객체지향프로그래밍과 자료구조

Ch 7. 함수 템플릿과 클래스 템플릿

정보통신공학과
교수 김 영 탁
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

◆함수 템플릿 (Function Template)

- Syntax, definition
- Compiler complications

◆클래스 템플릿 (Class Template)

- Syntax
- Example: array template class (T_Array)

◆템플릿과 상속

- Example: partially-filled array template class

◆STL (Standard Template Library)

- STL vector
- STL list
- STL deque

Template

템플릿 (Template)

◆ 템플릿이란?

- 다양한 도형을 쉽게 그릴 수 있도록 만들어진 모형 틀
- 옷을 쉽게 만들 수 있도록 미리 준비된 의복 디자인 본
- 문서, 발표자료 등을 주어진 형식에 따라 쉽게 만들 수 있도록 미리 준비된 양식

◆ C++ templates

- Allow very "general" definitions for functions and classes
- Type names (e.g., int, double, char, MyClass) are "parameters" instead of actual types
- Precise definition is determined at run-time

예)
- instance : 붕어빵, 국화빵
- class : 붕어빵 틀, 국화빵 틀,
- class template : 붕어빵 틀과 국화빵 틀을 만드는 기본 구조 틀
  (빵의 모양을 달리하여 다양한 " oo빵틀"을 생성할 수 있음

함수 템플릿 (Function Template)

- **Example**

```
void swapEntry(int& ent1, int& ent2)
{
    int temp;

    temp = ent1;
    ent1 = ent2;
    ent2 = temp;
}
```

- **Applies only to entries of type int**
- **But code would work for any types (e.g., double, char, string)!**

함수 템플릿과 함수 오버로딩

- **Could overload function for char's:**

```
void swapEntry(char& ent1, char& ent2)
{
    char temp;

    temp = ent1;
    ent1 = ent2;
    ent2 = temp;
}
```

- **But notice: code is nearly identical!**
  - Only difference is type used in 3 places (at two arguments passing, temp variable)

함수 템플릿 문법

- **Allow "swap entries" of any type variables:**

```
template<typename T>
void swapEntry(T& ent1, T& ent2)
{
        T temp;

        temp = ent1;
        ent1 = ent2;
        ent2 = temp;
}
```

- **First line "template<typename T>" is called "template prefix"**
  - Tells compiler what's coming is "template"
  - And that T is a type parameter

Template Prefix (1)

- **template<typename T>**
  - In this usage, "**typename**" means "type", or "classification"
  - C++ allows keyword "**class**" in place of keyword "typename" here;
    Can be confused with other "known" use of word "class"!

Template Prefix (2)

- **Again:**
  template<typename T>
- **T can be replaced by any type**
  - Predefined or user-defined (like a C++ class type)
  - int, double, char, float
  - Person, Student, Planet
- **In function definition body:**
  - **T** used like any other type
- **Note: can use other than "T", such as "K", "V", and "E"**

함수 템플릿 호출

- **Consider following call:**
  swapEntry(int1, int2);
  - C++ compiler "generates" function definition for two int parameters using template
- **Likewise for all other types**
- **Need not do anything "special" in call**
  - Required definition automatically generated

여러개의 파라메터 들을 가지는 템플릿

◆ **Can have:**
template<typename T1, typename T2>

◆ 예)

template<typename int, typename Student> // student_id as key
template<typename string, typename Student> // name as key
template<typename double, typename Student> // GPA as key

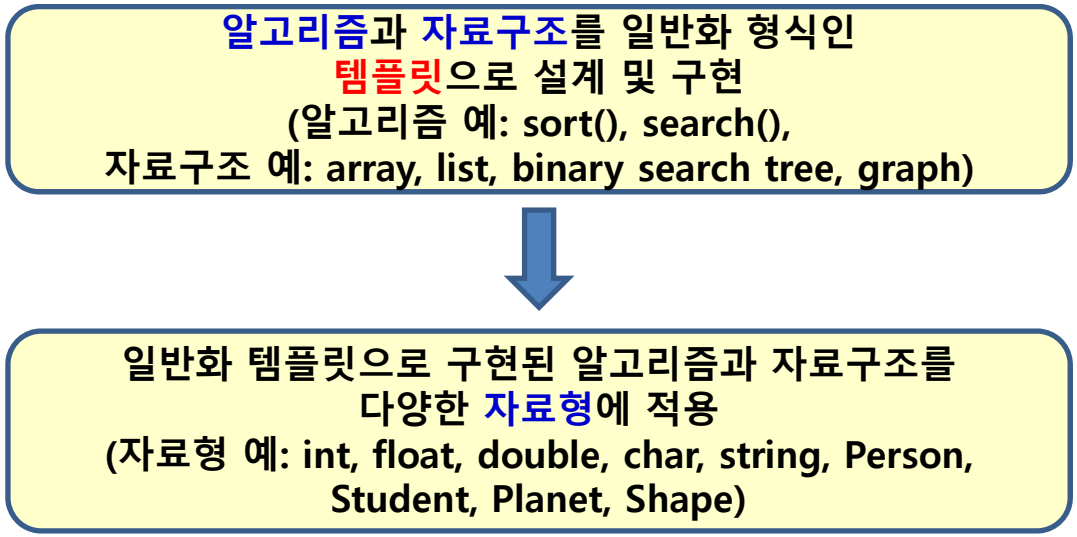
◆ **Not typical**

- Usually only need one "replaceable" type
- Cannot have "unused" template parameters
  - Each typename must be "used" in definition
  - Error otherwise!

알고리즘과 자료구조의 추상화
(Abstraction of Algorithm and Data Structure)

◆ 알고리즘과 자료구조의 추상화를 위한 템플릿

- 알고리즘과 자료구조를 일반화 형식인 템플릿으로 설계 및 구현
- 일반화 템플릿으로 구현된 알고리즘과 자료구조를 기반으로 컴파일러는 응용 프로그램에서 필요에 특정 자료형을 위한 알고리즘과 자료구조의 실행코드를 생성
- 추상화 알고리즘 : generic search and sorting
- 추상화 자료구조 : generic array, list, queue, tree, graph, etc.



함수 템플릿 설계 및 구현 예 (1)

◆ 함수 템플릿

- 함수의 구조와 기능은 동일하나, 사용되는 데이터 유형만 다른 경우, 각 데이터 유형별로 별도의 함수를 구현하지 않고, 하나의 템플릿 함수로 구현하면, 컴파일러가 필요에 따라 자동적으로 해당 함수를 생성하여 사용할 수 있도록 하여 주는 기능
- 하나의 템플릿 함수만 구현하면 되므로, 소스코드 구현 및 관리가 쉽다

◆ 함수 템플릿의 예 : sum_array()

```
template<typename T>
T sum_array(T array[], int size)
{
    T s;

    s = 0;
    for (int i = 0; i < size; i++)
        s += array[i];

    return s;
}
```

함수 템플릿 설계 및 구현 예 (2)

