STL 잘 사용하기 #1 – std::vector

TAMREF

기초적인 사용을 위해서는 1~5를, 심화 구문까지 알고 싶으시다면 그 이후까지 읽으시면 됩니다.

1. 준비물...

   벡터는 C++의 <vector> 헤더에 정의되어 있습니다. 즉 사용하기 위해선 코드의 맨 위에 #include <vector>를 추가하면 됩니다.

   다만 Visual Studio 계열의 IDE를 사용하지 않는 이상, <stdio.h>를 포함한 모든 헤더를 한 데 묶어놓은 헤더인 <bits/stdc++.h>를 사용하는 것을 권장합니다. 훨씬 간단한 코드를 짤 수 있으니까요.

   이 글에서 다루고 있는, 앞으로의 글에서 다룰 STL의 사용법은 극히 일부에 지나지 않습니다. 유연한 프로그래밍을 하기 위해서는 STL의 활용법을 더 적극적으로 알아볼 것을 권장합니다.

   http://en.cppreference.com 등의 reference에 익숙해지는 것이 가장 좋고, 아래 그림처럼 IDE에서 제공하는 기능들을 보고 사용법을 익혀도 됩니다.

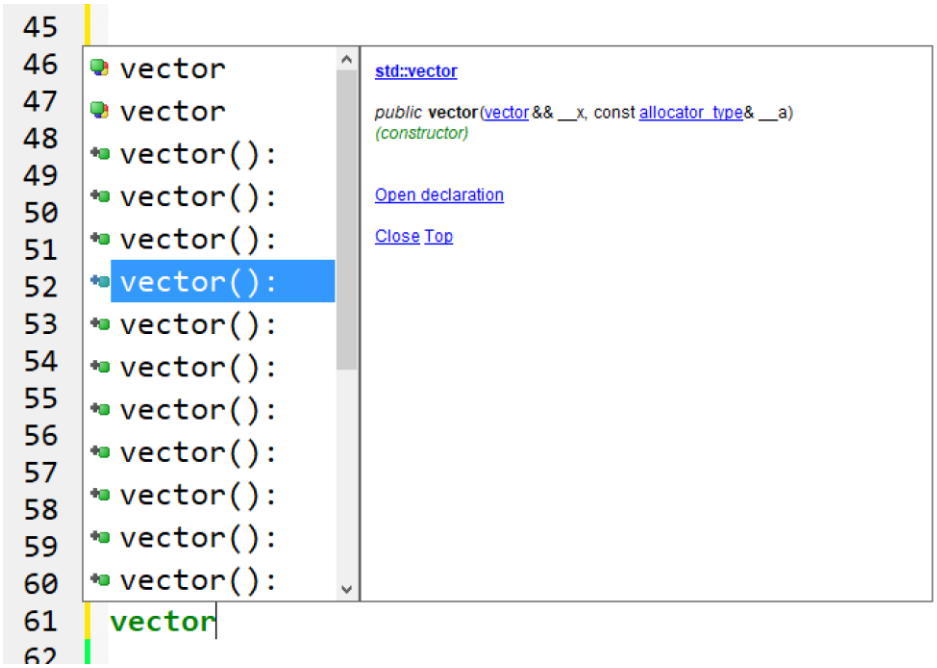

Figure 1 IDE에서는 수많은 형태의 vector 선언 방법, 함수 사용 방법 등에 대한 간단한 설명을

제공합니다. 심심할 때 읽어두면 여러 상황에서 편리합니다.

2. 벡터의 선언

벡터를 선언하는 방법은 [Figure 1]에서 보셨듯 여러 가지가 있지만, 가장 기본적인 몇 가지 형태만 다루려고 합니다.

```
60
61  vector<int> prime; //empty vector
62  vector<double> prime; //double - typed empty vector
63  vector<int> prime(1000); //size 1000 vector, filled with 0
64  vector<int> prime(5,1); //size 5 vector, filled with 1
65  vector<int> prime({2,3,5,7,11,13}); //size 6 vector. element : 2 3 5 7 11 13
66
```

벡터 선언문의 기본적인 구조는 아래와 같습니다. 다른 STL들도 이 구조에서 크게 벗어나지는 않아요.

$$\text{vector} < \text{T} > \text{Name}(\text{other operation});$$

엄밀히는 클래스 생성자를 호출하는 것이지만... 이 개념은 클래스 프로그래밍에 익숙해지게 되면 받아들일 수 있을 거고, 지금은 외웁시다.

T : 담고자 하는 자료의 type입니다. int, double, long long, ... 혹은 내 입맛대로 만든 구조체 타입 이름이 들어갈 수 있습니다.

Name : 벡터의 이름입니다. 일반적인 변수의 선언 규칙과 같아요.

Other operation : 위 그림과 같이 편의를 위한 추가적인 옵션입니다. 위 그림과 주석을 참조하세요.

여담이지만, 다음과 같은 구문도 쓸 수 있습니다. 생각보다 C++은 유연하거든요.

```
vector<vector<int>> V; //벡터가 들어있는 벡터
vector<int> V2[5]; //5개짜리 벡터 배열
vector<vector<int>> V3(5,vector<int>(5,0)); //5 * 5짜리 이차원 벡터
```

(주의) C++11 옵션을 체크하지 않을 경우, vector<vector<int>> 구문은 컴파일 에러가

납니다. 되도록 C++11을 쓰고, C++03을 너무 사랑해서 포기할 수 없는 경우에는 vector< vector<int> >와 같이 한 칸 띄어서 써 주세요.

