Notes on Modern C++ (c++11, 14, 17, 20)

Gyubeom Edward Im*

July 6, 2024

Contents

1 Introduction

2 Intermediate

2.1 Temporary

2.2 Trivial constructor

2.2.1 Trivial default constructor

생성자가 trivial하다는 말은 컴파일러가 자동으로 생성해주면서 동시에 아무 일도 하지 않을 때를 말한다

2.2.2 Trivial copy constructor

- 복사 생성자가 trivial하다는 말은 멤버변수 값을 복사하는 것 이외에 아무 일도 하지 않을 때를 말한다
- 복사 생성자가 trivial하다면 배열 전체를 memcpy와 memmove 등으로 복사하는 것이 빠르다!
- 복사 생성자가 trivial하지 않다면 배열의 모든 요소에 대해 하나씩 "복사 생성자"를 호출해서 생성자를 호출해야 한다

```
1  struct Point {
2    int x=0;
3    int y=0;
4  };
5
6  template<class T>
7  void constexpr copy_type(T* dst, T* src, std::size_t sz) {
8    if(std::is_trivially_copy_contructible_v<T>) {
9      std::cout << "using memcpy" << std::endl;
10     memcpy(dst, src, sizeof(T)*sz);
```

*blog: alida.tistory.com, email: criterion.im@gmail.com

```
  }
  else {
    std::cout << "using copy ctor" << std::endl;
    while(sz--){
      new(dst) T(*src);
      --dst, --src;
    }
  }
}

int main)(){
  Point arr1[5];
  Point arr2[5];
  copy_type(arr1, arr2, 5);
}
```

- 위 코드에서 Point 클래스는 int x,y와 같이 간단한 멤버변수만 존재하므로 trivial copy constructor이다.
- 하지만 virvual void foo() 같이 가상함수를 사용하거나 string s;와 같이 복사하는 클래스를 사용하게 되면 trivial하지 않게 된다 → 이런 경우에는 placement new 또는 std::construct_at 사용해야함!

2.3 Type deduction

컴파일 타임에 타입이 결정되는 auto 키워드에 대해 살펴보자

```
int main(){
  int n=10;
  cosnt int c =10;

  auto a1 = n;  // int a1=n;
  auto a2 = c;  // (1) const int a2 = c; --> no.
                // (2) int a2 = c;     --> ok.
}
```

- type deduction(타입 추론)이 발생하는 키워드는 다음과 같다: **template, auto, decltype**

```
#include <iostream>
template<class T> void foo(T arg){
  std::cout << typeid(T).name() << std::endl;
}
int main(){
  int n=10;
  foo(n);                    // T=int
  foo<const int&>(n); // T=const int&. But typeid(T).name() keep printing output 'int'
}
```

typeid(T).name()은 타입 이름만 추론할 뿐 const/volatile/reference 정보가 출력되지 않는다. 1. 이럴 때는 의도적으로 에러를 발생시켜서 정확한 타입을 에러 메시지를 통해 알 수 있다. 2. 또는 boost::type$_{i}ndex|tfitype_{i}d_{w}ith_{c}vr < T > ().pretty_{n}ame()DXt(3.fi|\Gamma\o slash Xl\|Xt\o slash .{}_{F}UNCTION_{:h_Xt\`XJL}{}_{P}RETTY_{F}UNCTION_{:g++,clang\o slash h_tt4}{}_{F}UNCSIG_{:cl.exe\Xi std::cout<<}{}_{F}UNCTION$

int main() int n=10; int r = n; const int c = 10; const int cr = c; foo(n); // T=int foo(r); // T=int 일 것 같지만 T=int foo(c); // T=const int 일 것 같지만 T=int foo(cr); // T=const int 일 것 같지만 T=int T 인자를 값으로 받을 때는 복사본 객체가 만들어져서 “const, volatile, reference” 속성을 제거하고 값만 받는다. 헷갈리는 것 중 하나가 값으로 받을 때는 인자의 const 속성은 제거되고 “인자가 가리키는 곳의 const 속성은 유지"한다. 무슨 이야기인지 살펴보자 include <iostream> template<class T> void foo(T arg) std::cout «

$_{P}RETTY_{F}UNCTION_{<<std::endl;}intmain()constchar*consts="hello";foo(s);//x0Xconst1@\o slash p\_\Gamma"hello"Xconst1@(!//constchar*arg="hello"\Gamma(!!Tx|8$

3 References

[1] (lecture) CODENURI - C++ Master

4 Revision log

- 1st: 2024-07-16