제13장 템플릿과 STL

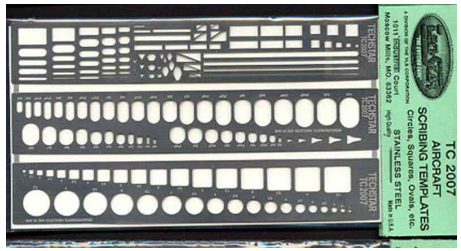
이번 장에서 학습할 내용

- 함수 템플릿
- 클래스 템플릿
- 스택 예제

일반적인 하나의 코드로 다양한 자료형을 처리하는 기법을 살펴봅시다.

템플릿이란?

- 템플릿(template): 물건을 만들 때 사용되는 틀이나 모형을 의미
- 함수 템플릿(function template): 함수를 찍어내기 위한 형틀



함수 get_max()

```
int get_max(int x, int y)
{
    if( x > y ) return x;
    else return y;
}
```

만약 float 값중에서 최대값을 구하는 함수가 필요하다면?

```
float get_max(float x, float y)
{
    if( x > y ) return x;
    else return y;
}
```

핵심적인 내용은 같고 매개 변수의 타입만 달라진다.

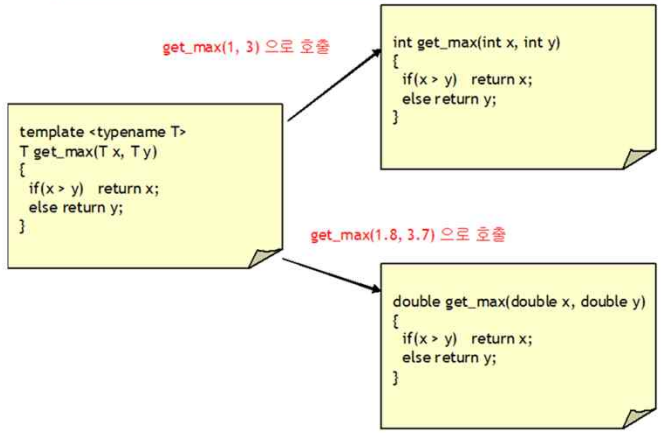
일반화 프로그래밍

- **generic programming:** 일반적인 코드를 작성하고 이 코드를 다양한 타입의 객체에 대하여 재사용하는 프로그래밍 기법

```
template<typename T>
T get_max(T x, T y)
{
    if( x > y ) return x;
    else return y;
}
```

자료형이 변수처럼 표기되어 있음을 알 수 있다

템플릿 함수의 사용



예제

get_max.cpp

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

template <typename T>
T get_max(T x, T y)
{
  if(x > y)    return x;
  else return y;
}

int main()
{
  // 아래의 문장은 정수 버전 get_max()를 호출한다.
  cout << get_max(1, 3) << endl;

  // 아래의 문장은 실수 버전 get_max()를 호출한다.
  cout << get_max(1.2, 3.9) << endl;

  return 0;
}
```

실행 결과

```
3
3.9
계속하려면 아무 키나 누르십시오 . . .
```

템플릿 함수의 특수화

```
template <typename T>                  // 함수 템플릿으로 정의
void print_array(T[] a, int n)
{
    for(int i=0;i<n; i++)
        cout << a[i] << " ";
    cout << endl;
}

template <>                            // 템플릿 특수화
void print_array(char[] a, int n)      // 매개 변수가 char인 경우 이 함수 호출
{
    cout << a << endl;
}
```

함수 템플릿과 함수 중복

swap_values.cpp

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

template <typename T>
void swap_values(T& x, T& y)
{
   T  temp;
   temp = x;
   x = y;
   y = temp;
}
void swap_values(char* s1, char* s2)
{
   int  len;

   len = (strlen(s1) >= strlen(s2)) ? strlen(s1) : strlen(s2);
   char* tmp = new char[len + 1];

   strcpy(tmp, s1);
   strcpy(s1, s2);
   strcpy(s2, tmp);
   delete[] tmp;
}
```

