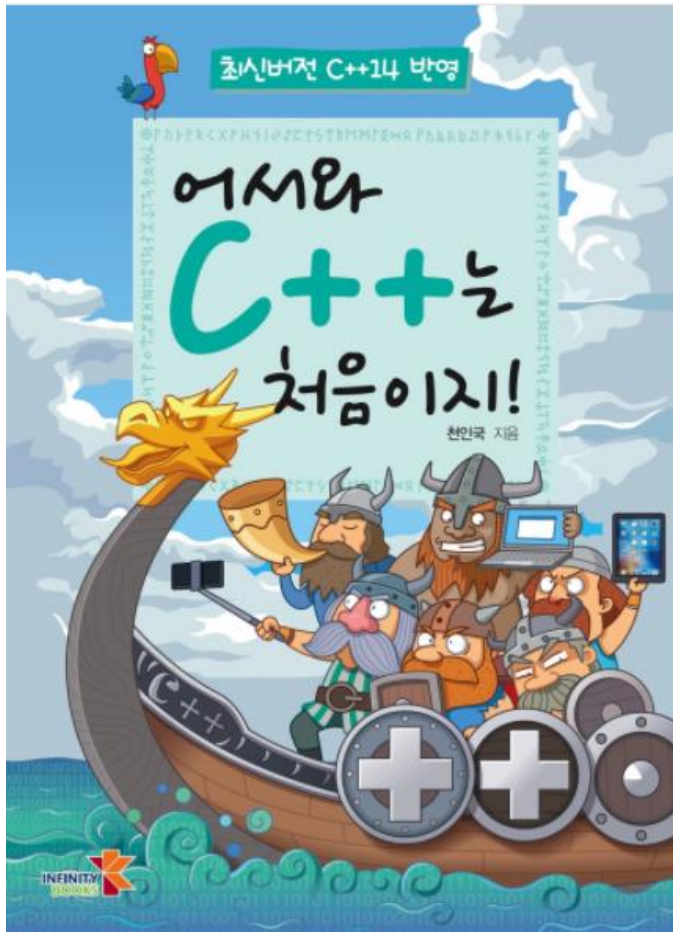


제6장 객체배열과 벡터



- 벡터(**vector**)는 동적 배열이다.
- 컴파일 시간에 배열의 크기를 미리 결정할 필요가 없다.

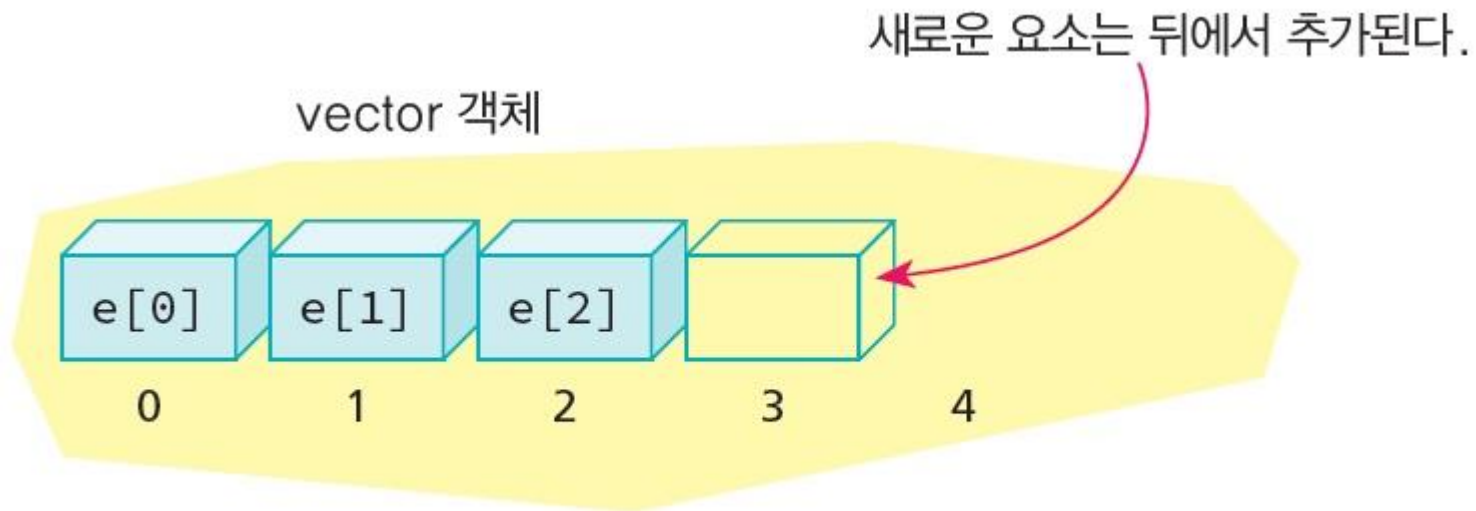


그림 6.2 벡터

iterator(반복자)

