Chapter 16. 템플릿

박 종 혁 교수
UCS Lab
(http://www.parkjonghyuk.net)

Tel: 970-6702
Email: jhpark1@snut.ac.kr

Learning Objectives

- 함수 템플릿
  - 구문, 정의
  - 컴파일 합병
- 클래스 템플릿
  - 문법
  - 예 : 배열 템플릿 클래스
- 템플릿 및 상속
  - 예 : 부분적으로 채워진 배열 템플릿 클래스

개요

- C++ templates
  - 함수와 클래스에 대한 매우 "일반"정의 허용
  - 입력한 이름 대신 실제 유형의 "매개 변수”
  - 정확한 정의는 실행 시간에 결정

함수 템플릿

- swapValues:

```
void swapValues(int& var1, int& var2)
{
        int temp;
        temp = var1;
        var1 = var2;
        var2 = temp;
}
```

- 오직 int형 변수에 적용
- 어느 종류의 자료형에서도 동장!

함수 템플릿 vs. 오버로딩

- 문자형에 함수 오버로드 가능:

```
void swapValues(char& var1, char& var2)
{
    char temp;
    temp = var1;
    var1 = var2;
    var2 = temp;
}
```

- 주의: 코드는 거의 동일!
  - 차이점은 3 장소에서 사용되는 유형

함수 템플릿 문법

- swap values 함수는 어느 자료형이든 가능:

```
template<class T>
void swapValues(T& var1, T& var2)
{
        T temp;
        temp = var1;
        var1 = var2;
        var2 = temp;
}
```

- 첫번째 줄은 "template prefix“
  - 컴파일러에게 "template“ 이라고 알려줌
  - T 는 형식 매개변수

Template Prefix

- Recall:
  template<class T>
- "class" 는 "type“이나 "classification“을 의미
- 앞에서 사용한 class와 혼동
  - C++ 은 "class" 자리에 키워드 "typename" 을 허용
  - 하지만 "class" 라고 주로 사용

Template Prefix 2

- Again:
  template<class T>
- T는 모든 종류로 교체 가능
  - 미리 정의되거나 사용자 정의(like a C++ class type)
- 함수 정의부:
  - T 는 다른 형식처럼 사용
- 참고: "T“ 외에 다른 것이 사용되기도 하지만 T 는 "traditional" 사용

함수 템플릿 정의

- swapValues() 함수 템플릿은 매우 큰 정의의 "collection”!
  - 각각의 가능한 유형에 대한 정의!
- 컴파일러는 필요에 따라 정의를 생성
  - 그러나 모든 유형에 대해 정의하는 경우
- 한 개의 정의 → 필요한 모든 유형으로 작동

함수 템플릿 호출

- swapValues(int1, int2);
  - C++ 컴파일러는 템플릿으로 두개의 정수형 인자를 사용하는 함수를 생성
- 다른 타입도 마찬가지
  - 필요한 정의 자동 생성

Another Function Template

- Declaration/prototype:

```
Template<class T>
void showStuff(int stuff1, T stuff2, T stuff3);
```

- Definition:

```
template<class T>
void showStuff(int stuff1, T stuff2, T stuff3)
{
        cout << stuff1 << endl
                << stuff2 << endl
                << stuff3 << endl;
}
```

showStuff Call

- showStuff(2, 3.3, 4.4);
- 컴파일러는 함수의 정의를 생성
  - T를 더블로 교체
    - 두 번째 인자가 더블형이기 때문
- Displays:
  2
  3.3
  4.4

Compiler Complications

- 함수 선언 및 정의
  - 일반적으로 구분이 필요
  - 템플릿 → 모든 컴파일러에서 지원되지 않음!
- 템플릿 함수 정의는 호출하는 파일 안에 안전한 장소에 위치

More Compiler Complications

- 컴파일러의 특정 요구사항을 확인
  - 일부는 특정 옵션 필요
  - 일부는 템플릿 정의 vs 다른 파일 item 들의 배열을 요구
- 대부분 사용 가능한 템플릿 프로그램 레이아웃:
  - 동일한 파일에서 사용되는 템플릿 정의
  - 사용되기 전 확인한 템플릿 정의

Multiple Type Parameters

- 가능:
  template<class T1, class T2>
- 일반적이지 않음
  - 보통 하나의 교체 타입이 필요
  - 사용하지 않는 템플릿 매개 변수를 가질수 없음
    - 각 정의에서 사용해야 함
    - Error otherwise!