템플릿함수

함수 중복

함수 템플릿과 함수 중복

```
int main()
{
   int x=100, y=200;
   swap_values(x, y);              // x, y가 모두 int 타입- OK!
   cout << x << " " << y << endl;

   char s1[100]="This is a first string";
   char s2[100]="This is a second string";
   swap_values(s1, s2);            // s1, s2가 모두 배열 - 오버로딩 함수 호출
   cout << s1<< " " << s2<< endl;
   return 0;
}
```

실행 결과

```
200 100
This is a second string This is a first string
계속하려면 아무 키나 누르십시오 . . .
```


두개의 타입 매개 변수

```
template<typename T1, typename T2>
void copy(T1 a1[], T2 a2[], int n)
{
   for (int i = 0; i < n; ++i)
      a1[i] = a2[i];
}
```


중간 점검 문제

1. 변수의 절대값을 구하는 int abs(int x)를 템플릿 함수로 정의하여 보자.
2. 두수의 합을 계산하는 int add(int a, int b)를 템플릿 함수로 구현하여 보자.
3. displayArray()라는 함수는 배열을 매개 변수로 받아서 반복 루프를 사용하여서 배열의 원소를 화면에 출력한다. 어떤 타입의 배열도 처리할 수 있도록 함수 템플릿으로 정의하여 보라.

클래스 템플릿

- 클래스 템플릿(class template): 클래스를 찍어내는 틀(template)

```
template <typename 타입이름, ...> class 클래스이름
{
...

}
```

- 예제: 하나의 값을 저장하고 있는 박스

예제

```
class Box {
    int data;
public:
    Box() { }
    void set(int value) {
        data = value;
    }
    int get() {
        return data;
    }
};

int main()
{
  Box box;
  box.set(100);
  cout << box.get() << endl;
  return 0;
}
```

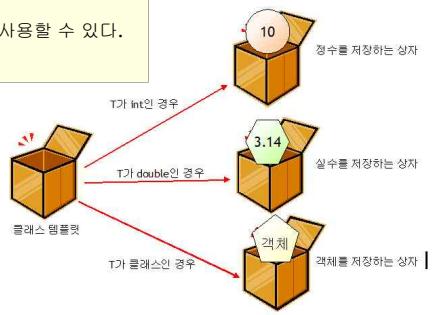
클래스 템플릿으로 만들어 보자.

실행 결과

```
100
계속하려면 아무 키나 누르십시오 . . .
```


클래스 템플릿 버전

```
template <typename T>
class 클래스이름
{
...// T를 어디서든지 사용할 수 있다.
}
```



예제

```
template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸
public:
    Box() {      }
    void set(T value) {
        data = value;
    }
    T get() {
        return data;
    }
};
```

```
int main()
{
  Box<int> box;
  box.set(100);
  cout << box.get() <<endl;

  Box<double> box1;
  box1.set(3.141592);
  cout << box1.get() <<endl;

  return 0;
}
```

정수 버전

실수 버전

실행 결과

```
100
3.14159
계속하려면 아무 키나 누르십시오 . . .
```

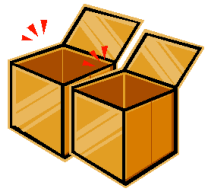
클래스 외부에 정의

```
template <typename T>
class Box {
    T data; // T는 타입(type)을 나타낸다.
public:
    Box();
    void set(T value);
    T get();
};
```

```
template <typename T>
Box<T>::Box() {
}

template <typename T>
void Box<T>::set(T value) {
   data = value;
}

template <typename T>
T Box<T>::get() {
   return data;
}
```


두개의 타입 매개 변수

- 두 개의 데이터를 저장하는 클래스 Box2



Box2 클래스 템플릿

```
template <typename T1, typename T2>
class Box2 {

   T1 first_data; // T1은 타입(type)을 나타낸다.
   T2 second_data; // T2는 타입(type)을 나타낸다.

public:
   Box2() {      }
   T1 get_first();
   T2 get_second();
   void set_first(T1 value) {
        first_data = value;
   }
   void set_second(T2 value) {
        second_data = value;
   }
};
```