◆ typeid(T).name()을 사용하여 typename 확인

```
/* Template_Print.h  (1) */

#ifndef TEMPLATE_PRINT_H
#define TEMPLATE_PRINT_H
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

template<typename T>
void fprintTypeNameValue(ostream& fout, T value)
{
      string typeName;

      typeName = typeid(T).name();
      fout << "TypeName(" << typeName << "), Value (";
      if ((typeName == "int") || (typeName == "short"))
      {
          fout << setw(5) << value <<")";
      }
```

```
/* Template_Print.h  (2) */

      else if ((typeName == "double") || (typeName == "float"))
      {
          fout.setf(ios::fixed | ios::showpoint);
          fout.precision(2);
          fout << setw(8) << value;
      }
      else // if ((typeName == "char") || (typeName == "class Student") || . . . . )
      {
          fout << value;
      }
      fout << ")";
}
#endif
```

```
/* main */
#include <iostream>
#include <iomanip>
#include <string>
#include "Template_Print.h"
#include "Student.h"

void main()
{
	int i = 10;
	double d = 123.456;
	string str = "Test string";
	Time t(10, 20, 30);
	Date dt(2018, 10, 17);
	Student st(21811000, string("Kim, G-M"),  Date(1990, 10, 5), Time(11, 45, 10), 3.57);

	fprintTypeNameValue(cout, i); cout << endl;
	fprintTypeNameValue(cout, d); cout << endl;
	fprintTypeNameValue(cout, str); cout << endl;
	fprintTypeNameValue(cout, t); cout << endl;
	fprintTypeNameValue(cout, dt); cout << endl;
	fprintTypeNameValue(cout, st); cout << endl;
}
```

```
TypeName(int), Value (   10))
TypeName(double), Value (  123.46)
TypeName(class std::basic_string<char,struct std::char_traits<char>,class std::allocator<char> >), Value (Test string)
TypeName(class Time), Value ((10:20:30))
TypeName(class Date), Value ([2018.10.17] )
TypeName(class Student), Value (Student [ st_id: 21811000, name: Kim, G-M  , arrival: (11:45:10), gpa:  3.57])
계속하려면 아무 키나 누르십시오 . . .
```

템플릿에서 사용할 수 없는 자료형

◆Can use any type in template for which code makes "sense"

- Code must behave in appropriate way
- 단, 대입연산자 ('=')가 정의되어 있지 않은 자료형은 사용할 수 없다

◆e.g., swapEntry() template function

- Example: an array:
  int a[10], b[10];
  swapEntry(a, b);
  - Arrays cannot be "assigned"!

class Template 설계 및 구현 예
Template Class T_Array<T>

클래스 템플릿

◆ Can also "generalize" classes template<typename T>

- Can also apply to class definition
- All instances of "T" in class definition are replaced by type parameter
- Just like for function templates!

◆ Example of class template

```
template<typename T>
class DLLNode // generalized doubly linked list node
{}

template<typename T>
class BinTreeNode // generalized binary tree node
{}
```