`begin()` : beginning iterator를 반환

`end()` : end iterator를 반환

추가 및 삭제

`push_back(element)` : 벡터 제일 뒤에 원소 추가

`pop_back()` : 벡터 제일 뒤에 원소 삭제

`insert(iter, element)` : 벡터에서 `iter` 위치 앞에 `element` 삽입

`erase(iter)` : 벡터에서 `iter`가 가리키는 원소를 삭제

조회

`[i]` : `i`번째 원소를 반환

`at(i)` : `i`번째 원소를 반환

`front()` : 첫번째 원소를 반환

`back()` : 마지막 원소를 반환

기타

`empty()` : 벡터가 비어있으면 `true` 아니면 `false`를 반환

`size()` : 벡터 원소들의 수를 반환

배열과의 차이

동적으로 원소를 추가할 수 있으며 크기가 자동으로 늘어난다.

벡터의 선언

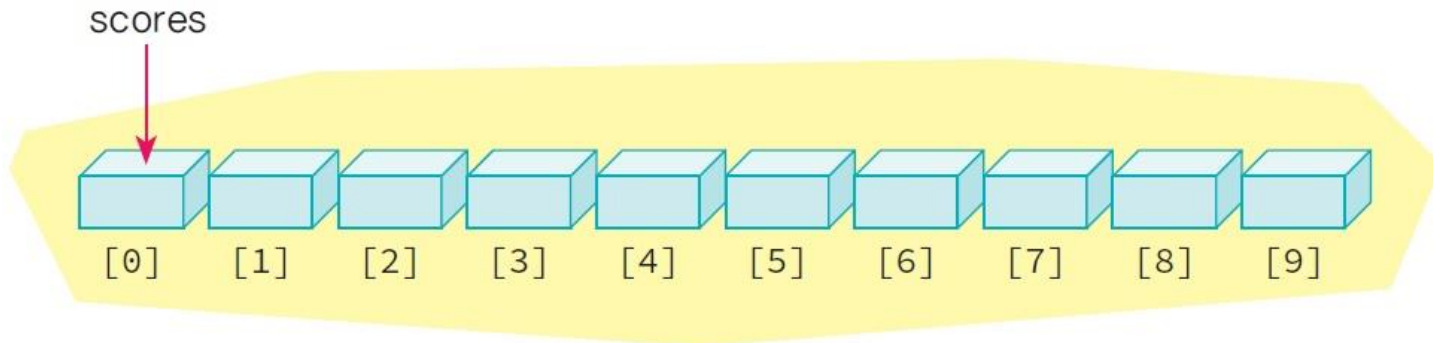
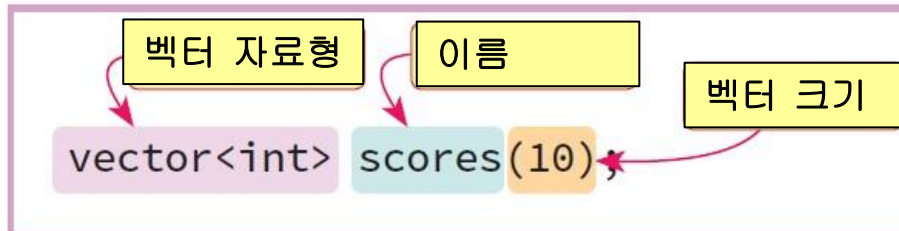
벡터 선언

vector<자료형> 변수이름

vector<자료형> 변수이름 (크기)

문법 6.1

벡터 선언



벡터의 사용

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

int main(void)
{
    vector<int> fibonacci { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };

    for (int i = 0; i < fibonacci .size(); ++i)
        cout << fibonacci[i] << ' ';

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

C:\Windows\system32\cmd.exe

0 1 1 2 3 5 8 13 21 34 55 89
계속하려면 아무 키나 누르십시오 . . .