예제

```
template <typename T1, typename T2>
T1 Box2<T1, T2>::get_first() {
   return first_data;
}
template <typename T1, typename T2>
T2 Box2<T1, T2>::get_second() {
   return second_data;
}
int main()
{
   Box2<int, double> b;
   b.set_first(10);
   b.set_second(3.14);
   cout << "(" << b.get_first() << ", " << b.get_second() << ")" << endl;
   return 0;
}
```

실행 결과

```
(10, 3.14)
계속하려면 아무 키나 누르십시오 . . .
```

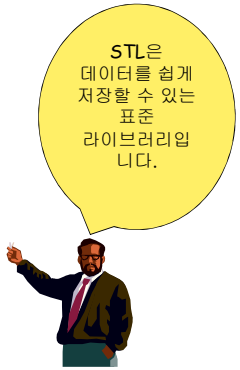

중간 점검 문제

1. 클래스 템플릿 형태로 라이브러리를 제공하면 어떤 장점이 있는가?

2. 세개의 데이터를 가지고 있는 Triple라는 클래스를 클래스 템플릿으로 작성하여 보라.

이번 장에서 학습할 내용

- STL의 소개
- 벡터
- 덱크
- 리스트
- 집합
- 맵
- 스택
- 큐



STL

- STL: 표준 템플릿 라이브러리(Standard Template Library)의 약자로서 많은 프로그래머들이 공통적으로 사용하는 자료 구조와 알고리즘에 대한 클래스

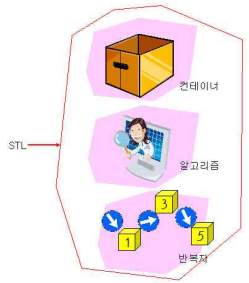


그림 18-1 STL

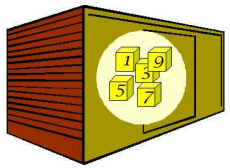
STL의 3가지 컴포넌트

- 컨테이너(container)
  - 자료를 저장하는 구조이다.
  - 벡터, 리스트, 맵, 집합, 큐, 스택과 같은 다양한 자료 구조들이 제공된다.
- 반복자(iterator)
  - 컨테이너 안에 저장된 요소들을 순차적으로 처리하기 위한 컴포넌트
- 알고리즘(algorithm)
  - 정렬이나 탐색과 같은 다양한 알고리즘을 구현

STL의 장점

- STL은 전문가가 만들어서 테스트를 거친 검증된 라이브러리
- STL은 객체 지향 기법과 일반화 프로그래밍 기법을 적용하여서 만들어졌으므로 어떤 자료형에 대해서도 적용
- STL을 사용하면 개발 기간을 단축할 수 있고 버그가 없는 프로그램

컨테이너



컨테이너

그림 18-2 컨테이너의 개념

분류	컨테이너 클래스	설명	헤더 파일
순차 컨테이너	vector	벡터처럼 입력된 순서대로 저장	<vector>
	list	순서가 있는 리스트	<list>
	deque	양끝에서 입력과 추출이 가능	<deque>
연관 컨테이너	set	수학에서의 집합 구현	<set>
	multiset	다중 집합(중복을 허용)	<set>
	map	사전과 같은 구조	<map>
	multimap	다중 맵(중복을 허용)	<map>
컨테이너 어댑터	stack	스택(후입선출)	<stack>
	queue	큐(선입선출)	<queue>
	priority_queue	우선순위큐(우선순위가 높은 원소가 먼저 추출)	<queue>

순차 컨테이너

- 순차 컨테이너:
  - 자료를 순차적으로 저장
  - 벡터(vector): 동적 배열처럼 동작한다. 뒤에서 자료들이 추가된다.
  - 데크(deque): 벡터와 유사하지만 앞에서도 자료들이 추가될 수 있다.
  - 리스트(list): 벡터와 유사하지만 중간에서 자료를 추가하는 연산이 효율적이다.

