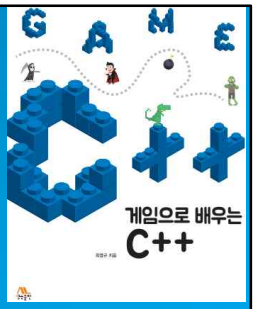
게임으로 배우는 C++



CHAPTER 13

템플릿

몬스터 월드8 (벡터로 만든 세상)

- Monster World 8
  - 벡터를 템플릿으로 구현
  - 동적 배열을 이용
  - 벡터 템플릿으로 몬스터 세상을 만듦

2

13장 학습 목표

- 일반화 프로그래밍을 이해한다.
- 함수 템플릿을 이해하고 활용할 수 있는 능력을 기른다.
- 클래스 템플릿을 이해하고 활용할 수 있다.
- 동적 배열 개념과 벡터를 이해한다.
- 벡터를 템플릿으로 구현할 수 있다.
- 벡터를 이용해 몬스터 세상을 꾸밀 수 있다.

3

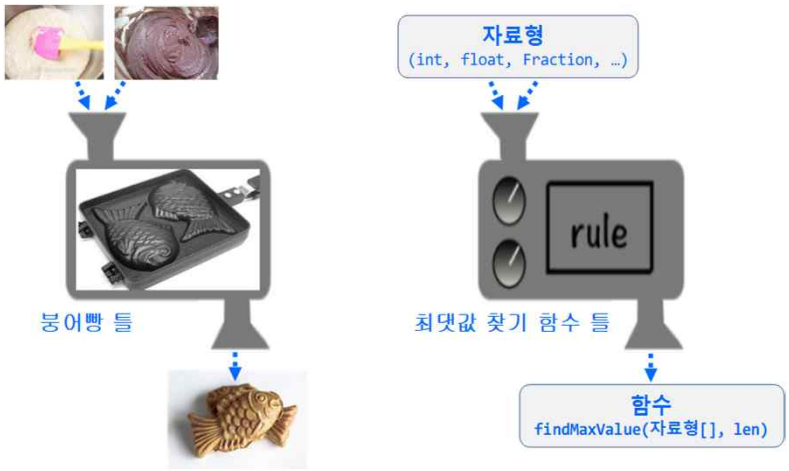
13.1 일반화 프로그래밍

- 일반화 프로그래밍
- C++의 일반화 프로그래밍

4

일반화 프로그래밍

- 클래스나 함수 틀(template)



5

C++의 일반화 프로그래밍

- 일반화 프로그래밍
  - 클래스나 함수 틀을 만드는 것
  - 자료형에 따라 각기 다른 함수나 클래스를 만들지 않음
  - C++ → 템플릿(template)을 통해 지원

- C++의 템플릿 (template)
  - 일반화 프로그래밍을 지원.
  - 컴파일 동안 일어나는 정적 다형성 → 효율적
  - 함수 템플릿과 클래스 템플릿이 있음
  - 키워드: template, typename
  - 형식 매개변수 (또는 타입 매개변수)
    - 꺽은 괄호 "< >"를 사용

6

13.2 함수 템플릿

- 함수 템플릿
- 함수 템플릿의 특수화(specialization)
- 여러 개의 타입 매개변수
- 여러 개의 타입 매개변수

함수 템플릿

- 일반 함수

```
int findMaxValue(int a[], int len)
{
    int maxVal = a[0];
    for( int i=1 ; i<len ; i++ )
        if( maxVal < a[i] )
            maxVal = a[i];
    return maxVal;
}
```

```
void main() {
  int iArr[5] = {1,...};

  int iMax=findMaxValue(iArr,5);
  cout << "iMax = "
       << iMax << endl;
}
```

- **함수 템플릿**

```
template < typename T >
T findMaxValue(T a[], int len)
{
    T maxVal = a[0];
    for (int i = 1; i<len; i++)
        if (maxVal < a[i])
            maxVal = a[i];
    return maxVal;
}
```

