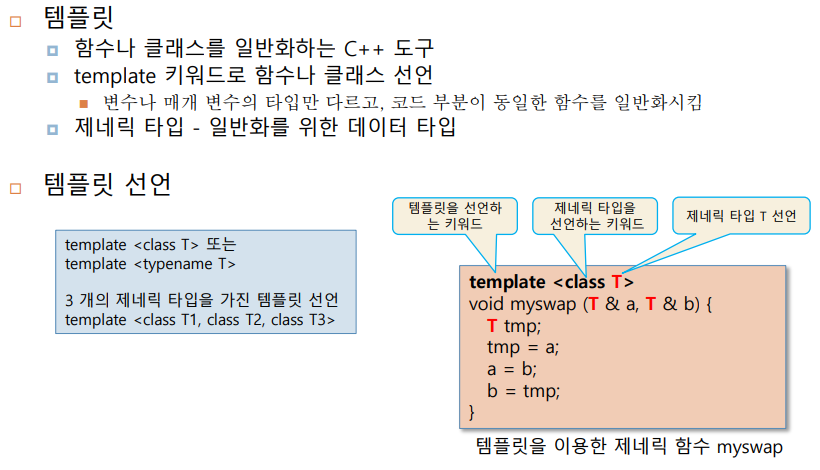
객체지향프로그래밍 (202004100 장주훈, 202084023 이병길)

**12주차** – **템플릿과 표준 탬플릿 라이브러리**

템플릿 선언을 통해 함수의 중복을 제거 가능

일반화: 여러가지 사례를 통해 어디서나 볼 수 있는 공식 (함수나 클래스를 일반화시키고, 매개 변수 타입을 지정하여 틀에서 찍어 내듯이 함수나 클래스 코드를 생산하는 기법)





텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

제네릭 타입 T는 안의 형태(int double float char등 안의 타입에 따라 일반화되어 나타나짐.)

|  |
| --- |
| #include <iostream>  using namespace std;  class Circle {  int radius;  public:  Circle(int radius = 1) { this->radius = radius; }  int getRadius() { return radius; }  };  template <class T>  void myswap(T& a, T& b) {  T tmp;  tmp = a;  a = b;  b = tmp;  }  int main() {  int a = 4, b = 5;  myswap(a, b);  cout << "a=" << a << ", " << "b=" << b << endl;  double c = 0.3, d = 12.5;  myswap(c, d);  cout << "c=" << c << ", " << "d=" << d << endl;  Circle donut(5), pizza(20);  myswap(donut, pizza);  cout << "donut반지름=" << donut.getRadius() << ", ";  cout << "pizza반지름=" << pizza.getRadius() << endl;  } |

템플릿은 틀(직접 호출되지 않음) – 구체화(구체화 과정은 안보임) 후 컴파일 수행

가독성은 사람마다 다름.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| #include <iostream>  using namespace std;  template <class T>  T bigger(T a, T b) {  if (a > b)  return a;  else  return b;  }  int main() {  float a = 20.5, b = 5.0;  char c = 'a', d = 'z';  cout << "bigger(20.5, 5.0)의 결과는 " << bigger(a, b) << endl;  cout << "bigger('a', 'z')의 결과는 " << bigger(c, d) << endl;  } |

템플릿을 작성할 때 Tip! 함수에서 int형 입력으로 함수를 짜고 확인한 후, 일반화 T로 변환.

Ex) 10–3

|  |
| --- |
| #include <iostream>  using namespace std;  int add(int data[], int n) { // 배열 data에서 n개의 원소를 합한 결과를 리턴  int sum = 0;  for (int i = 0; i < n; i++) {  sum += data[i];  }  return sum; // sum와 타입과 리턴 타입이 모두 T로 선언되어 있음  }  int main() {  int x[] = { 1,2,3,4,5 };  int d[] = { 1, 2, 3, 4, 5, 6 };  cout << "sum of x[] = " << add(x, 5) << endl; // 배열 x와 원소 5개의 합을 계산  cout << "sum of d[] = " << add(d, 6) << endl; // 배열 d와 원소 6개의 합을 계산  } |
| #include <iostream>  using namespace std;  template <class T>  T add(T data[], int n) { // 배열 data에서 n개의 원소를 합한 결과를 리턴  T sum = 0;  for (int i = 0; i < n; i++) {  sum += data[i];  }  return sum; // sum와 타입과 리턴 타입이 모두 T로 선언되어 있음  }  int main() {  int x[] = { 1,2,3,4,5 };  double d[] = { 1.2, 2.3, 3.4, 4.5, 5.6, 6.7 };  cout << "sum of x[] = " << add(x, 5) << endl; // 배열 x와 원소 5개의 합을 계산  cout << "sum of d[] = " << add(d, 6) << endl; // 배열 d와 원소 6개의 합을 계산  } |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

중복함수가 먼저 실행되고 템플릿이 실행된다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

|  |
| --- |
| #include <iostream>  using namespace std;  template <class T>  class MyStack {  int tos;// top of stack  T data[100]; // T 타입의 배열. 스택의 크기는 100  public:  MyStack();  void push(T element); // element를 data [] 배열에 삽입  T pop(); // 스택의 탑에 있는 데이터를 data[] 배열에서 리턴  };  template <class T>  MyStack<T>::MyStack() { // 생성자  tos = -1; // 스택은 비어 있음  }  template <class T>  void MyStack<T>::push(T element) {  if (tos == 99) {  cout << "stack full";  return;  }  tos++;  data[tos] = element;  }  template <class T>  T MyStack<T>::pop() {  T retData;  if (tos == -1) {  cout << "stack empty";  return 0; // 오류 표시  }  retData = data[tos--];  return retData;  }  int main() {  MyStack<int> iStack; // int 만 저장하는 스택  iStack.push(3);  cout << iStack.pop() << endl;  MyStack<double> dStack; // double 만 저장하는 스택  dStack.push(3.5);  cout << dStack.pop() << endl;  MyStack<char>\* p = new MyStack<char>(); // char만 저장하는 스택  p->push('a');  cout << p->pop() << endl;  delete p;  } |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

**vector 컨테이너**

특징: 가변 길이 배열을 구현한 제네릭 클래스

* 개발자가 벡터의 길이에 대한 고민할 필요 없이 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원 및 벡터에 저장된 원소는 인덱스로 접근 가능
* 인덱스는 0부터 시작

**iterator 사용**

**텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명**



map 컨테이너

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

STL 알고리즘 사용하기

sort() 함수: 두 개의 매개 변수 -> 첫 번째 매개 변수 : sorting을 시작한 원소의 주소, 두 번째 매개 변수 : sorting범위의 마지막 원소 다음 주소

**텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명**



자동으로 타입추론

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명**