연관 컨테이너

- 연관 컨테이너
  - 사전과 같은 구조를 사용하여서 자료를 저장
  - 원소들을 검색하기 위한 키(key)
  - 자료들은 정렬
  - 집합(set): 중복이 없는 자료들이 정렬되어서 저장된다.
  - 맵(map): 키-값(key-value)의 형식으로 저장된다. 키가 제시되면 해당되는 값을 찾을 수 있다.
  - 다중-집합(multiset): 집합과 유사하지만 자료의 중복이 허용된다.
  - 다중-맵(multimap): 맵과 유사하지만 키가 중복될 수 있다.

컨테이너 어댑터

- 컨테이너 어댑터
  - 순차 컨테이너에 제약을 가해서 데이터들이 정해진 방식으로만 입출력
  - 스택(stack): 먼저 입력된 데이터가 나중에 출력되는 자료 구조
  - 큐(queue): 데이터가 입력된 순서대로 출력되는 자료 구조
  - 우선 순위큐(priority queue): 큐의 일종으로 큐의 요소들이 우선순위를 가지고 있고 우선 순위가 높은 요소가 먼저 출력되는 자료 구조

반복자

- 반복자(iterator)
  - 현재 처리하고 있는 자료의 위치를 기억하는 객체
  - 포인터와 유사
  - * 연산자 사용 가능
  - ++ 연산자 사용 가능

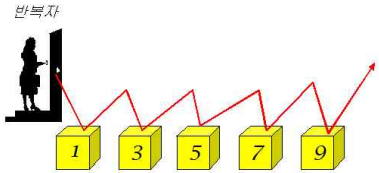


그림 18.3 반복자는 컨테이너에 저장된 요소들을 순차적으로 방문한다.

알고리즘

- 탐색(find): 컨테이너 안에서 특정한 자료를 찾는다.
- 정렬(sort): 자료들을 크기순으로 정렬한다.
- 반전(reverse): 자료들의 순서를 역순으로 한다.
- 삭제(remove): 조건이 만족되는 자료를 삭제한다.
- 변환(transform): 컨테이너의 요소들을 사용자가 제공하는 변환 함수에 따라서 변환한다.

그림 18.4 알고리즘의 예

벡터

- 벡터 == 동적 배열 == 스마트 배열
- 템플릿으로 설계

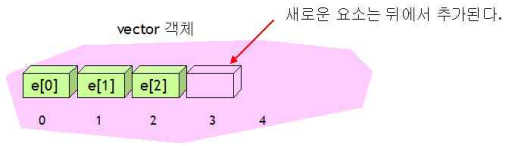
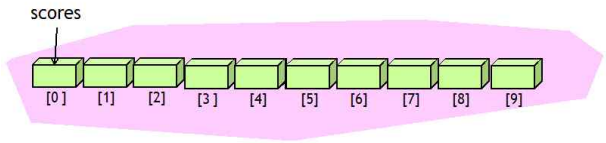


그림 18.5 벡터

기본 예제

- 학생들의 성적을 저장하는 벡터를 생성하여 보자.



기본 예제

vector1.cpp

```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
   vector<double> scores(10);        // 벡터를 생성한다.

   for(int i = 0; i < scores.size() ; i++)
   {
        cout << "성적을 입력하시오: ";
        cin >> scores[i];
   }

   double highest = scores[0];
   for(int i = 1; i < scores.size() ; i++)
        if( scores[i] > highest )
                highest = scores[i];
   cout << "최고 성적은 " << highest << "입니다.\n";

   return 0;
}
```


실행 결과

실행 결과

```
성적을 입력하시오: 10
성적을 입력하시오: 20
...
성적을 입력하시오: 90
성적을 입력하시오: 100
최고 성적은 100입니다.
```


push_back()과 pop_back()