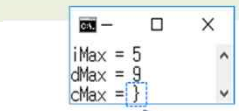
```
void main() {
  int iArr[5] = { 1, 4, 2, 5, 3 };
  double dArr[5] = { 5.0,8.,7.,9.,6};
  char cArr[] = "game over ! {09}";
  int iMax = findMaxValue(iArr,5);
  double dMax = findMaxValue(dArr,5);
  char cMax = findMaxValue(cArr,
                strlen(cArr));
  cout << "iMax = " << iMax << endl;
  cout << "dMax = " << dMax << endl;
  cout << "cMax = " << cMax << endl;
}
```

```
iMax = 5
dMax = 9
cMax = }
```

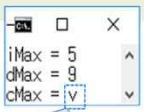
함수 템플릿의 특수화(specialization)

- 특정한 매개변수에 대해서 다른 동작을 하고 싶을 때

```
template <>
char findMaxValue(char a[], int len)
{
        char maxVal = 'a'-1;
        for (int i = 0; i<len; i++)
                if ('a' <= a[i] && a[i] <= 'z' && maxVal < a[i])
                        maxVal = a[i];
        return maxVal;
}
```



아스키 코드 중에서 가장 큰 값인 '}'가 출력됨

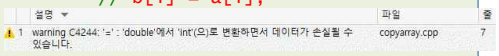


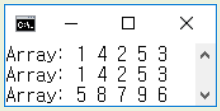
문자열 " game over ! (09) " 에서 알파벳 중에 가장 뒤에 있는 'v'가 출력됨

9

여러 개의 타입 매개변수

```
template < typename T1, typename T2 >
void copyArray(T1 a[], T2 b[], int len) {
        for (int i = 0; i < len; i++)
                b[i] = (T2)a[i];          // b[i] = a[i];
}
template < typename T >
void printArray(T a[], int len) {
        cout << "Array: ";
        for (int i = 0; i < len; i++)
                cout << a[i] << " ";
        cout << endl;
}
void main() {
        int iArr[5] = { 1, 4, 2, 5, 3 };
        float fArr[5];
        double dArr[5] = { 5.0, 8.0, 7.0, 9.0, 6.0 };
        copyArray(iArr, fArr, 5);
        printArray(iArr, 5);
        printArray(fArr, 5);
        copyArray(dArr, iArr, 5);
        printArray(iArr, 5);
}
```





10

클래스가 타입 매개변수로 사용되는 경우

```
class Complex {
        double   real, imag;
public:
        Complex(double r=0.0, double i=0.0): real(r), imag(i) { }
};

template < typename T >
void printArray(T a[], int len) {
        cout << "Array: ";
        for (int i = 0; i < len; i++)
                cout << a[i] << " ";       // 오류: cout << 복소수객체 ?
        cout << endl;
}

void main() {
        Complex cArr[3] = { Complex(1, 1), Complex(2, 2), Complex(3, 3) };
        printArray(cArr, 3);
}
```

```
        friend ostream& operator << (ostream& os, const Complex& c) {
                os << "(" << c.real << "," << c.imag << ")";
                return os;
        }
```

```
설명                                                    파일                줄
1  error C2679: 이항 '<<' : 오른쪽 피연산자로 'Complex' 형식을 사용하는 연산자가 없거나 허용되는 변환이 없습니다.   copyarraycomplex.cpp  17
```

```
C:₩WINDOWS₩syst...
Array: (1,1) (2,2) (3,3)
```

11

13.3 클래스 템플릿

- 클래스 템플릿
- Point 클래스 템플릿
- Point 템플릿 구현
- 멤버 함수를 외부에서 정의하기
- 여러 개의 타입 매개변수
- 템플릿의 유용한 정보