Template class T_Array

```
/* Template class T_Array.h (1) */
#ifndef T_Array_H
#define T_Array_H
#include <iostream>
#include <iomanip>
using namespace std;
enum SortingOrder { INCREASING, DECREASING };

template<typename T>
class T_Array
{
public:
    T_Array(int n, string nm); // constructor
    ~T_Array(); // destructor
    int size() { return num_elements; }
    bool empty() { return num_elements == 0; }
    string getName() { return name; }
    void reserve(int new_capacity);
    void insert(int i, T element);
    void insertBack(T element);
    void remove(int i);
```

```
/* Template class T_Array.h (2) */

    T& at(int i);
    void set(int i, T& element);
    T& getMin(int begin, int end);
    T& getMax(int begin, int end);
    void shuffle();
    int sequential_search(T search_key); // search and return the index; -1 if not found
    int binary_search(T search_key); // search and return the index; -1 if not found
    void selection_sort(SortingOrder sortOrder=INCREASING);
    void quick_sort(SortingOrder sortOrder=INCREASING);
    void fprint(ofstream &fout, int elements_per_line);
    void fprintSample(ofstream &fout, int elements_per_line, int num_sample_lines);
    bool isValidIndex(int i);
    T& operator[](int index) { return t_array[index]; }
private:
    T *t_array;
    int num_elements;
    int capacity;
    string name;
};
```

```
/* Template class T_Array.h (3) */

template<typename T>
T_Array<T>::T_Array(int new_capacity,  string nm)  // constructor
{
    capacity = new_capacity;
    t_array = (T *) new T[capacity];
    if (t_array == NULL)
    {
        cout << "Error in creation of dynamic array of size (" << new_capacity << ") !!" << endl;
        exit;
    }
    num_elements = 0;
    name = nm;
}

template<typename T>
T_Array<T>::~T_Array() // destructor
{
    if (t_array != NULL)
       delete[] t_array;
}
```

```
/* Template class T_Array.h (4) */

template<typename T>
void T_Array<T>::reserve(int new_capacity) // reserve at least new_capacity
{
    if (capacity >= new_capacity)
        return;  // already big enough
    T *t_newGA = (T *) new T[new_capacity];
    if (t_newGA == NULL)
    {
       cout << "Error in creation of dynamic array of size (" << new_capacity << ") !!" << endl;
       exit;
    }
    cout << this->getName() << " expands capacity to " << setw(3)
          << new_capacity << endl;
    for (int i = 0; i < num_elements; i++)
        t_newGA[i] = t_array[i];

    delete[] t_array;
    t_array = t_newGA;
    capacity = new_capacity;
}
```

```
/* Template class T_Array.h (5) */

template<typename T>
bool T_Array<T>::isValidIndex(int index)
{
    if ((index < 0) || (index > num_elements))
       return false;
    else
       return true;
}

template<typename T>
void T_Array<T>::insert(int i, T new_element)
{
    if (num_elements >= capacity) // full ?
    {
        int new_capa;
        new_capa =  ((2 * capacity) > 1) ? 2 * capacity : 1;
        reserve(new_capa);
    }
    if (isValidIndex(i))
    {
        for (int j = num_elements - 1; j >= i; j--)
           t_array[j + 1] = t_array[j];  //shift up elements in one position
        t_array[i] = new_element;
        num_elements++;
    }
}
```

```
/* Template class T_Array.h (6) */

template<typename T>
void T_Array<T>::insertBack(T new_element)
{
    if (num_elements >= capacity) // full ?
    {
        int new_capa;
        new_capa =  ((2 * capacity) > 1) ? 2 * capacity : 1;
        reserve(new_capa);
    }
    t_array[num_elements] = new_element;
    num_elements++;
}
```

```
/* Template class T_Array.h (7) */

template<typename T>
void T_Array<T>::remove(int i)
{
   if (isValidIndex(i))
   {
      for (int j = i + 1; j < num_elements; j++)
         t_array[j - 1] = t_array[j];  //shift down elements in one position
      num_elements--;
   }
   if (num_elements < (capacity / 2))
   {
      int new_capacity = capacity / 2;
      T *t_newGA = (T *) new T[new_capacity];
      if (t_newGA == NULL)
      {
          return; // new memory allocation failed.
                  // Just return without modification.
      }
      cout << this->getName()
           << " reduces capacity to " << setw(3)
           << new_capacity << endl;
      for (int i = 0; i < num_elements; i++)
          t_newGA[i] = t_array[i];

      delete[] t_array;
      t_array = t_newGA;
      capacity = new_capacity;
   }
}
```

```
/* Template class T_Array.h (8) */

template<typename T>
T& T_Array<T>::at(int i)
{
    if (isValidIndex(i))
    {
        return t_array[i];
    }
}

template<typename T>
void T_Array<T>::set(int i, T& element)
{
    if (isValidIndex(i))
    {
       t_array[i] = element;
    }
}
```

```
/* Template class T_Array.h (9) */
template<typename T>
void T_Array<T>::shuffle()
{
    srand(time(0));
    int index1, index2;
    int rand_1, rand_2;
    T temp;

    for (int i = 0; i < num_elements; i++)
    {
        rand_1 = rand();
        rand_2 = rand();
        index1 = ((rand_1 << 15) | rand_2) % num_elements;
        rand_1 = rand();
        rand_2 = rand();
        index2 = ((rand_1 << 15) | rand_2) % num_elements;

        temp = t_array[index1];
        t_array[index1] = t_array[index2];
        t_array[index2] = temp;
    }
}
```

```
/* Template class T_Array.h (10) */
template<typename T>
T& T_Array<T>::getMin(int begin, int end)
{
    T minValue;
    int index_min;

    minValue = t_array[begin];
    index_min = begin;
    for (int i = begin + 1; i <= end; i++)
    {
        if (t_array[i] < minValue) // T must provide operator<() overloading !!
        {
            minValue = t_array[i];
            index_min = i;
        }
    }
    return t_array[index_min];
}

template<typename T>
T& T_Array<T>::getMax(int begin, int end)
{
    T maxValue;
    int index_max;

    maxValue = t_array[begin];
    index_max = begin;
    for (int i = begin + 1; i <= end; i++)
    {
        if (t_array[i] > maxValue) // T must provide operator>() overloading !!
        {
            maxValue = t_array[i];
            index_max = i;
        }
    }
    return t_array[index_max];
}
```

```
/* Template class T_Array.h (11) */

template<typename T>
int T_Array<T>::sequential_search(T search_key)
{
    int index;
    for (int index = 0; index < num_elements; index++)
    {
        if (t_array[index] == search_key)  // T must provide operator==() overloading !!
            return index;
    }
    return -1;
}
```

```
/* Template class T_Array.h (12) */

template<typename T>
int T_Array<T>::binary_search(T search_key)
{
    int low, mid, high;
    int loop = 1;

    low = 0;
    high = num_elements - 1;

    while (low <= high)
    {
        cout << setw(2) << loop << "-th loop: current search range [" << setw(3)
             << low << ", " << setw(3) << high << "]" << endl;
        mid = (low + high) / 2;
        if (t_array[mid] == search_key) // T must provide operator==() overloading !!
            return mid;
        else if (t_array[mid] > search_key) // T must provide operator>() overloading !!
            high = mid - 1;
        else
            low = mid + 1;
        loop++;
    }
    return -1;
}
```

```
/* Template class T_Array.h (13) */

template<typename T>
void T_Array<T>::selection_sort(SortingOrder sortOrder=INCREASING)
{
    int index_min, index_max;  // index of the element with minimum value
    T minValue; // minimum value
    T maxValue;

    for (int i = 0; i < num_elements - 1; i++)
    {
        if (sortOrder == INCREASING) { // sorting in increasing (non_decreasing) order
            index_min = i;
            minValue = t_array[i];
            for (int j = i + 1; j < num_elements; j++)
            {
               if (t_array[j] < minValue) // T must provide operator<() overloading !!
               {
                   index_min = j;
                   minValue = t_array[j];
               }
            }
            if (index_min != i) // if a smaller element is found, then swap
            {
               /* minValue is t_array[min] */
               t_array[index_min] = t_array[i];
               t_array[i] = minValue;
            }
        }
```

```
/* Template class T_Array.h (14) */

        else { // sorting in decreasing (non_increasing) order
            index_max = i;
            maxValue = t_array[i];
            for (int j = i + 1; j < num_elements; j++)
            {
               if (t_array[j] > maxValue) // T must provide operator>() overloading !!
               {
                   index_max = j;
                   maxValue = t_array[j];
               }
            }
            if (index_max != i) // if a smaller element is found, then swap
            {
               /* maxValue is t_array[max] */
               t_array[index_max] = t_array[i];
               t_array[i] = maxValue;
            }
        }
    }  // end for
}
```

```
/* Template class T_Array.h (15) */

template<typename T>
int _partition(T *array, int size, int left, int right, int pivotIndex,
     SortingOrder sortOrder=INCREASING)
{
    T pivotValue, temp; // pivot value
    int newPI; // new pivot index

    /* place the pivot element at right-position */
    pivotValue = array[pivotIndex];
    array[pivotIndex] = array[right];
    array[right] = pivotValue; // Move pivot to array[right]

    newPI = left; // newPI is the index that points the position
                  //   where pivot element will be finally re-located
    for (int i = left; i <= (right - 1); i++) {
     if (sortOrder == INCREASING) // sorting in increasing order
     {
        if (array[i] <= pivotValue) // T must provide operator<=() overloading !!
        {
           temp = array[i];
           array[i] = array[newPI];
           array[newPI] = temp;

           newPI = newPI + 1;
              // note: all elements in left of index newPI are equal or smaller than pivot_value
        }
     }
```

```
/* Template class T_Array.h (16) */

        else // sorting in decreasing (non_increasing) order
        {
             if (array[i] > pivotValue) // T must provide operator>() overloading !!
            {
                temp = array[i];
                array[i] = array[newPI];
                array[newPI] = temp;

                newPI = newPI + 1;
                // note: all elements in left of index newPI are greater than pivot_value
             }
        }
    } // end for

    // swap array[newPI] and array[right]; Move pivot element to its final place
    temp = array[newPI];
    array[newPI] = array[right];
    array[right] = temp;

    return newPI;
}
```

```
/* Template class T_Array.h (17) */

template<typename T>
void _quickSort(T *array, int size, int left, int right,
    SortingOrder sortOrder=INCREASING)
{
    int pI, newPI; // pivot index

    if (left >= right)
    {
        return;
    }
    else
    {//select a pI (pivotIndex) in the range   left ≤ pI ≤ right
      pI = (left + right) / 2;
    }
    newPI = _partition(array, size, left, right, pI, sortOrder);

    if (left < (newPI - 1)) {
        _quickSort(array, size, left, newPI - 1, sortOrder);
        // recursively sort elements on the left of  pivotNewIndex
    }
    if ((newPI + 1) < right) {
        _quickSort(array, size, newPI + 1, right, sortOrder);
        // recursively sort elements on the right of  pivotNewIndex
    }
}
```

```
/* Template class T_Array.h (18) */

template<typename T>
void T_Array<T>::quick_sort(SortingOrder sortOrder=INCREASING)
{
    int pI, newPI; // pivot index

    _quickSort(this->t_array, num_elements, 0, num_elements - 1, sortOrder);

}

template<typename T>
void T_Array<T>::fprint(ofstream &fout, int elements_per_line)
{
    int count = 0;
    while (count < num_elements)
    {
        for (int i = 0; i < elements_per_line; i++)
        {
            fout << t_array[count] << "   ";
            count++;
            if (count % elements_per_line == 0)
               fout << endl;
        }
    }
    fout << endl;
}
```

```
/* Template class T_Array.h (19) */

template<typename T>
void T_Array<T>::fprintSample(ofstream &fout, int elements_per_line,
     int num_sample_lines)
{
    string T_type;
    int last_block_start;
    int index = 0;

    T_type = typeid(T).name();
    for (int i = 0; i<num_sample_lines; i++)
    {
        for (int j = 0; j < elements_per_line; j++)
        {
            if (index >= num_elements)
            {
                fout << endl;
                return;
            }
            if ((T_type == string("int")) || (T_type == string("double")) ||
               (T_type == string("class std::basic_string<char,struct std::char_traits<char>,class
               std::allocator<char> >")))
                fout << setw(10) << t_array[index];
            else
                fout << t_array[index] << "  ";
            index ++;
        }
        fout << endl;
    }
```

```
/* Template class T_Array.h (20) */

    if (count < (num_elements - elements_per_line * num_sample_lines))
       count = num_elements - elements_per_line * num_sample_lines;
    fout << "      . . . . .  " << endl;

    for (int i = 0; i < num_sample_lines; i++)
    {
        for (int j = 0; j< elements_per_line; j++)
        {
            if (index >= num_elements)
            {
                fout << endl;
                return;
            }
            if ((T_type == string("int")) || (T_type == string("double")) ||
               (T_type == string("class  std::basic_string<char, struct  std::char_traits<char>, class
                  std::allocator<char> >")))
               fout << setw(10) << t_array[index];
            else
               fout << t_array[index] << "  ";
            index ++;
        }
        fout << endl;
    }
    fout << endl;
}
#endif
```