- push_back()
  - 새로운 데이터를 벡터의 끝에 추가하고 벡터의 크기를 1만큼 증가
- pop_back()
  - 벡터의 끝에서 요소를 제거하고 벡터의 크기를 하나 감소

예제

vector2.cpp

```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
   vector<double> scores;            // 벡터를 생성한다.

   while(true)
   {
        double value = 0.0;
        cout << "성적을 입력하시오(종료는 -1): ";
        cin >> value;
        if( value < 0.0 ) break;
        scores.push_back(value);
   }
```

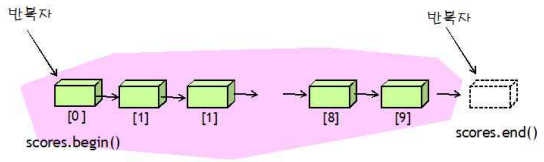
예제

```
    double highest = scores[0];
    for(int i = 1; i < scores.size() ; i++)
        if( scores[i] > highest )
            highest = scores[i];
    cout << "최고 성적은 " << highest << "입니다.\n";

    return 0;
}
```

실행 결과

```
성적을 입력하시오(종료는 -1): 10
성적을 입력하시오(종료는 -1): 20
성적을 입력하시오(종료는 -1): 30
성적을 입력하시오(종료는 -1): -1
최고 성적은 30입니다.
```


반복자의 사용



예제

```
#include <iostream>
#include <vector>                    // 벡터를 사용하려면 이 헤더 파일을 포함하여야 한다
using namespace std;

int main()
{
    vector<double> scores;           // 벡터를 생성한다.

    while(true)
    {
        double value = 0.0;
        cout << "성적을 입력하시오(종료는 -1): ";
        cin >> value;
        if( value < 0.0 ) break;
        scores.push_back(value);
    }
```


예제

```
    double highest = -100;
    vector<double>::iterator it;
    for(it = scores.begin(); it < scores.end() ; it++)
        if( *it > highest )
            highest = *it;


    cout << "최고 성적은 " << highest << "입니다.\n";

    return 0;
}
```

실행 결과

```
성적을 입력하시오(종료는 -1): 10
성적을 입력하시오(종료는 -1): 20
성적을 입력하시오(종료는 -1): 30
성적을 입력하시오(종료는 -1): -1
최고 성적은 30입니다.
```

벡터와 연산자

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

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

   vec.push_back("MILK");                  // 벡터의 끝에 자료를 저장한다.
   vec.push_back("BREAD");
   vec.push_back("BUTTER");

   vector<string>::iterator it;     // 벡터를 순회하기 위해여 반복자를 선언한다.
   for(int i=0;i<vec.size();i++)
      cout << vec[i] << " ";        // [] 연산자 사용
   cout << endl;
```

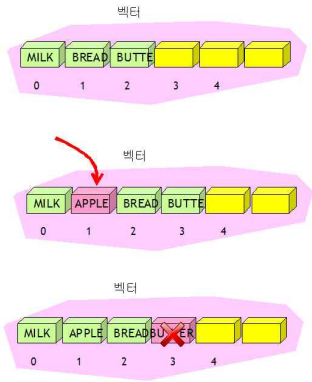

벡터와 연산자

```
   vec.insert(vec.begin()+1, "APPLE");      // 벡터의 첫부분에 자료를 저장한다.
   vec.pop_back();                  // 벡터의 끝에서 자료를 삭제한다.

