Practice 8
Class/Vector

2018 second semester
Computer Engineering Programming
CONTENTS

Class Constructor
Static & Constant
Vector
Class Constructor

• Constructors are used for initialization of objects (classes).
  – Can initialize some or all member variables.
  – Other actions possible as well.

• Constructors are automatically called when classes declared.

• Definition Example:

```cpp
class Student
{
public:
    Student(string student_id, string name);
    //Constructor initializes student_id and name.
    void set_studentInfo(...);
    ...
private:
    string student_id;
    string name;
    ...
};
```

**Cannot return a value; not even void!**

**Must same name as class**
Class Constructor Example 1

```cpp
#include <iostream>
#include <string>

using namespace std;

class Student {
public:
    Student(string i, string n, int a, int c);
    void print_studentInfo();
private:
    string student_id;
    string name;
    int age;
    int score;
};

int main()
{
    Student a("2017312724", "KIM MIN SU", 20, 80);
    a.print_studentInfo();
    return 0;
}

Student::Student(string i, string n, int a, int c)
{
    student_id = i;
    name = n;
    age = a;
    score = c;
}

void Student::print_studentInfo()
{
    cout << student_id << "\t" << name << "\t" << age << "\t" << score << "\n";
}
```

```cpp
#include <iostream>
#include <string>

using namespace std;

class Student {
public:
    Student(string i, string n, int a, int c);
    void print_studentInfo();
private:
    string student_id;
    string name;
    int age;
    int score;
};

int main()
{
    Student a("2017312724", "KIM MIN SU", 20, 80);
    a.print_studentInfo();
    return 0;
}

Student::Student(string i, string n, int a, int c)
{
    student_id = i;
    name = n;
    age = a;
    score = c;
}

void Student::print_studentInfo()
{
    cout << student_id << "\t" << name << "\t" << age << "\t" << score << "\n";
}
```
Class Constructor Example 2

```cpp
#include <iostream>
#include <string>

using namespace std;

class Student
{
public:
    Student(string i, string n, int a, int c);
    Student(string i, string n);
    void print_studentInfo();
private:
    string student_id;
    string name;
    int age;
    int score;
};

int main()
{
    Student a("2017312724", "KIM MIN SU", 20, 80);
    Student b("2015872123", "LEE CHANG MIN");
    a.print_studentInfo();
    b.print_studentInfo();
    return 0;
}

Student::Student(string i, string n, int a, int c)
    : student_id(i),
      name(n),
      age(a),
      score(c)
{
}

Student::Student(string i, string n)
    : student_id(i),
      name(n),
      age(20),
      score(0)
{
}

void Student::print_studentInfo()
{
    cout<<student_id<<"\t"<<name<<"\t"<<age<<"\t"<<score<<"\n";
}
```
Class Constructor Example 3

```cpp
#include <iostream>
#include <string>

using namespace std;

class Student {
public:
    Student(string i, string n, int a, int c);
    Student(const string i, const string n);
    Student();
    void print_studentInfo();
private:
    string student_id;
    string name;
    int age;
    int score;
};

int main()
{
    Student a("2017312724", "KIM MIN SU", 20, 80);
    Student b("2015872123", "LEE CHANG MIN");
    Student c;
    a.print_studentInfo();
    b.print_studentInfo();
    c.print_studentInfo();
    return 0;
}

Student::Student(const string i, const string n)
    : student_id(i), name(n), age(a), score(c);

Student::Student(string i, string n)
    : student_id(i), name(n)
}

void Student::print_studentInfo()
{
    cout<<student_id<<"\t"<<name<<"\t"<<age<<"\t"<<score<<"\n";
}
```

Explicit Constructor Calls for re-initialization

supported by C++11
Static & Constant

- **Static member variables**
  - All objects of class "share" one copy
  - One object changes it → all see change
  - ex) static int a;

- **Static Functions**
  - If no access to object data needed
  - And still "must" be member of the class
  - Can then be called outside class
  - Can only use static data and functions!

- **Constant member variables**
  - Cannot be changed after defined
  - If no need for function modifications
  - Protect parameter with const
  - ex) const int b;
Exercise

- Implement static function `get_total_student_num()`.
- Implement `constructor` which count up the `total_student_num`.

```cpp
#include <iostream>

using namespace std;

class Student
{
    public:
        static int get_total_student_num();
        Student();
    private:
        static int total_student_num;
};

int Student::total_student_num = 0;

int main()
{
    Student s1;
    Student s2;
    Student s3;
    cout << "current student num : " << Student::get_total_student_num() << endl;
    Student s4;
    Student s5;
    cout << "current student num : " << Student::get_total_student_num() << endl;
    return 0;
}```
• **Vectors** are arrays with dynamic size.
  - can grow and shrink *during program execution*
  - Recall that arrays are fixed size!!.

• Vectors are used similar to array, but differently declared.
  - `vector<Base_Type> variable_name`
    • Any type *can be "plugged in"* to Base_Type (int, double ...)
    • Example
      - `vector<int> v;`
```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> v_int;
    int num;
    int i = 0;

    while(1) {
        cout << "Enter a list of positive numbers (if you want to stop, enter -1): " << endl;
        cin >> num;
        if (num != -1) {
            v_int.push_back(num);
            cout << " is inserted in Vector." << endl;
            cout << "Current Vector Size is: " << v_int.size() << endl;
            i++;
        } else break;
    }

    cout << "Making list is completed..." << endl;
    cout << "Check the list!!" << endl;

    for (i = 0; i < v_int.size(); i++) {
        cout << v_int[i] << " ";
    }

    cout << endl;
    return 0;
}
```
Exercise

• Make a program that construct Student objects and store in vector. And you should implement vector searching function that search corresponding student from vector.

```cpp
// Get index of corresponding name from vector.
– Get_index_by_name(vector<Student> stdlist, string name)

// Get name of corresponding index from vector.
– Get_name_by_index(vector<Student> stdlist, int idx)
```
```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Student {
    public:
        Student(string i, string n, int a, int c);
        string GetName();
    private:
        string student_id;
        string name;
        int age;
        int score;
};

int Get_index_by_name(vector<Student> stdlist, string name);
string Get_name_by_index(vector<Student> stdlist, int idx);

int main() {
    vector<Student> stds;
    Student s1("2017123543", "LEE GWANG SU", 30, 15);
    Student s2("2014312123", "KIM MIN SU", 24, 89);

    stds.push_back(s1);
    stds.push_back(s2);

    cout << "Student KIM MIN SU's index is " << Get_index_by_name(stds, "KIM MIN SU") << endl;
    cout << "Second student of stds vector is " << Get_name_by_index(stds, 1) << endl;
```
Exercise

hsherlcok@uni:~/18_fall_cpp$ ./exec
Student KIM MIN SU's index is 1
Second student of stds vector is KIM MIN SU