Template Class의 응용
템플릿 클래스와 상속

T_Array을 class Planet에 적용

```
/* Planet.h (1) */
#ifndef C_PLANET_H
#define C_PLANET_H
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Planet
{
    friend ostream& operator<<(ostream& fout, const Planet& p)
    {
       fout.precision(2);
       fout.setf(ios::showpoint);
       fout.setf(ios::fixed);
       fout << "[" << setw(8) << p.getName() << ", relMass(" << setw(8)
            << p.getRelMass() << "), dist(" << setw(8) << p.getDist() << ")]";
       return fout;
    }
public:
    Planet(string nm, double rM, double dist)
    : name(nm), relativeMass(rM), distance(dist)
    { }
    Planet() {}
    ~Planet() {}
```

```
/* Planet.h (2) */

    string getName() const { return name; }
    void setName(string nm) { name = nm; }
    double getRelMass() const { return relativeMass; }
    void setRelMass(double rM) { relativeMass = rM; }
    double getDist() const { return distance; }
    void setDist(double dist) { distance = dist; }
    bool operator<(const Planet& p) { return (distance < p.distance); }
    bool operator<=(const Planet& p) { return (distance <= p.distance); }
    bool operator>(const Planet& p) { return (distance > p.distance); }
    bool operator>=(const Planet& p) { return (distance >= p.distance); }
    void fprint(ostream& fout)
    {
        fout.precision(2);
        fout.setf(ios::showpoint);
        fout.setf(ios::fixed);
        fout << "[" << setw(8) << name << ", relMass(" << setw(8) << relativeMass
            << "), dist(" << setw(8) << distance << ")]";
    }
private:
    string name;
    double relativeMass;
    double distance;
};
#endif
```

main()

```
/* main_T_Array.cpp (1) */

#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include "T_Array.h"
#include "Planet.h"

using namespace std;
#define ELEMENTS_PER_LINE 10
#define SAMPLE_LINES 5
#define NUM_ELEMENTS 500
#define MIN_NUM_ELEMENTS 10

#define NUM_PLANETS 9
Planet planets[NUM_PLANETS]
{
    Planet(string("Mercury"), 0.0558, 57.9),
    Planet(string("Venus"), 0.815, 108),
    Planet(string("Earth"), 1.0, 150),
    Planet(string("Mars"), 0.107, 228),
    Planet(string("Jupitor"), 318, 778),
    Planet(string("Saturn"), 95.1, 1430),
    Planet(string("Uranus"), 14.5, 2870),
    Planet(string("Neptune"), 17.2, 4500),
    Planet(string("Pluto"), 0.11, 5900)
};
```

```
/* main_T_Array.cpp (2) */

void main()
{
    ofstream fout;
    T_Array<Planet> planetArray(NUM_PLANETS, "Array of Planets");
    Planet *pPlanet;
    string str, key_str;
    int key_index;

    fout.open("output.txt");
    if (fout.fail())
    {
        cout << "Fail to open output.txt file for results !!" << endl;
        exit;
    }

    srand(time(0));
    for (int i = 0; i < NUM_PLANETS; i++)
    {
        pPlanet = &planets[i];
        planetArray.insert(i, *pPlanet);
    }
```

```
/* main_T_Array.cpp (3) */


    fout << "Elements in planetArray after
       initialization :" << endl;
    planetArray.fprint(fout, 1);

    fout << "Elements in planetArray after
       shuffling :" << endl;
    planetArray.shuffle();
    planetArray.fprint(fout, 1);

    fout << "Elements in planetArray after sorting :"
       << endl;
    planetArray.selection_sort(INCREASING);
    planetArray.fprint(fout, 1);

    fout << "Elements in planetArray after sorting :"
        << endl;
    planetArray.quick_sort(DECREASING);
    planetArray.fprint(fout, 1);

    fout.close();
}
```

```
Elements in planetArray after initialization :
[ Mercury, relMass(    0.06), dist(   57.90)]
[   Venus, relMass(    0.81), dist(  108.00)]
[   Earth, relMass(    1.00), dist(  150.00)]
[    Mars, relMass(    0.11), dist(  228.00)]
[ Jupitor, relMass(  318.00), dist(  778.00)]
[  Saturn, relMass(   95.10), dist( 1430.00)]
[  Uranus, relMass(   14.50), dist( 2870.00)]
[ Neptune, relMass(   17.20), dist( 4500.00)]
[   Pluto, relMass(    0.11), dist( 5900.00)]

Elements in planetArray after shuffling :
[   Venus, relMass(    0.81), dist(  108.00)]
[   Earth, relMass(    1.00), dist(  150.00)]
[ Mercury, relMass(    0.06), dist(   57.90)]
[  Saturn, relMass(   95.10), dist( 1430.00)]
[ Neptune, relMass(   17.20), dist( 4500.00)]
[   Pluto, relMass(    0.11), dist( 5900.00)]
[  Uranus, relMass(   14.50), dist( 2870.00)]
[ Jupitor, relMass(  318.00), dist(  778.00)]
[    Mars, relMass(    0.11), dist(  228.00)]

Elements in planetArray after sorting :
[ Mercury, relMass(    0.06), dist(   57.90)]
[   Venus, relMass(    0.81), dist(  108.00)]
[   Earth, relMass(    1.00), dist(  150.00)]
[    Mars, relMass(    0.11), dist(  228.00)]
[ Jupitor, relMass(  318.00), dist(  778.00)]
[  Saturn, relMass(   95.10), dist( 1430.00)]
[  Uranus, relMass(   14.50), dist( 2870.00)]
[ Neptune, relMass(   17.20), dist( 4500.00)]
[   Pluto, relMass(    0.11), dist( 5900.00)]

Elements in planetArray after sorting :
[   Pluto, relMass(    0.11), dist( 5900.00)]
[ Neptune, relMass(   17.20), dist( 4500.00)]
[  Uranus, relMass(   14.50), dist( 2870.00)]
[  Saturn, relMass(   95.10), dist( 1430.00)]
[ Jupitor, relMass(  318.00), dist(  778.00)]
[    Mars, relMass(    0.11), dist(  228.00)]
[   Earth, relMass(    1.00), dist(  150.00)]
[   Venus, relMass(    0.81), dist(  108.00)]
[ Mercury, relMass(    0.06), dist(   57.90)]
```