12

클래스 템플릿

- 클래스를 찍어내는 틀

- 템플릿 정의

```
template < typename T >   //또는 <typename T1,..., typename Tn>
class 템플릿명
{
        ...               // 클래스 몸체
};
```

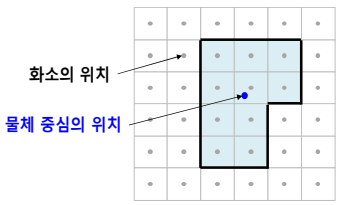
- 객체 선언

```
템플릿명 < 타입1, 타입2, ... 타입n > 객체이름;
```

13

Point 클래스 템플릿

- 두 가지 자료형의 Point
  - 화소의 위치: 정수형
  - 물체 중심의 위치: 실수형
  - → Point 템플릿 사용



- Point 클래스 구현
  - 정수형 Point로만 사용

```
class Point {
        int x, y;
public:
        Point(int xx = 0, int yy = 0) : x(xx), y(yy) { }
        double magnitude() { return sqrt((double)x*x + y*y); }
        Point operator+(Point p) { return Point(x + p.x, y + p.y); }
        Point operator-(Point p) { return Point(x - p.x, y - p.y); }
        friend Point operator*(double s, Point p) {...}
        friend ostream& operator << (ostream& os, const Point& p) {...}
};
```

14

Point 템플릿 구현

```
template <typename T>
class Point {
        T x, y;
public:
        Point(T xx = 0, T yy = 0) : x(xx), y(yy) { }
        double magnitude() { return sqrt((double)x*x + y*y); }
        Point operator+(Point p) { return Point(x + p.x, y + p.y); }
        Point operator-(Point p) { return Point(x - p.x, y - p.y); }
        friend Point operator*(double s, Point p) {
                return Point((T)(s*p.x), (T)(s*p.y));
        }
        friend ostream& operator << (ostream& os, const Point& p) {...}
};
```

타입 매개변수가 int 인 새로운 Point 클래스가 만들어 짐

새로운 클래스의 객체 p1과 p2가 만들어 짐

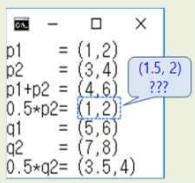
```
Point <int>     p1(1,2), p2(3,4);

Point <double>  q1(5.0,6.0), q2(7.0,8.0);
```

타입 매개변수가 double 인 새로운 Point 클래스가 만들어 짐

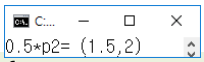
새로운 클래스의 객체 q1과 q2가 만들어 짐

Point 템플릿 구현

```
void main() {
        Point <int> p1(1,2), p2(3,4);
        Point <double> q1(5.0,6.0), q2(7.0,8.0);
        cout << "p1 = " << p1 << endl;
        cout << "p2 = " << p2 << endl;
        cout << "p1+p2 = " << p1+p2 << endl;
        cout << "0.5*p2= " << 0.5*p2 << endl;
        cout << "q1 = " << q1 << endl;
        cout << "q2 = " << q2 << endl;
        cout << "0.5*q2= " << 0.5*q2 << endl;
}
```



- 함수의 반환형 수정
  - 실수와 Point의 곱을 항상 실수로 처리함

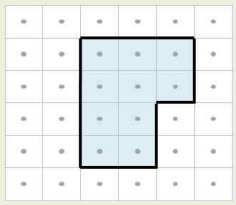


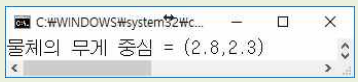

```
        friend Point<double> operator*(double s, Point p) {
                return Point<double>(s*p.x, s*p.y);
        }
```