벡터의 사용

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

int main(void)
{
    vector<int> fibonacci { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };

    for (auto& number : fibonacci)
        cout << number << ' ';

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



C:\Windows\system32\cmd.exe

0 1 1 2 3 5 8 13 21 34 55 89
계속하려면 아무 키나 누르십시오 . . .

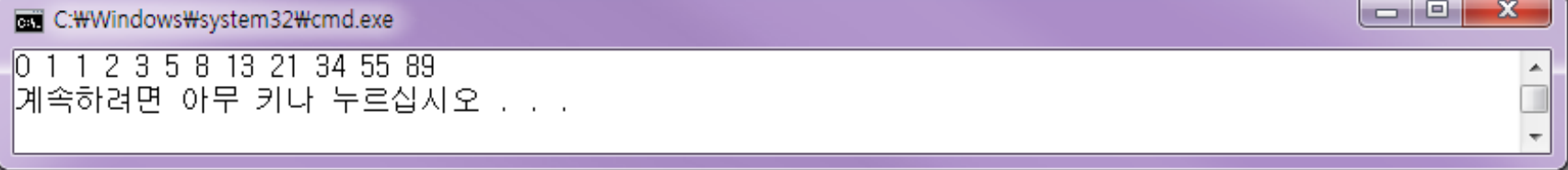
벡터의 사용

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

int main(void)
{
    vector<int> fibonacci { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };
    vector<int>::iterator it;

    for (it = fibonacci.begin(); it != fibonacci.end(); ++it)
        cout << *it << ' ';

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



C:\Windows\system32\cmd.exe

0 1 1 2 3 5 8 13 21 34 55 89
계속하려면 아무 키나 누르십시오 . . .

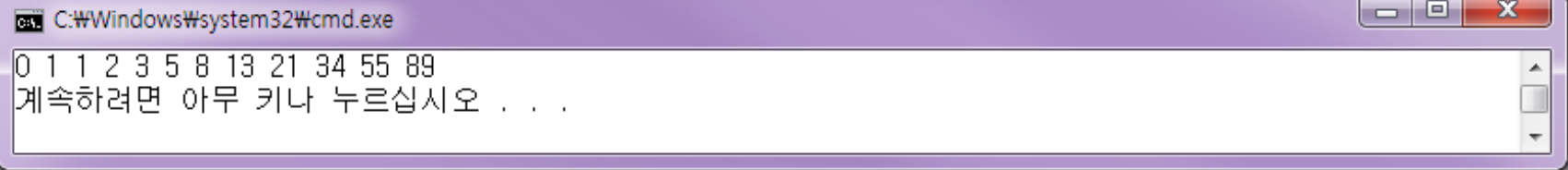
벡터의 사용

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

int main(void)
{
    vector<int> fibonacci { 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };

    for (auto it = fibonacci.begin(); it != fibonacci.end(); ++i)
        cout << *it << ' ';