Friends and Templates

◆Friend functions can be used in template classes

- Same as with ordinary classes
- Simply requires type parameter where appropriate

◆Very common to have friends of template classes

- Especially for operator overloadings (as we've seen) of insertion ('<<') and extraction('>>')

템플릿과 상속

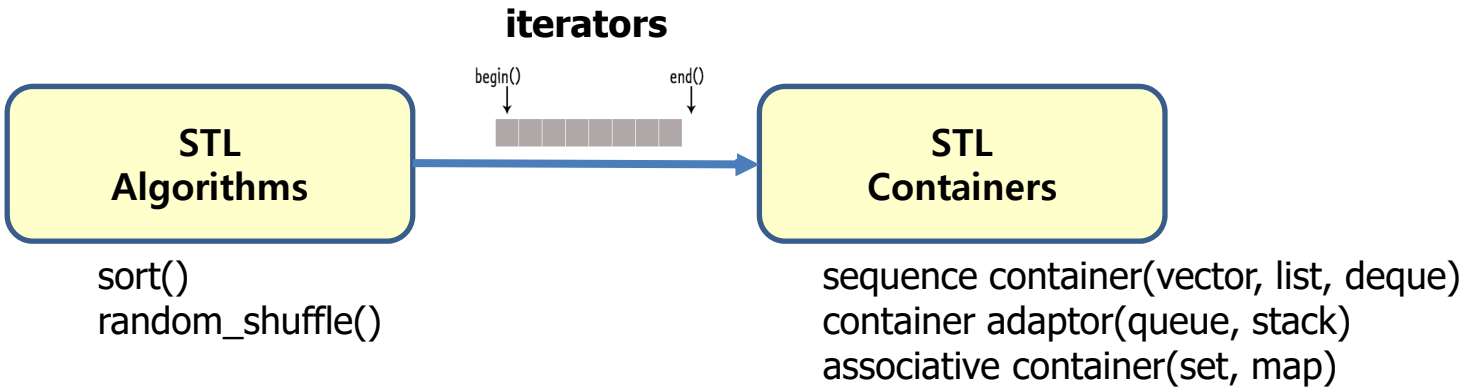
- **Nothing new here**
- **Derived template classes**
  - Can derive from template or non-template class
  - Derived class is then naturally a template class
- **Syntax is same as ordinary class derived from ordinary class**

Standard Template Library (STL)

Standard Template Library (STL)

◆ Standard Template Library (STL)

- Algorithm: sort(), binary_search(), random_shuffle()
- Container
  - Sequence container: vector, list, slist, deque
  - Container Adaptor: queue, stack
  - Associative container: set, map
- Iterator



STL Algorithm

◆ **<algorithm>** (참조 http://www.cplusplus.com/reference/algorithm/)

Category	Library Operations (example)
Non-modifying sequence operations	find(), count(), search(), equal()
Modifying sequence operations	copy(), swap(), fill(), generate(), reverse(), shuffle()
Partitions	partition()
Sorting	sort(), partial_sort(), is_sorted()
Binary search	binary_search()
Merge	merge(), set_union()
Heap	make_heap(), sort_heap()
Min / Max	min(), max()
Other	lexicographical_compare(), next_permutation(), prev_permutation()

STL Containers

◆ STL Sequence, Ordered Collections

- implement data structures which can be accessed in sequence
- vector
- list: doubly linked list
- slist: singly linked list
- deque (double-ended queue)

◆ STL Container Adaptor

- container with specific interface
- queue
- priority queue
- stack

◆ STL Associative Container

- implements sorted data structure that can be quickly searched
- set
- map

STL Iterator (1)

◆ STL Iterator (반복자)

- Iterators are used to point at the memory addresses of STL containers. They are primarily used in sequence of numbers, characters etc. They reduce the complexity and execution time of program.
- An *iterator* is any object that, pointing to some element in a range of elements (such as an array or a container), has the ability to iterate through the elements of that range using a set of operators (with at least the increment (++) and dereference (*) operators).
- The most obvious form of iterator is a *pointer*: A pointer can point to elements in an array, and can iterate through them using the increment operator (++). But other kinds of iterators are possible. For example, each container type (such as a vector or list) has a specific *iterator* type designed to iterate through its elements.
- Notice that while a pointer is a form of iterator, not all iterators have the same functionality of pointers;

STL Iterator (2)

◆ Operations for Iterator

Operation of STL Containers	Remark
begin()	used to return the beginning position of the container
end()	used to return the “after end” position of the container
prev()	move to the position of previous element
next()	move to the position of next element
advance()	advance iterator
distance()	distance between iterators

STL Sequences (ordered collections)

Sequences (arrays/linked lists): ordered collections	
vector	배열과 같은 동적 배열로서 객체를 삽입하거나 제거할 때 자동으로 자신의 크기를 조정하는 능력을 갖는다. vector의 end에 요소를 삽입하거나, 마지막 요소를 제거하는 동작은 단지 일정한 시간을 필요로 하는데, 크기 조정이 일어나지 않기 때문이다. 그러나 처음 또는 중간에 새로운 요소를 삽입하거나 기존 요소를 삭제하는 것은 그 삽입/삭제 대상 요소 이후의 모든 요소들에 대한 재배치 동작을 실행해야하기 때문에 전체 요소 개수에 비례하는 시간이 걸린다.
list	이중 연결 리스트 (doubly linked list)이며, 각 요소들은 근접하지 않는 메모리에 저장될 수 있다. 전체 원소 개수에 선형으로 비례하는 접근 시간과 느린 검색 성능을 갖지만, 한 번 위치가 찾아지면 짧은 시간 내에 삽입과 제거가 가능하다.
slist	단일 연결 리스트 (singly linked list) 이며, 각 요소들이 근접하지 않는 메모리에 저장될 수 있다. 전체 원소 개수에 선형으로 비례하는 접근 시간과 느린 검색 성능을 갖지만, 한 번 위치가 찾아지면 짧은 시간 내에 삽입과 제거가 가능하다. slist는 이중 연결 리스트보다 삽입과 제거에서 약간 효율적이며 적은 메모리를 사용한다. 하지만 단지 전방향 (forward)으로만 진행할 수 있다.
deque	double-ended queue 시작과 끝에서 삽입과 삭제 기능을 상수 시간에 수행

STL Container Adaptors

Container Adaptor	
queue	FIFO (first in first out) Queue 인터페이스를 push/pop/front/back 연산들로 제공한다. front(), back(), push_back(), 그리고 pop_front() 연산을 지원하는 어느 시퀀스 (예를 들면 list 그리고 deque) 들도 큐의 구현에 사용될 수 있다
priority queue	우선순위 큐 인터페이스를 push/pop/top 연산들로 제공한다 (높은 우선순위 요소가 위에 존재한다). 연산 front(), push_back(), 그리고 pop_back() 를 지원하는 어느 임의 접근 시퀀스도 priority_queue를 인스턴스화하는데 사용될 수 있다(예를 들면 vector , deque). 이것은 힙을 사용하여 구현된다. 어느 요소가 높은 우선순위를 갖는 지와 먼저 pop되어야 하는지를 결정하기 위한 비교 (comparison) 기능을 지원해야 한다
stack	LIFO (last in first out) stack 인터페이스를 push/pop/top 연산들을 통해 지원한다. 마지막에 삽입된 요소가 top에 존재한다. back(), push_back(), 그리고 pop_back()을 지원하는 어떤 시퀀스 자료구조 (예를 들면 vector, list, 그리고 deque)도 stack으로 사용할 수 있다.