사용 예

- 영상에서 물체의 무게 중심 계산

```
void main() {
  int image[6][6] = { { 0, 0, 0, 0, 0, 0 },
                      { 0, 0, 1, 1, 1, 0 },
                      { 0, 0, 1, 1, 1, 0 },
                      { 0, 0, 1, 1, 0, 0 },
                      { 0, 0, 1, 1, 0, 0 },
                      { 0, 0, 0, 0, 0, 0 }
  };
  Point <double> sum(0.0,0.0), center;
  int nPixel = 0;
  for (int y = 0; y < 6; y++)
  for (int x = 0; x < 6; x++) {
    if (image[y][x] == 1) {
        Point <double> p(x,y);
        sum = sum + p;
        nPixel++;
    }
  }
  center = (1.0 / nPixel) * sum;
  cout << "물체의 무게 중심 = " << center << endl;
}
```





멤버 함수를 외부에서 정의하기

- 클래스 내부 구현

```
Point operator-(Point p) { return Point(x - p.x, y - p.y); }
```

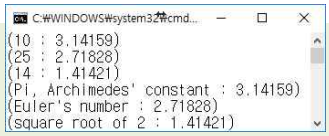
- 클래스 외부 구현 (cpp 파일)

```
template <typename T>
Point<T> Point<T>::operator-(Point p) {
        return Point(x - p.x, y - p.y);
}
```

- 클래스 외부 inline 구현

```
template <typename T>
inline Point<T> Point<T>::operator-(Point p) {
        return Point(x - p.x, y - p.y);
}
```

여러 개의 타입 매개변수




```
template <typename T1, typename T2>
class Pair {
    T1   data1;
    T2   data2;
public:
    Pair() { }
    void set(T1 d1, T2 d2) { data1 = d1; data2 = d2; }
    friend ostream& operator << (ostream& os, const Pair& p) {...}
};
void main() {
        Pair<int, double> i2d[3];
        i2d[0].set(10, 3.14159);
        i2d[1].set(25, 2.71828);
        i2d[2].set(14, 1.41421);
        for (int i = 0; i < 3; i++)  cout << i2d[i] << endl;

        Pair<string, double> map[3];
        map[0].set("Pi, Archimedes' constant", 3.14159);
        map[1].set("Euler's number", 2.71828);
        map[2].set("square root of 2", 1.41421);
        for (int i = 0; i < 3; i++) cout << map[i] << endl;
}
```


템플릿의 유용한 정보

- typedef 의 사용

```
typedef Pair<string, double> str2dbl;
str2dbl map[3];
```

- 디폴트 타입 매개변수 사용

```
template <typename T1=string, typename T2=double>
```

- 템플릿의 상속

```
class WordPair : public Pair< string, string > { ... }
```

- 함수의 매개 변수나 반환형으로 사용 가능

```
Point<double> average(Point<int> arr[], int len) { ... }
```

- C++ 표준 템플릿 라이브러리
  - 14장에서 공부함.

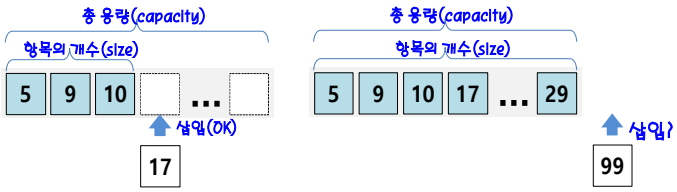
13.4 응용: 벡터 템플릿

- 벡터 템플릿 (심화학습)
- 동적 배열의 개념
- 클래스 다이어그램
- 클래스와 템플릿 비교
- 다차원으로의 확장

21

벡터 템플릿 (심화학습)