   for (it = vec.begin(); it != vec.end(); ++it)
        cout << *it << " ";
   cout << endl;

   return 0;
}
```

실행 결과

```
MILK BREAD BUTTER
MILK APPLE BREAD
계속하려면 아무 키나 누르십시오 . . .
```


실행 결과

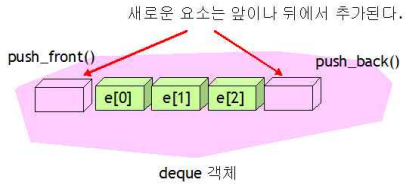


컨테이너의 공통 함수

함수	설명
Container()	기본 생성자
Container(size)	크기가 size인 컨테이너 생성
Container(size, value)	크기가 size이고 초기값이 value인 컨테이너 생성
Container(iterator, iterator)	다른 컨테이너로부터 초기값의 범위를 붙여서 생성
begin()	첫 번째 요소의 반복자 위치
clear()	모든 요소를 삭제
empty()	비어있는지를 검사
end()	반복자가 마지막 요소를 지난 위치
erase(iterator)	컨테이너의 중간 요소를 삭제
erase(iterator, iterator)	컨테이너의 지정된 범위를 삭제
front()	컨테이너의 첫 번째 요소 반환
insert(iterator, value)	컨테이너의 중간에 value를 삽입
pop_back()	컨테이너의 마지막 요소를 삭제
push_back(value)	컨테이너의 끝에 데이터를 추가
rbegin()	끝을 나타내는 역반복자
rend()	역반복자가 처음을 지난 위치
size()	컨테이너의 크기
operator=(Container)	할당 연산자의 중복 정의

데크

- 데크의 경우, 전단과 후단에서 모두 요소를 추가하고 삭제하는 것을 허용



예제

```
int main()
{
   deque<int> dq;
   dq.push_back(99);
   dq.push_back(1);
   dq.push_front(35);
   dq.push_front(67);

   for(int i=0;i<dq.size();i++)
        cout << dq[i] << " ";          // [] 연산자 사용
   cout << endl;
   dq.pop_back();
   dq.pop_front();
   for(int i=0;i<dq.size();i++)
        cout << dq[i] << " ";          // [] 연산자 사용
   cout << endl;

   return 0;
}
```


리스트

- 외부에서 보면 벡터와 동일
- 이중 연결 리스트로 구현

그림 18.7 배열의 중간에 삽입하려면 원소들을 이동하여야 한다.

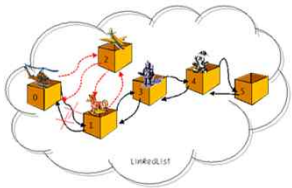


그림 18.8 연결 리스트 중간에 삽입하려면 링크만 수정하면 된다.

예제

```
void print_list(list<int>& li);
int main()
{
   list<int> my_list;

   my_list.push_back(10);
   my_list.push_back(20);
   my_list.push_back(30);
   my_list.push_back(40);

   my_list.insert(my_list.begin(), 5);
   my_list.insert(my_list.end(), 45);
   print_list(my_list);
   return 0;
}

void print_list(list<int>& li)
{
   list<int>::iterator it;
   for(it=li.begin(); it!=li.end(); ++it)
        cout << *it << " ";
   cout << endl;
}
```

실행 결과

실행 결과

```
5 10 20 30 40 45
계속하려면 아무 키나 누르십시오 . . .
```


집합

- 수학적인 집합과 비슷
- 집합(set)은 동일한 키를 중복해서 가질 수 없다.
- (예) A = { 1, 2, 3, 4, 5 }는 집합이지만 B = { 1, 1, 2, 2, 3 }은 집합이 아니다.

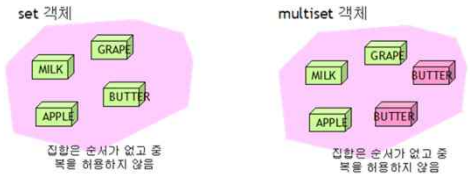


그림 18-9 집합과 다중집합

예제

```
template  <typename T>
void print_list(const T& container);

int main()
{
   set<int> my_set;
   multiset<int> my_multiset;

   my_set.insert(1);
   my_set.insert(2);
   my_set.insert(3);

   my_multiset.insert(my_set.begin(), my_set.end());
   my_multiset.insert(3);
   my_multiset.insert(4);

   print_list(my_set);
   print_list(my_multiset);
   return 0;
}
```


예제

```
template  <typename T>
void print_list(const T& container)
{
   T::const_iterator it;
   for(it=container.begin(); it!=container.end(); ++it)
        cout << *it << " ";
   cout << endl;
}
```

실행 결과