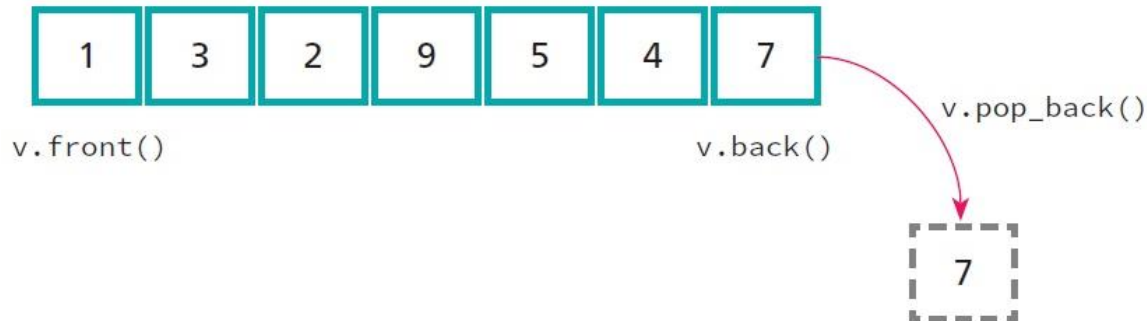
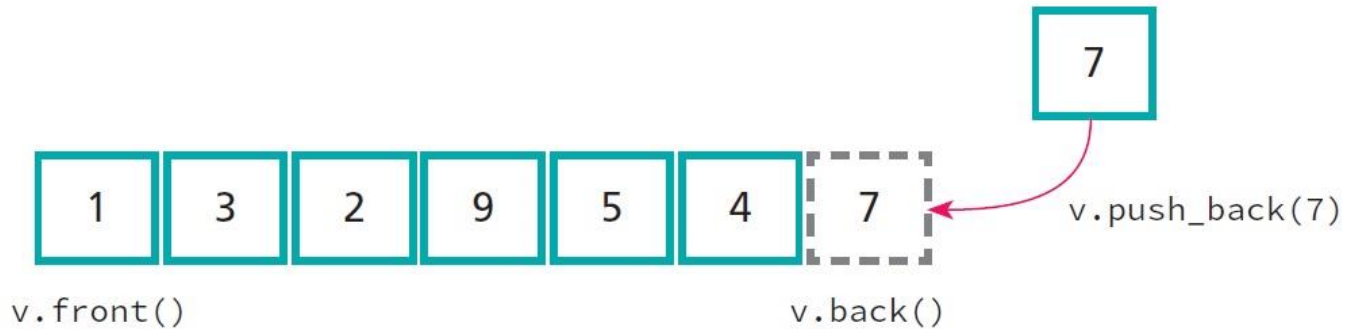
    cout << endl;
    return 0;
}
```



C:\Windows\system32\cmd.exe

0 1 1 2 3 5 8 13 21 34 55 89
계속하려면 아무 키나 누르십시오 . . .

push_back()과 pop_back()



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

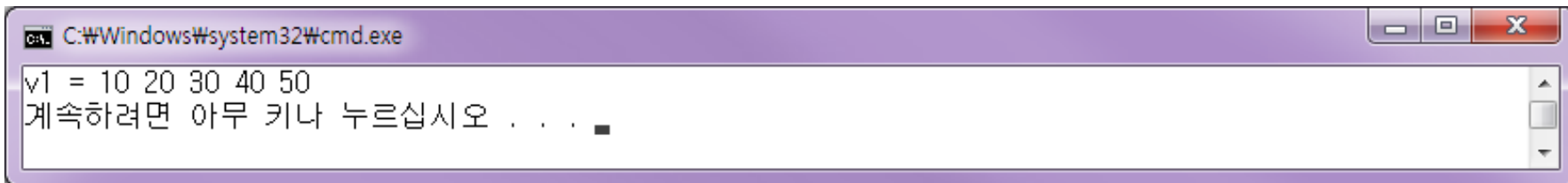
int main(void)
{
    vector<int> v1;

    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    cout << "v1 = ";
    for (auto& e : v1) {
        cout << e << " ";
    }
    cout << endl;
    return 0;
}
```

실행결과

결과



```
C:\Windows\system32\cmd.exe
v1 = 10 20 30 40 50
계속하려면 아무 키나 누르십시오 . . .
```

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

int main(void) {
    vector<int> v;
    for (int i = 0; i < 10; ++i) {
        v.push_back(i);
    }

    cout << "현재의 v = ";
    for (auto& e : v)
        cout << e << " ";
    cout << endl;

    cout << "삭제 요소 = ";
    // 벡터가 공백이 될 때까지 pop_back() 호출
    while (v.empty() != true) {
        cout << v.back() << " ";
        v.pop_back();
    }
    cout << endl;
}
```

실행결과



```
C:\Windows\system32\cmd.exe
현재의 v = 0 1 2 3 4 5 6 7 8 9
삭제 요소 = 9 8 7 6 5 4 3 2 1 0
계속하려면 아무 키나 누르십시오 . . .
```

벡터에서 요소의 위치

- 벡터에서 요소의 위치는 반복자(iterator)를 이용하여 표시한다.