- 배열의 크기 고정 문제

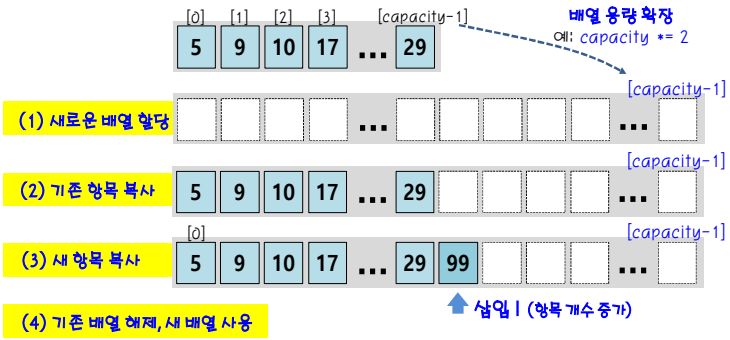


- 해결 방안
  - 크기가 동적으로 변할 수 있는 배열 → 동적 배열?
    → 템플릿으로 벡터를 구현해 보자.
  - STL에서 제공하는 vector 템플릿을 참고함

22

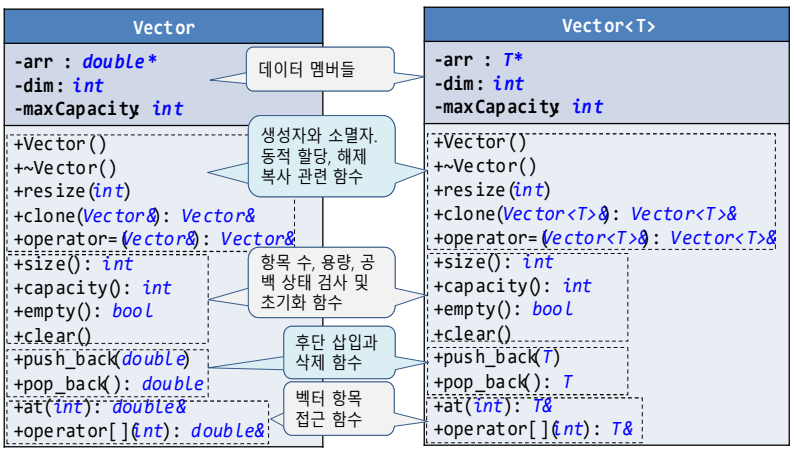
동적 배열의 개념

- 동적 배열에서 항목 삽입 과정(기존 배열이 포화일 때)



클래스 다이어그램

- STL의 벡터를 참고하여 설계



클래스와 템플릿 비교

```
class Vector {
  double* arr;
  int    dim;
  int    maxCapacity;
public:
  ...
  void resize(int size) {
    if (size > maxCapacity) {
      double* old = arr;
      maxCapacity = size;
      arr = new double[maxCapacity];
      for (int i = 0; i < dim; i++)
         arr[i] = old[i];
      delete[] old;
    }
  }
  void push_back(double val) {
    if (dim == maxCapacity)
      resize(maxCapacity * 2);
    arr[dim++] = val;
  }
};
```

```
template <typename T>
class Vector {
  T*     arr;
  int    dim;
  int    maxCapacity;
public:
  ...
  void resize(int size) {
    if (size > maxCapacity) {
      T* old = arr;
      maxCapacity = size;
      arr = new T[maxCapacity];
      for (int i = 0; i < dim; i++)
         arr[i] = old[i];
      delete[] old;
    }
  }
  void push_back(T val) {
    if (dim == maxCapacity)
      resize(maxCapacity * 2);
    arr[dim++] = val;
  }
};
```