3. 벡터의 원소 삽입, 삭제

사실상 벡터의 가장 핵심적인 개념이 되겠습니다.

벡터 원소 삽입은 매우 간단합니다.

다음과 같이 push_back(원하는 원소)를 호출해 주면 됩니다.

다만 line 13 ~ 16과 같은 실수를 범하지 않게 주의하세요.

```
 5  int main()
 6  {
 7      vector<int> G;
 8      G.push_back(5); //G : 5
 9      for(int i = 0; i < 5; i++){
10          G.push_back(i);
11      } // G : 5 0 1 2 3 4
12
13      vector<int> H(5); //H : 0 0 0 0 0
14      for(int i = 0; i < 3; i++){
15          H.push_back(i);
16      }//H : 0 0 0 0 0 0 1 2
17  }
18
```

원소 삭제도 매우 간단합니다. Pop_back()을 호출하면 되거든요.

하지만 **빈 벡터에는 절대로 pop_back()을 호출하지 않도록 주의하세요.** 런타임 에러가 발생할 수 있습니다.

```cpp
int main()
{
    vector<int> G;
    G.pop_back(); //DO NOT TRY THIS... (underflow occurs)

    vector<int> H;
    for(int i = 0; i < 5; i++) H.push_back(5 - i); //H : 5 4 3 2 1

    for(int i = 0; i < 2; i++) H.pop_back(); //H : 5 4 3
}
```

4. 벡터의 원소 참조

벡터의 원소 참조는 배열과 거의 동일합니다. 꺾쇠 괄호를 이용해서 i번째 원소를 참조할 수 있거든요. 다만 벡터의 크기를 넘어가는 위치를 참조하면 쓰레기 값을 뱉거나, 런타임 에러를 던지고 프로그램이 종료될 수 있습니다.

```cpp
int main()
{
    vector<int> H;
    for(int i = 0; i < 5; i++) H.push_back(5 - i); //H : 5 4 3 2 1

    for(int i = 0; i < 2; i++) H.pop_back(); //H : 5 4 3
    cout << H[3] << '\n'; //do not try this... (runtime error occurs)

    for(int i = 0; i < (int)H.size(); i++) cout << H[i] <<'\n';
}
```

원소 참조에 유용한 함수들은 다음과 같습니다. 보시는 바와 같이 사용하시면 되어요.

```cpp
    cout << (int)H.size() << '\n'; //3, H.size() : unsigned int type
    cout << H.back() << '\n'; //3, last element in vector

    H.back() = 8; //H : 5 4 8
    H.size() = 8; ///DO NOT TRY THIS... (Compile Error occurs)
    H.resize(8); //ok. H : 5 4 8 0 0 0 0 0
}
```

5. Iterator

벡터뿐만 아니라 대부분의 STL에서 사용되는 구문인 iterator는... 복잡하지만 일단은 **포인터**와 같다고 생각하시면 됩니다. 대신 &v[i]와 같은 구문을 쓸 일은 거의 없고, begin()와 end()를 아래와 같이 사용하는 편입니다.

```
7     vector<int> H;
8     for(int i = 0; i < 5; i++) H.push_back(5 - i); //H : 5 4 3 2 1
9
10    for(int i = 0; i < 2; i++) H.pop_back(); //H : 5 4 3
11
12    for(auto it = H.begin(); it != H.end(); it++){
13        cout << *it << ' '; //prints 5 4 3
14    }
15
16    sort(H.begin(), H.end()); //H : 3 4 5
17    reverse(H.begin() + 1, H.end()); // reverses [ H[1], H[3](end) ), H : 3 5 4
18 }
19
```

대부분의 STL 함수는 기본적으로 begin(), end()에 맞춰서

F(B,E) : [B, E)의 구간에 뭔가 한다.

의 형태로 구현되어 있는 것이 대부분입니다. 직접 굴러보면서 사용합시다.

6. Foreach문 (range – based for)

일반적으로 벡터 전체를 순회하기 위해선 아래와 같은 구문을 쓰는데... 좀 길죠?

```
7     vector<int> H;
8     for(int i = 0; i < 5; i++) H.push_back(5 - i); //H : 5 4 3 2 1
9
10    for(int i = 0; i < 2; i++) H.pop_back(); //H : 5 4 3
11    for(int i = 0; i < H.size(); i++){
12        H[i] += i * i;
13    }// H : 5 5 7
```

그래서 line 14 ~ 15와 같이 줄여서 사용할 수 있어요.

```
7     vector<int> H;
8     for(int i = 0; i < 5; i++) H.push_back(5 - i); //H : 5 4 3 2 1
9
10    for(int i = 0; i < 2; i++) H.pop_back(); //H : 5 4 3
11    for(int i = 0; i < H.size(); i++){
12        H[i] += i * i;
13    }// H : 5 5 7
14
15    for(int &u : H) u *= 2; //H : 10 10 14
```

이를 foreach문, 또는 range-based for문이라고 합니다. H[i]를 u로 쓸 수 있고, 귀찮은 인덱스 표기를 생략할 수 있다는 점에서 편하지만, i가 필요한 경우에는 살짝 빡칩니다...