STL Associative Containers (unordered collections)

Associative containers: unordered collections	
Set (집합)	집합 연산 union, intersection, difference, symmetric difference 그리고 inclusion 테스트를 제공한다. 데이터의 형은 반드시 비교 연산자 < 또는 명시된 비교연산자 함수를 통해서 구현되어야 한다; 이러한 비교 연산 또는 비교연산자 함수는 반드시 strict weak ordering을 보장해야 하며, 그렇지 않으면 행위가 정의되지 않는다. 일반적으로 자가 균형 (self-balancing) 이진 탐색 트리를 사용해서 구현된다. set에서 요소를 삽입하고 제거하는 것은 set에서 가리키는 iterator를 무효화하지 않는다.
multiset	set과 같지만 중복 요소들을 허용하며, 수학적으로 중복집합(Multiset)에 해당한다.
Map (맵)	연관 배열이며, 한 데이터 아이템(키)에서 다른 것(값)으로 매핑 (mapping)을 허용한다. 키의 타입은 반드시 비교 연산 < 또는 사용자가 정의한 비교연산자를 통해 구현되어야 한다; 이러한 비교 연산 또는 비교연산자 함수는 반드시 strict weak ordering을 보장해야 하며, 아니면 행위가 정의되지 않는다. 일반적으로 자가 균형 (self-balancing) 이진 탐색 트리를 사용해서 구현된다.
multimap	map과 같지만 중복 키들을 허용한다.

STL Vector

- **Similar to array:**
  - Has base type
  - Stores collection of base type values
  - Recall: arrays are fixed size
  - Vectors: "arrays that can grow and shrink" during program execution
  - vectors are formed from a template class in the Standard Template Library (STL)
- **Collections and Vectors**
  - a collection S of n elements stored in a certain linear order
    - list, sequence
    - index (or rank) of range [0, n-1]
  - vector: a sequence that supports access to its elements by their indices
- **Declared differently:**
  - Syntax: vector<Base_Type>
    - Indicates template class
    - Any type can be "plugged in" to Base_Type
    - Produces "new" class for vectors with that type
  - Example declaration:

```
vector<int> v;
typename vector<int>::iterator p;   p = v.begin();
```

STL Vector 주요 멤버 함수

STL vector 주요 멤버함수	설명
begin() end()	첫 번째 원소를 가리키는 반복자 (iterator)를 반환 마지막 원소 다음 (after end) 위치 를 가리키는 반복자(iterator)를 반환
at(n) operator[n] front() back()	n 번째 원소를 참조할 때 사용, 범위 점검을 함 n 번째 원소를 참조할 때 사용, 범위 점검을 하지 않음 첫 번째 원소의 참조를 반환 마지막 원소의 참조를 반환
empty() size() resize() capacity() reserve()	원소가 아무것도 없으면 true, 하나라도 존재하면 false 원소의 개수를 반환 vector의 크기를 변경하고, default 값이나 임의의 값으로 초기화 vector에 할당된 메모리의 크기를 반환 지정한 크기 만큼의 메모리를 미리 할당
clear() erase() insert() push_front() pop_front() push_back() pop_back() swap() sort() random_shuffle()	vector의 모든 원소를 제거 지정된 위치의 원소나 지정된 범위의 원소를 삭제 지정된 위치에 지정된 값을 삽입 제공되지 않음 제공되지 않음 vector의 끝에 원소를 추가 vector의 마지막 원소를 제거 v1.swap(v2)일 때 v1과 v2를 swap시킴 제공되지 않음, <algorithm> 라이브러리에서 제공 제공되지 않음, <algorithm> 라이브러리에서 제공

Applications of Vectors

- **Direct applications**
  - Sorted collection of objects (elementary database)
- **Indirect applications**
  - Auxiliary data structure for algorithms
  - Component of other data structures

Application of STL Vector

```
/* Application of STL Vector (1) */
#include <cstdlib> // provides EXIT_SUCCESS
#include <iostream> // I/O definitions
#include <vector> // provides vector
#include <algorithm> // for sort, random_shuffle
using namespace std; // make std:: accessible

template <typename T>
void printVector(vector<T> &v)
{
    cout << "Vector size(" << v.size() << "), elements : ";
    typename vector<T>::iterator p;
    for (p = v.begin(); p != v.end(); ++p)
        cout << *p << " ";
    cout << endl;
}

int main()
{
    /* Testing vector<int> */
    int a[] = { 17, 12, 33, 15, 62, 45 };
    vector<int> v(a, a + 6);
    printVector(v);  // v: 17 12 33 15 62 45
```

```
/* Application of STL Vector (2) */

    v.pop_back();    printVector(v);  // v: 17 12 33 15 62
    v.push_back(19);
    printVector(v);  // v: 17 12 33 15 62 19
    cout << endl;
    cout << "v.front: " << v.front() << ", v.back: " << v.back() << endl; // outputs: 17 19
    sort(v.begin(), v.begin() + 4);   printVector(v); // v: 12 15 17 33 62 19
    cout << endl;
    v.erase(v.end() - 4, v.end() - 2); // v: 12 15 62 19
    cout << "current v.size() = " << v.size() << endl;      // outputs: 4
    printVector(v);  cout << endl;

    /* Testing vector<char> */
    char b[] = { 'b', 'r', 'a', 'v', 'o' };
    vector<char> w(b, b + 5);      // w: b r a v o
    cout << "at initialization, vector<char> w : " << endl;
    printVector(w); cout << endl;
    random_shuffle(w.begin(), w.end()); // w: o v r a b
    w.insert(w.begin(), 's');      // w: s o v r a b

    cout << "after shuffling and inserting s at the begin: "
      << endl;
    printVector(w); cout << endl;  // outputs: s o v r a b
    return 0;
}
```

```
Vector size(6), elements : 17 12 33 15 62 45
Vector size(5), elements : 17 12 33 15 62
Vector size(6), elements : 17 12 33 15 62 19

v.front: 17, v.back: 19
Vector size(6), elements : 12 15 17 33 62 19

current v.size() = 4
Vector size(4), elements : 12 15 62 19

at initialization, vector<char> w :
Vector size(5), elements : b r a v o

after shuffling and inserting s at the begin:
Vector size(6), elements : s o r v a b

계속하려면 아무 키나 누르십시오 . . .
```