25

사용 예

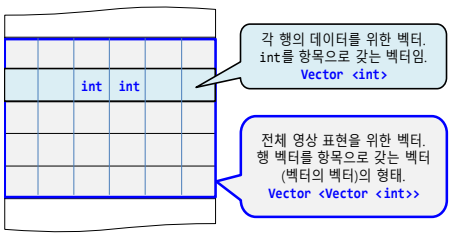
```
void main()
{
        Vector<double> vd(6);
        for (int i = 0; i < vd.size(); i++)
                vd[i] = rand() % 100 * 0.1;
        cout << "Vector<double> = " << vd << endl;
        vd.pop_back();
        cout << "Vector.pop_back() = " << vd << endl << endl;

        Vector<int> vi;
        for (int i = 0; i < 10; i++)
                vi.push_back(rand() % 10);
        cout << "Vector<int> = " << vi << endl << endl;

        Vector<string> vs;
        vs.push_back("hello");
        vs.push_back("world");
        vs.push_back("game");
        vs.push_back("over");
        cout << "Vector<string> = " << vs << endl;
        vs[3] = "I Love";
        vs.push_back("C++");
        cout << "Vector:push_back()= " << vs << endl;
}
```

```
Vector<double>    = < 4.1 6.7 3.4 0 6.9 2.4 >
Vector.pop_back() = < 4.1 6.7 3.4 0 6.9 >

Vector<int>       = < 8 8 2 4 5 5 1 7 1 1 >

Vector<string>    = < hello world game over >
Vector:pop-push() = < hello world game I Love C++ >
```

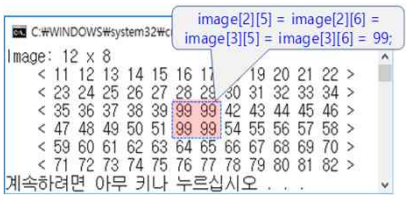
26

다차원으로의 확장

- 벡터의 벡터를 이용한 2차원 맵의 표현




```
Vector<Vector<int>> image(6);
```



27

13.5 응용: MonsterWorld 8

- Monster World 8: 벡터로 만든 세상
- 몬스터 월드 맵 수정
- 몬스터 클래스 수정
- 고찰

28

Monster World 8: 벡터로 만든 세상

- Canvas 클래스
  - 벡터 템플릿 사용

```
class Canvas {
        Vector<string> line;    // 화면 출력을 위한 벡터 객체
        int xMax, yMax;         // 맵의 크기
public:
        Canvas(int nx = 10, int ny = 10): line(ny),xMax(nx),yMax(ny) {...}
        void draw(int x, int y, string val) {...}
        void clear(string val = " ") {...}
        void print(char *title = "<My Canvas>") {...}
};
```

- 몬스터 월드의 몬스터 배열을 벡터로 수정

```
class MonsterWorld
{
        Matrix   world;
        int      xMax, yMax, nMove;
        Vector <Monster*> pMon;
        Canvas   canvas;
        ...
```

29

몬스터 월드 맵 수정

- 몬스터 맵과 몬스터 배열을 벡터로 수정

```
class MonsterWorld
{
        Vector < Vector<int> > world;
        int      xMax, yMax, nMove;
        Vector <Monster*> pMon;
        Canvas   canvas;

        int& Map(int x, int y) { return world[y][x]; }
        ...
public:
        MonsterWorld(int w,int h): world(h),canvas(w,h), xMax(w), yMax(h){
                for (int y = 0; y < yMax; y++)
                        world[y] = Vector<int>(w);
                ... /* 나머지 코드 동일 */
        }
        ...
```

30

몬스터 클래스 수정

- Monster 클래스의 eat()함수

```
//void eat(int** map) {
void eat(Vector<Vector<int>>& map) {
```

- 각종 몬스터 클래스들의 move()함수

```
//virtual void move(int** map, int maxx, int maxy) {
virtual void move(Vector<Vector<int>>& map, int maxx, int maxy) {
```


고찰

- 템플릿을 만드는 과정은 약간 복잡하지만, 사용하는 것은 일반 클래스와 거의 비슷하다. 복잡한 클래스나 함수가 템플릿으로 제공되면 적극적으로 사용해 보아야 할 것이다.
- 몬스터 맵과 같이 2차원 데이터로 Vector를 이용해 손쉽게 만들 수 있었다. 특히 동적 메모리 할당이나 해제를 더 이상 신경 쓰지 않아도 되므로 매우 편리하다.

13장 요약문제, 연습문제, 실습문제

33

감사합니다!

34