```
vector<int> v;  
.....  
vector<int>::iterator it;  
for (it = v.begin(); it != v.end(); ++it)  
    cout << *it << endl;
```

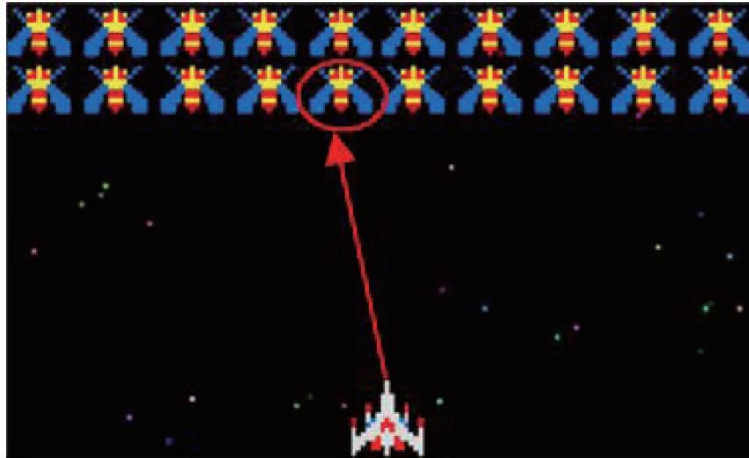
```
vector<int> v;  
.....  
  
for (auto it = v.begin(); it != v.end(); ++it)  
    cout << *it << endl;
```

벡터의 중간에서 삭제

특정 위치에 있는 원소를 삭제

`erase(it);` // `it`는 iterator – `it` 위치 원소 삭제

`erase(start, end);` // `start`, `end`는 iterator – `start`에서 `end` 까지 원소들을 삭제



예: `v.erase(v.begin()+i);`

벡터의 중간에 삽입

특정 위치에 새로운 원소를 삽입

`insert(it,element);` // it는 iterator, element는 삽입되는 원소

예: `v.insert(v.begin()+i, 10);`

벡터와 연산자

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

int main(void)
{
    vector<int> v1{ 1, 2, 3, 4, 5 };
    vector<int> v2(v1);

    if (v1 == v2) {
        cout << "2개의 벡터가 일치합니다. " << endl;
    }
    return 0;
}
```

cmd C:\Windows\system32\cmd.exe

2개의 벡터가 일치합니다.
계속하려면 아무 키나 누르십시오 . . .

벡터에 문자열 저장

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

int main(void)
{
    vector<string> vec;                                // 벡터를 생성한다.

    vec.push_back("MILK");                               // 벡터의 끝에 자료를 저장한다.
    vec.push_back("BREAD");
    vec.push_back("BUTTER");
    for (auto e : vec) {
        cout << " " << e;
    }
    cout << endl;
    return 0;
}
```

벡터와 알고리즘

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;

class Person {
private:
    string name;
    int age;
public:
    Person::Person(string n, int a)    {
        name = n;
        age = a;
    }
    string get_name() { return name; }
    int get_age()     { return age; }
    void print()      {
        cout << name << " " << age << endl;
    }
};
```

벡터와 알고리즘

```
bool compare(Person &p, Person &q)
{
    return p.get_age() < q.get_age();
}

int main()
{
    vector<Person> list;

    list.push_back(Person("Kim", 30));
    list.push_back(Person("Park", 22));
    list.push_back(Person("Lee", 26));

    sort(list.begin(), list.end(), compare);

    for (auto& e : list) {
        e.print();
    }
    return 0;
}
```

실행결과



```
C:\Windows\system32\cmd.exe
Park 22
Lee 26
Kim 30
계속하려면 아무 키나 누르십시오 . . .
```

Lab: 성적평균 계산하기

- 학생들의 평균 성적을 계산하는 예제에서 학생이 몇 명인지 알 수 없다고 하자. 동적 배열인 벡터를 이용하여서 작성해보자.



```
C:\Windows\system32\cmd.exe
성적을 입력하시오(종료는 -1) : 10
성적을 입력하시오(종료는 -1) : 20
성적을 입력하시오(종료는 -1) : 30
성적을 입력하시오(종료는 -1) : 40
성적을 입력하시오(종료는 -1) : 50
성적을 입력하시오(종료는 -1) : -1
성적 평균=30
계속하려면 아무 키나 누르십시오 . . .
```

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

int main() {
    vector<int> scores;           // int 동적 배열을 생성한다.
    int i, sum = 0;

    while (true) {
        int score;
        cout << "성적을 입력하시오(종료는 -1) : ";
        cin >> score;
        if (score == -1) break;
        scores.push_back(score);
    }

    for (auto& value : scores) {
        sum += value;
    }
    double avg = (double)sum / scores.size();
    cout << "성적 평균=" << avg << endl;

    return 0;
}
```

- **vector**는 생성과 소멸을 하는데 상당한 시간이 소요된다. 따라서 **vector**의 장점이 많지만 성능 때문에 기존의 배열을 사용하는 경우도 많다.