STL List

◆ STL List

```
#include <list>
using std::list;

list<float> myFloatList;
list<float>::iterator p;

p = myFloatList.begin();
```

STL List 주요 멤버 함수

STL list 주요 멤버함수	설명
begin() end()	첫 번째 원소를 가리키는 반복자 (iterator)를 반환 마지막 원소 다음 (after end) 위치를 가리키는 반복자(iterator)를 반환
at(n) operator[n] front() back()	n 번째 원소를 참조할 때 사용, 범위 점검을 함 제공하지 않음 첫 번째 원소의 참조를 반환 마지막 원소의 참조를 반환
empty() size() resize() capacity() reserve()	원소가 아무것도 없으면 true, 하나라도 존재하면 false 원소의 개수를 반환 리스트의 크기를 변경하고, default 값이나 임의의 값으로 초기화 제공하지 않음 제공하지 않음
clear() erase() insert() push_front() pop_front() push_back() pop_back() swap() sort() merge()	list의 모든 원소를 제거 지정된 위치의 원소나 지정된 범위의 원소를 삭제 지정된 위치에 지정된 값을 삽입 list의 처음에 원소를 추가 list의 첫 번째 원소를 제거 list의 끝에 원소를 추가 list의 마지막 원소를 제거 lst1.swap(lst2)일 때 lst1과 lst2를 swap시킴 list에 포함된 원소를 정렬시킴 L1.merge(L2) : list L2의 원소들을 list L1으로 병합시킴

STL List 응용예제

```
/** List_Iterator.cpp (1) */
#include <cstdlib> // provides EXIT_SUCCESS
#include <iostream> // I/O definitions
#include <list> // provides STL list

using namespace std; // make std:: accessible

template <typename E>
void printList(list<E> &lst)
{
    cout << "List size (" << lst.size() << "), elements : ";
    typename list<E>::iterator p;
    for (p = lst.begin(); p != lst.end(); ++p)
        cout << *p << " ";
    cout << endl;
}
int main()
{
     int num_data;
     int a[] = { 17, 12, 37, 15, 62, 45, 3, 1, 7, 9, 10, 25, 33 };
     int b[] = { 11, 17, 5, 4, 8 };
     list<int> L1, L2;          //
     num_data = sizeof(a) / sizeof(int);
     for (int i = 0; i < num_data; i++)
     {
         L1.push_back(a[i]);
     }
```

```
After data insertions,
 L1: List size (13), elements : 17 12 37 15 62 45 3 1 7 9 10 25 33
Pop_back()
Current size of L1 = 12
Push_back(19)
Front of L1 = 17, back of L1 = 19
Current size of L1 = 13
After L1 is sorted,
 L1: List size (13), elements : 1 3 7 9 10 12 15 17 19 25 37 45 62
List size (5), elements : 11 17 5 4 8
L2 is sorted
List size (5), elements : 4 5 8 11 17
After merging L2 into L1
 L1: List size (18), elements : 1 3 4 5 7 8 9 10 11 12 15 17 17 19 25 37 45 62
 L2: List size (0), elements :
```

```
/** List_Iterator.cpp (2) */

    cout << "After data insertions,\n L1: "; printList(L1); // L1: 17 12 37 15 62 45 3 1 7 9 10 25 33

    cout << "Pop_back()" << endl;
    L1.pop_back();              // L1 : 17 12 37 15 62 45 3 1 7 9 10 25
    cout << "Current size of L1 : " << L1.size() << endl;    // outputs: 12

    L1.push_back(19);      // v: 17 12 37 15 62 45 3 1 7 9 10 25 19
    cout << "Push_back(19)" << endl;
    cout << "Front of L1 : " << L1.front() << ", back of L1 : " << L1.back() << endl; // outputs: 17 19
    cout << "Current size of L1 : " << L1.size() << endl;      // outputs: 13

    L1.sort(); // L1: 1 3 7 9 10 12 15 17 19 25 37 45 62
    cout << "After L1 is sorted, \n L1 : " ; printList(L1);

    num_data = sizeof(b) / sizeof(int);
    for (int i = 0; i < num_data; i++)
    {
        L2.push_back(b[i]);
    }
    printList(L2); // L2: 11 17 5 4 8
    L2.sort(); // 4 5 8 11 17
    cout << "L2 is sorted, \n L2: "; printList(L2);

    L1.merge(L2);
    cout << "After merging L2 into L1; \n L1:";
    printList(L1);
    cout << " L2: "; printList(L2);

    return EXIT_SUCCESS;
}
```

```
After data insertions,
 L1: List size (13), elements : 17 12 37 15 62 45 3 1 7 9 10 25 33
Pop_back()
Current size of L1 = 12
Push_back(19)
Front of L1 = 17, back of L1 = 19
Current size of L1 = 13
After L1 is sorted,
 L1: List size (13), elements : 1 3 7 9 10 12 15 17 19 25 37 45 62
List size (5), elements : 11 17 5 4 8
L2 is sorted
List size (5), elements : 4 5 8 11 17
After merging L2 into L1
 L1: List size (18), elements : 1 3 4 5 7 8 9 10 11 12 15 17 17 19 25 37 45 62
 L2: List size (0), elements :
```

STL Deque

◆ STL Deque (double-ended queue)

STL deque 주요 멤버함수	설명
begin() end()	첫 번째 원소를 가리키는 반복자(iterator)를 반환 마지막 원소 다음 (after end) 위치를 가리키는 반복자(iterator)를 반환
at(n) operator[n] front() back()	n 번째 원소를 참조할 때 사용, 범위 점검을 함 제공하지 않음 첫 번째 원소의 참조를 반환 마지막 원소의 참조를 반환
empty() size() resize()	원소가 아무것도 없으면 true, 하나라도 존재하면 false 원소의 개수를 반환 deque의 크기를 변경하고, default 값이나 임의의 값으로 초기화
clear() erase() insert() push_front() pop_front() push_back() pop_back() swap() sort() random_shuffle()	모든 원소를 제거 지정된 위치의 원소나 지정된 범위의 원소를 삭제 지정된 위치에 지정된 값을 삽입 deque의 맨 처음에 원소를 추가 eque의 맨 처음 원소를 제거 deque의 끝에 원소를 추가 deque의 마지막 원소를 제거 dq1.swap(dq2)일 때 dq1과 dq2를 swap시킴 제공되지 않음, <algorithm> 라이브러리에서 제공 제공되지 않음, <algorithm> 라이브러리에서 제공