알고리즘 추상화

- 템플릿 구현을 의미
- 알고리즘을 표현하는 일반적이 방법:
  - 알고리즘은 모든 타입의 변수에 적용
  - 세부사항을 무시
  - 알고리즘의 중요한 부분에 집중
- 함수 템플릿은 알고리즘 추상화를 지원하는 방법

Defining Templates Strategies

- 일반적 함수 개발
  - 실제 데이터 타입 사용
- 일반적인 함수 완벽하게 디버그
- 템플릿으로 변환
  - 필요한 경우 매개변수와 타입 이름 교체
- 이점:
  - 구체적인 케이스 해결이 더 쉬움

템플릿에서 부적절한 타입

- 템플릿에서 코드는 의미를 만들어 어떠한 타입에서도 사용할 수 있음
  - 코드가 적절한 방식으로 행동해야함
- 예, swapValues() template function
  - 할당 연산자가 정의되지 않은 형식을 사용할 수 없음
  - Example: an array:
    int a[10], b[10];
    swapValues(a, b);
    - Arrays cannot be "assigned"!

Class Templates

- template<class T>
  - 클래스 정의에 적용 가능
  - 클래스 정의에서 "T"의 모든 인스턴스는 형식 매개 변수로 대체
  - 그냥 함수 템플릿처럼!
- 템플릿 정의되면, 클래스의 객체를 선언

Class Template Definition

- template<class T>
  class Pair
  {
  public:
  Pair();
  Pair(T firstVal, T secondVal);
  void setFirst(T newVal);
  void setSecond(T newVal);
  T getFirst() const;
  T getSecond() const;
  private:
  T first; T second;
  };

Template Class Pair Members

- template<class T>
  Pair<T>::Pair(T firstVal, T secondVal)
  {
  first = firstVal;
  second = secondVal;
  }
  template<class T>
  void Pair<T>::setFirst(T newVal)
  {
  first = newVal;
  }

Template Class Pair

- 클래스의 객체는 T 타입 값의 쌍을 갖음
- 다음 개체를 선언 가능:

```
Pair<int> score;
Pair<char> seats;
```

- Example uses:

```
score.setFirst(3);
score.setSecond(0);
```

Pair Member Function Definitions

- 멤버 함수 정의:
  - 각각의 정의는 템플릿 자체
  - 정의 전에 템플릿 prefix 필요
  - Class name before :: is "Pair<T>"
    - Not just "Pair"
  - 생성자 이름 "Pair"
  - 소멸자 이름 "~Pair"

Class Templates as Parameters

- int addUP(const Pair<int>& the Pair);
  - call-by-reference
- 표준 타입에 템플릿 타입을 사용 가능

Restrictions on Type Parameter

- T에서 합당한 타입은 대체 가능
- 고려:
  - 할당연산자는 "well-behaved"
  - 복사 연산자 작동
  - T가 포인터를 포함하면, 소멸자가 적합해야 함
- 함수 템플릿과 유사한 문제

Type Definitions

- 새로운 클래스 타입 이름을 정의 가능
  - 특별한 클래스 템플릿 이름 표현
- Example:
  typedef Pair<int> PairOfInt;
- PairOfInt pair1, pair2;
- 이름도 매개변수로 사용하거나 다른 곳에 타입 이름으로 사용 가능

Friends and Templates

- Friend 함수는 템플릿 클래스와 함께 사용가능
  - 일반 클래스와 동일
  - 단순히 적절한 형식 매개변수를 요구
- 템플릿 클래스의 friend 사용은 일반적
  - 특히 연선자 오버로드

Predefined Template Classes

- vector class
  - 템플릿 클래스
- Another: basic_string template class
  - 어떤 타입 요소의 문자열을 다룸
  - 예,

basic_string<char> works for char's
basic_string<double> works for doubles
basic_string<YourClass> works for YourClass objects

basic_string Template Class

- "string“
  - basic_string<char>
  - 모든 멤버함수는 basic_string<T>와 유사하게 동작

템플릿 및 상속

- 파생 템플릿 클래스
  - 템플릿이나 비템플릿 클래스에서 파생 가능
  - 파생 클래스는 하나의 템플릿 클래스
- 문법은 일반 클래스에서 파생된 일반 클래스와 동일

요약

- 함수 템플릿
  - 타입에 대한 매개 변수와 함께 함수를 정의
- 클래스 템플릿
  - 클래스의 부분에 대한 매개변수와 함께 클래스를 정의
- Predefined vector, basic_string는 템플릿 클래스
- 템플릿 베이스 클래스에서 파생된 템플릿 클래스 정의 가능