider double _x=0.7_;** In the computer, values are stored as binary bits.
   - Convert 0.7 to binary, we get: _0.1011001100110011..._
   - (infinite digits!!)
   - Ref: Conversion to binary number: [http://www.digitconvert.com](http://www.digitconvert.com)
   - The 8-byte data type (double) cannot represent 0.7 accurately since it can't keep infinite digits!!
   - (No simple data type can do it!!)
   - Eg. Try "if (0.1+0.2+0.4==0.7)", you will get a false result.
   - The same problem happens to a lot of other values which cannot be converted exactly to binary.

For details of all above, you may refer to the links at course web (under Useful Links + Notes)
Constants

- "Constants" means fixed values.
  
  Examples:
  ```
  if (choice=='w')
    cout << "Hello";
  perimeter = 2 * radius * 3.14;
  ```

- Depending on the typed format, the compiler determines their data types:
  
  o An integer constant: eg. 2001 (DON'T add comma: 2,001, DON'T start with 0: 08)
  o A double eg. 3.14, 2.0
  o A character constant: use single quotes (e.g. 'A', '2', '%')

  Escape sequence
  ```
  \t the horizontal tab character
  \' the backslash character
  \" the double quote character
  \n eq. to endl
  ```

  o A string constant: use double quotes (eg. "abc", "Hello\nHow are you?\n")

Operators

Some types of operators:

- Arithmetic operators: +, -, *, /, %
- Assignment operators: =, +=, -=, *=, /=, %=
- Relational operators: >, ==, >=, <, <=, !=
- Boolean (Logical) operators: &&, ||
- Increment/Decrement operators: ++, --

Arithmetic Operators (+, -, *, /, %)

- **Division (/)**
  
  The result depends on the RHS and LHS operands: integer or double?
  
  o If both RHS and LHS operands are integers, it is treated as integral division:
    E.g., \( \frac{7}{4} \) gives 1; the decimal part (0.75) is discarded.
  
  o Otherwise (i.e. either or both RHS / LHS operand is double), floating point division is done: E.g., \( \frac{7.0}{4} \) gives 1.75; the decimal part is kept.

- **%** is the modulo operator
  
  You will know that % is very useful in programming!
  
  - e.g. \( 17 \% 5 \) gives 2 (%) applies on integers only)

- **Level of Precedence and use of ( )**
  
  - The usual precedence rules: first *, /, %, next + and -
  - Use ( ) when needed: E.g. \((x+y*2)^2\) - Note that we do NOT use [ ] or { }!

Facts about *, /, and %

(Visual C++ Language Reference)

- \( x \% y \) is eq. to \( x - (x/y) \times y \)
- Try -17\%5
  
  For %, if there is a -ve operand, the result is implementation dependent.
  
  - **Division by 0** (/ or %)
    
    (eg. 15/0, 28%0) is undefined and causes a run-time error!!

Memorization like "there are 5 .., the first one is .." is not needed.
We will know which one to use when we are given a question to solve.

However, it is now a good time to briefly recognize these terms.

How to say the following?

```
Mary said, "I'm hungry!"
Press any key to continue ...
```
Assignment Operator (=)  

- When we define variables, we can initialize them along using `=`
  
  E.g.  
  ```
  double rate=0.07, time, balance=0.0;
  ```

Efficient / Shorthand Assignments: `+=`, `*=` , `/=` , `%=`

Example 1:  
```
  cnt += 2;
```  
is equivalent to:  
```
  cnt = cnt + 2;
```  

Example 2: The Compound Interest Program

Given initial amount: $1000.00 and yearly interest: 0.04, show 5 years' results.

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

void main()
{

double start_amount=1000.00;

double yearly_interest=0.04;

int year=0;

double accum_amount=start_amount;

year=year+1;

accum_amount=accum_amount*(1+yearly_interest);

cout << "Year " << year <<": " << accum_amount << endl;

year=year+1;

accum_amount=accum_amount*(1+yearly_interest);

cout << "Year " << year <<": " << accum_amount << endl;

year=year+1;

accum_amount=accum_amount*(1+yearly_interest);

cout << "Year " << year <<": " << accum_amount << endl;

year=year+1;

accum_amount=accum_amount*(1+yearly_interest);

cout << "Year " << year <<": " << accum_amount << endl;

year=year+1;

accum_amount=accum_amount*(1+yearly_interest);

cout << "Year " << year <<": " << accum_amount << endl;

}
```

The above lines A and B can be rewritten as:

- **Line A'**  
  ```
  year++;  
  ```

- **Line B'**  
  ```
  accum_amount*= (1 + yearly_interest);  
  ```

Type Conversions

- General rule: do not assign data to variables of a different type
  
  ```
  int i = 2.99; // problem (warning) **
  ```

- `double x = 2; // OK`

- `char c = 65; // c gets the character with ASCII value 65, i.e. c becomes 'A'`

- `int i = 'Z'; // i gets the ASCII value of 'Z'

III. I/O Statements

Input Using cin (standard input stream): e.g.  
```
  cin >> x >> y;  
  ```

- Excess input from `cin` not “consumed” by the variables are kept in `cin` and are left for the next `>>` operation.

- Leading whitespaces are ignored (whitespaces mean tab, space, and new line)

- Numeric values are delimited by whitespaces.

** The compiler will warn you and set i: to 2 (discard 0.99 !!).
Output Formatting
- `<iostream>`: I/O facilities
- `<iomanip>`: formatting: fixed, setfill(..), setprecision(..), setw(..)

```cpp
Fixed-point Notation
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
    cout << fixed << setprecision(2)
         << 2343 << endl
         << 2343.0 << endl
         << 2343.346 << endl;
}
```

The Hourly Session Program:

```cpp
setw and setfill
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
    int start_hr;
    cout << "Input start hour: ";
    cin >> start_hr;

    cout << "The session is: " << setfill('0')
         << setw(2) << start_hr << ":00"
         << " to "
         << setw(2) << start_hr+1 << ":00"
         << endl;
}
```

setprecision(..) doesn't affect integer values.

```cpp
C:\WINDOWS\system32\cmd.exe
2343
2343.00
2343.35
Press any key to continue ...
```

```cpp
setw specifies the width of the next << operation. (Displayed contents are right-aligned.)
setw only works 1 time. So we need to repeat it for every output whenever needed.
setfill
If the specified width (by setw) is more than needed, fill character(s) will be added in-front.
The default fill character is <space>. We can use setfill to specify the fill character ourselves.
```

How about insufficient width?
consider
```cpp
cout << setw(2) << k;
```
but `k` is 2345 (ie. 4 digits!).
The output length will extend to fit all `k`'s digits (setw(2) is ignored.)

To Improve:
Suppose the start hour is 23, the output should show: 23:00-00:00.
However, the code above will show 23:00-24:00.
How can we fix it? [This question will be included in Lab02.]
IV. Checkpoints and Your Tasks

Summary / Check Point of this Lecture:
- Input / processing / output portions of a program
- Variables, data types, identifiers / naming
- Constants, Escape sequence,
- Arithmetic Operators and Assignment operators, ++
- if-else (see lecture exercise [green handout] )
- Input / Output and output formatting

Before next lecture:
- Review this lecture sheet in front of the computer (trying with Visual Studio along).
- Attend Lab and finish the Take-home exercises
- Read the textbook:
  (1) For review / strengthening : Chapter 2
  (2) For preparation of next lecture: Chapter 3 – up to and including Chp. 3.2 only.
- Preview the notes (print from course web) and exercise (given green handout) of Lecture 03.

--- end ---