Create Software Programming Assignment (week-2)

Every assignment will be announced on **Thursday** and should be sumitted by next **Tuesday**.

In this week **Handed out will be Sep 10, 2019, Due Sep 15, 2020**

Submit your assignment only through the hconnect GitLab

1. Your assignment should be `week-2` directory.
2. Commit & Push to the hconnect GitLab when you done.

1. I/O

C++ uses a convenient abstraction called streams to perform input and output operations in sequential media such as the screen, the keyboard or a file. A stream is an entity where a program can either insert or extract characters to/from. There is no need to know details about the media associated to the stream or any of its internal specifications. All we need to know is that streams are a source/destination of characters, and that these characters are provided/accepted sequentially (i.e., one after another).

The standard library defines a handful of stream objects that can be used to access what are considered the standard sources and destinations of characters by the environment where the program runs:

- `cin`: standard input stream
- `cout`: standard output stream
- `cerr`: standard error (output) stream
- `clog`: standard logging (output) stream

```
// file: io.cc
#include <iostream>

int main() {
    // print "Hello World!" through output stream (std::cout).
    // The std::endl manipulator can be used to break lines ('\n').
    std::cout << "Hello World!" << std::endl;

    // cout can print integer, string and other various types.
    std::cout << 3 << ", " << 2 << std::endl;

    // x is int type variable
    // Unlike Python, you need to declare a type of variable
    // Also cout can print value of variable
    int x = 3;
    std::cout << x << std::endl;

    // You can use `cin` to store the input value in a variable
    std::cin >> x;

    // And just printout inputed value that store in variable x
    std::cout << x << std::endl;
```

```
    // std::string is string type contains some characters.
    std::string str;

    // save inputed string to variable str
    std::cin >> str;

    std::cout << str << std::endl;

    return 0;
}
```

2. Pointer

For a C++ program, the memory of a computer is like a array of memory cells, each one byte in size, and each with a unique address. These single-byte memory cells are ordered in a way that allows data representations larger than one byte to occupy memory cells that have consecutive addresses.

This way, each cell can be easily located in the memory by means of its unique address. For example, the memory cell with the address 1776 always follows immediately after the cell with address 1775 and precedes the one with 1777, and is exactly one thousand cells after 776 and exactly one thousand cells before 2776.

When a variable is declared, the memory needed to store its value is assigned a specific location in memory (its memory address). Generally, C++ programs do not actively decide the exact memory addresses where its variables are stored. Fortunately, that task is left to the environment where the program is run - generally, an operating system that decides the particular memory locations on runtime. However, it may be useful for a program to be able to obtain the address of a variable during runtime in order to access data cells that are at a certain position relative to it.

```
// file: pointer.cc
#include <iostream>

int main()
{
    int var = 129;
    // int* is type for point of int
    // So, A* is type for point of A
    int *var_address = &var;
    int vluae_of_var_address = *var_address;

    std::cout << "Value of var: " << var << std::endl;
    std::cout << "Address of var: " << var_address << std::endl;
    std::cout << "Value of Address of var: " << vluae_of_var_address <<
std::endl;

    // pointer is just number of address
    int ary[3];
    ary[0] = 6;
    ary[1] = 23;
    ary[2] = 42;

    int *address_of_ary_0 = &ary[0];
    int *address_of_ary_1 = &ary[1];
    int *address_of_ary_2 = &ary[2];
```

```
    std::cout << "Address of ary[0]: " << address_of_ary_0 << std::endl;
    std::cout << "Address of ary[1]: " << address_of_ary_1 << std::endl;
    std::cout << "Address of ary[2]: " << address_of_ary_2 << std::endl;

    // The difference between the address value of the ary is 1
    std::cout << "Address of ary[2] - ary[1]: " << address_of_ary_2 -
address_of_ary_1 << std::endl;
    std::cout << "Address of ary[1] - ary[0]: " << address_of_ary_1 -
address_of_ary_0 << std::endl;

    // Therefore, adding 1 to the address value of ary[0] gives the value of
ary[1], and adding 2 gives the value of ary[2].
    std::cout << "The value of (address of ary[0]): " << *address_of_ary_0 <<
std::endl;
    std::cout << "The value of (address of ary[0] + 1): " << *
(address_of_ary_0+1) << std::endl;
    std::cout << "The value of (address of ary[0] + 2): " << *
(address_of_ary_0+2) << std::endl;

    return 0;
}
```

3. Structure

A data structure is a group of data elements grouped together under one name. These data elements, known as members, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```
struct type_name {
    int member_int;
    std::string member_string;
    .
    .
} object_names;
```

```
// file: structure.cc
#include <iostream>

// Conventionally, _t is appended to indicate that it is a type.
struct student_t {
    int id;
    std::string name;
} student_a, student_b;

int main() {
    // access to member using '.'
    student_a.id = 2019170229;
    student_a.name = "jiun";

    student_b.id = 1;
    student_b.name = "b";

    std::cout << "Student A, id: " << student_a.id << std::endl;
    std::cout << "Student A, name: " << student_a.name << std::endl;
```

```
    std::cout << "Student B, id: " << student_b.id << std::endl;
    std::cout << "Student B, name: " << student_b.name << std::endl;

    return 0;
}
```

reference

- http://www.cplusplus.com/