STL Deque 응용 예제

```
/* main() STL deque (1) */
#include <iostream> // I/O definitions
#include <fstream>
#include <deque> // provides deque
#include <algorithm> // for sort, random_shuffle
using namespace std; // make std:: accessible

template<typename T>
void fprintDeque(ostream& fout, deque<T>& dq)
{
    typename deque<T>::iterator p;

    fout << "Deque size (" << dq.size() << ") : ";
    for (p = dq.begin(); p != dq.end(); ++p)
        fout << *p << " ";
    fout << endl;
}

int main()
{
    ofstream fout;

    fout.open("output.txt");
    if (fout.fail())
    {
       cout << "Fail to open output.txt !!";
       exit;
    }
```

```
/* main() STL deque (2) */
    /* Testing vector<int> */
    int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    deque<int> dq_int;
    for (int i = 0; i < 5; i++)
        dq_int.push_front(a[i]);
    fprintDeque<int>(fout, dq_int);

    for (int i = 5; i < 10; i++)
        dq_int.push_back(a[i]);
    fprintDeque<int>(fout, dq_int);

    dq_int.pop_back();
    dq_int.push_back(99);
    fout << "after changing the last element into 99 :" << endl;
    fprintDeque<int>(fout, dq_int);

    fout << "dq_int.front: " << dq_int.front()
         << ", dq_int.back: " << dq_int.back() << endl; // outputs: 5 99
    sort(dq_int.begin(), dq_int.end()); //
    fout << "after sorting the deque : " << endl;
    fprintDeque<int>(fout, dq_int);
    dq_int.erase(dq_int.end() - 5, dq_int.end());
    fout << "after erasing the last 5 elements :" << endl;
    fprintDeque<int>(fout, dq_int);

    fout.close();
}
```

```
Deque size (5) : 5 4 3 2 1
Deque size (10) : 5 4 3 2 1 6 7 8 9 10
after changing the last element into 99 :
Deque size (10) : 5 4 3 2 1 6 7 8 9 99
dq_int.front: 5, dq_int.back: 99
after sorting the deque :
Deque size (10) : 1 2 3 4 5 6 7 8 9 99
after erasing the last 5 elements :
Deque size (5) : 1 2 3 4 5
```

Homework 7

Homework 7

7.1 Class Time 구현

```
class Time
{
    friend ostream&  operator<<(ostream&, const Time&);
public:
    Time(); // default constructor
    Time(int h, int m, int s);
    Time& operator+(int s);
    int elasedSec();
    Time getTime();
    bool operator<(Time&);
    bool operator<=(Time&);
    bool operator>(Time&);
    bool operator>=(Time&);
    bool operator==(Time&);
private:
    bool isValidTime(int, int, int);
    int hour;
    int min;
    int sec;
};
```

7.2 class Person 구현

```
class Person
{
    friend ostream & operator<< (ostream &, const Person &);
public:
    Person();
    Person(string n);
protected:
    string name;
};
```

7.3 class Student 구현

```
class Student : public Person
{
    friend ostream & operator<< (ostream &, const Student &);
public:
    Student();  // default constructor
    Student(int id, string n, Time avt, double gpa);
    bool operator<(Student& st); // compare by arrival time
    bool operator<=(Student& st); // compare by arrival time
    bool operator>(Student& st); // compare by arrival time
    bool operator>=(Student& st); // compare by arrival time
private:
    int st_id;
    double gpa;
    Time arrivalTime;
};
```

7.4 template T_Array 구현

```
template<typename T>
class T_Array
{
public:
   T_Array(int n, string nm); // constructor
   ~T_Array(); // destructor
   string getName() { return name; }
   void reserve(int new_capacity);
   void insert(int i, T element);
   void remove(int i);
   T& at(int i);
   void set(int i, T& element);
   void shuffle();
   void selection_sort(SortingOrder sortOrder=INCREASING);
   void quick_sort(SortingOrder sortOrder=INCREASING);
   void fprint(ofstream &fout, int elements_per_line);
   bool isValidIndex(int i);
   T& operator[](int index) { return t_array[index]; }
private:
    T *t_array;
    int num_elements;
    int capacity;
    string name;
};
```

7.5 main()

```
/* main_T_Array.cpp (1) */
/* main_T_Array.cpp */
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include "T_Array.h"
#include "Student.h"

using namespace std;
#define ELEMENTS_PER_LINE 10
#define SAMPLE_LINES 5
#define NUM_ELEMENTS 500
#define MIN_NUM_ELEMENTS 10

#define NUM_STUDENTS 5
Student students[10] =
{
    Student(21811000, string("Kim, G-M"), Date(1990, 10, 5), Time(3, 0, 30), 3.57),
    Student(21611075, string("Yoon, S-M"), Date(1990, 4, 5), Time(7, 30, 0),  4.37),
    Student(21411015, string("Hwang, S-S"), Date(1989, 1, 10), Time(2, 0, 50), 2.72),
    Student(21611054, string("Lee, K-M"), Date(1991, 5, 15), Time(5, 30, 0), 3.35),
    Student(21311340, string("Hong, G-M"), Date(1990, 2, 5), Time(1, 10, 0),  3.89),
};
```

```
/* main_T_Array.cpp (2) */

void main()
{
    ofstream fout;
    T_Array<Student> studentArray(NUM_STUDENTS, "Array of Students");
    Student *pStudent;

    fout.open("output.txt");
    if (fout.fail())
    {
        cout << "Fail to open output.txt file for results !!" << endl;
        exit;
    }
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        pStudent = &students[i];
        studentArray.insert(i, *pStudent);
    }

    fout << "Elements in studentArray after initialization :" << endl;
    studentArray.fprint(fout, 1);

    fout << "Elements in studentArray after sorting :" << endl;
    studentArray.selection_sort(INCREASING);
    studentArray.fprint(fout, 1);

    fout << "Elements in studentArray after sorting :" << endl;
    studentArray.quick_sort(DECREASING);
    studentArray.fprint(fout, 1);

    fout.close();
}
```

◆ 실행 결과 (예)

```
Elements in studentArray after initialization :
Student [ st_id: 21811000, name: Kim, G-M  , arrival: ( 3: 0:30), gpa:  3.57]
Student [ st_id: 21611075, name: Yoon, S-M , arrival: ( 7:30: 0), gpa:  4.37]
Student [ st_id: 21411015, name: Hwang, S-S, arrival: ( 2: 0:50), gpa:  2.72]
Student [ st_id: 21611054, name: Lee, K-M  , arrival: ( 5:30: 0), gpa:  3.35]
Student [ st_id: 21311340, name: Hong, G-M , arrival: ( 1:10: 0), gpa:  3.89]

Elements in studentArray after sorting by increasing order of arrival times :
Student [ st_id: 21311340, name: Hong, G-M , arrival: ( 1:10: 0), gpa:  3.89]
Student [ st_id: 21411015, name: Hwang, S-S, arrival: ( 2: 0:50), gpa:  2.72]
Student [ st_id: 21811000, name: Kim, G-M  , arrival: ( 3: 0:30), gpa:  3.57]
Student [ st_id: 21611054, name: Lee, K-M  , arrival: ( 5:30: 0), gpa:  3.35]
Student [ st_id: 21611075, name: Yoon, S-M , arrival: ( 7:30: 0), gpa:  4.37]

Elements in studentArray after sorting by decreasing order of arrival times :
Student [ st_id: 21611075, name: Yoon, S-M , arrival: ( 7:30: 0), gpa:  4.37]
Student [ st_id: 21611054, name: Lee, K-M  , arrival: ( 5:30: 0), gpa:  3.35]
Student [ st_id: 21811000, name: Kim, G-M  , arrival: ( 3: 0:30), gpa:  3.57]
Student [ st_id: 21411015, name: Hwang, S-S, arrival: ( 2: 0:50), gpa:  2.72]
Student [ st_id: 21311340, name: Hong, G-M , arrival: ( 1:10: 0), gpa:  3.89]
```