```
1 2 3
1 2 3 3 4
계속하려면 아무 키나 누르십시오 . . .
```

집합에서의 탐색

```
int main()
{
   set<int> my_set;

   my_set.insert(1);
   my_set.insert(2);
   my_set.insert(3);

   set<int>::iterator  pos = my_set.find(2);
   if( pos != my_set.end() )
        cout << "값 " << *pos << "가 발견되었음" << endl;
   else
        cout << "값이 발견되지 않았음" << endl;
   return 0;
}
```

실행 결과

```
값 2가 발견되었음
계속하려면 아무 키나 누르십시오 . . .
```


맵

- Map은 사전과 같은 자료 구조
- 키(key)가 제시되면 Map은 값(value)을 반환한다

Map

그림 18-10 Map의 개념

예제

map1.cpp

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

int main()
{
    map<string, string> dic;
    dic["boy"]="소년";
    dic["school"]="학교";
    dic["office"]="직장";
    dic["house"]="집";
    dic["morning"]="아침";
    dic["evening"]="저녁";

    cout << "house의 의미는 " << dic["house"] << endl;         // 등록이 된 단어
    cout << "morning의 의미는 " << dic["morning"] << endl;    // 등록이 된 단어
    cout << "unknown의 의미는 " << dic["unknown"] << endl; // 등록이 안된 단어
    return 0;
}
```

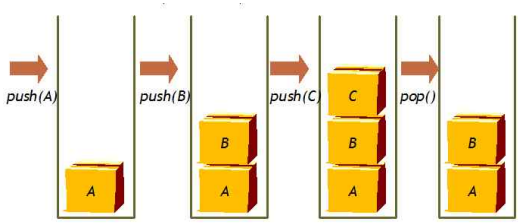

실행 결과

실행 결과

```
house의 의미는 집
morning의 의미는 아침
unknown의 의미는
계속하려면 아무 키나 누르십시오 . . .
```

스택

- 스택은 늦게 들어온 데이터들이 먼저 나가는 자료 구조



예제

stack.cpp

```
// 벡터를 변형하여서 스택을 생성한다.
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main()
{
   stack<string> st;
   string sayings[3] =
   {"The grass is greener on the other side of the fence",
   "Even the greatest make mistakes",
   "To see is to believe" };

   for (int i =0; i < 3; ++i)
        st.push(sayings[i]);
   while (!st.empty()) {
        cout << st.top() << endl;
        st.pop();
   }
   return 0;
}
```


실행 결과

실행 결과

```
To see is to believe
Even the greatest make mistakes
The grass is greener on the other side of the fence
계속하려면 아무 키나 누르십시오 . . .
```


큐

- 큐(queue)는 먼저 들어온 데이터들이 먼저 나가는 자료 구조

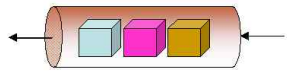


그림 18-11 큐

예제

queue.cpp

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

int main()
{
   queue<int> qu;
   qu.push(100);
   qu.push(200);
   qu.push(300);
   while (!qu.empty()) {
        cout << qu.front() << endl;
        qu.pop();
   }
   return 0;
}
```

실행 결과

```
100
200
300
계속하려면 아무 키나 누르십시오 . . .
```

0

우선 순위 큐

- 우선 순위큐
  - 원소들은 들어온 순서와는 상관없이 우선 순위가 높은 원소가 먼저 나가게 된다.

우선 순위큐

그림 18-12 우선 순위큐

62/40

예제

pqueue.cpp

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

int main()
{
   priority_queue<int> pq;
   pq.push(100);
   pq.push(200);
   pq.push(300);
   while (!pq.empty()) {
        cout << pq.top() << endl;
        pq.pop();
   }
   return 0;
}
```

실행 결과

```
300
200
100
계속하려면 아무 키나 누르십시오 . . .
```

0