**함수의 선언과 정의 분리**

#include <iostream>

int add(int a, int b);

using namespace std;

int main() {

cout<<add(1, 3)<<endl;

}

int add(int a, int b) {

return a + b;

}

**헤더파일 만들기**

위에 add 함수를 파일로 분리해보자

add.cpp 파일을 하나 만든다.

*add.cpp*

int add(int a, int b) {

return a + b;

}

*add.h*

int add(int a, int b);

*main*

#include “add.h”

**헤더가드가 필요한 이유**

=> 함수의 중복 선언으로 생기는 에러를 막기 위해서

#pragma once

**Namespace**

#include <iostream>

using namespace std; //namespace 사용

namespace Myspace1{

int dosomething(int a, int b) {

return a + b;

}

}

int dosomething(int a, int b) {

return a \* b;

}

int main() {

cout << Myspace1::dosomething(3, 4) << endl; //7

cout << dosomething(3, 4) << endl;//12

}

**전처리기와의 만남**

# define MYNUM 9

MYNUM을 만나면 9로 그냥 바꿔주는 문서 편집기 같은 역할을 한다.

**std::string**

#include <iostream>

#include <string>

using namespace std;

int main() {

cout << "name : ";

string name;

cin >> name;

cout << "age : ";

string age;

cin >> age;

cout << name <<" " << age << endl;

}



띄어쓰기를 기준으로 버퍼에 담았다가 출력함

*sol*

#include <iostream>

#include <string>

using namespace std;

int main() {

cout << "name : ";

string name;

getline(cin, name);

cout << "age : ";

string age;

getline(cin, age);

cout << name <<" " << age << endl;

}

getline은 엔터 칠 때 까지 (한 줄을) 입력받음

#include <iostream>

#include<string>

using namespace std;

int main() {

char str[255];

cin.getline(str, 255);

cout << str << endl;

}

***buffer 비우는 방법***

cin::ignore(32767,’\n’);

\n (엔터)를 입력하기 전까지 32767 개의 글자를 무시해라.

**c스타일 문자열**

char source[] = "copy";

char dest[50];

strcpy(dest, source);

dest에 source를 복사한다.

 strcat 함수를 사용하여 서로 붙일 수 있다

{

char s1[10] = "world";

char s2[20] = "Hello"; // s2 뒤에 붙일 것이므로 배열 크기를 크게 만듦

**strcat(s2, s1);** // s2 뒤에 s1를 붙임

printf("%s\n", s2); // Helloworld

strcmp (str1, str2)

같으면 0, 다르면 -1을 리턴한다.

널포인터

포인터에 선언하고 할당할 주소가 없으면 널 포인터를 선언한다

#include <iostream>

using namespace std;

int main() {

double\* ptr = 0;// c- style

double\* ptr2 = nullptr; //modernc++

if(ptr2!=nullptr){

}

else {

}

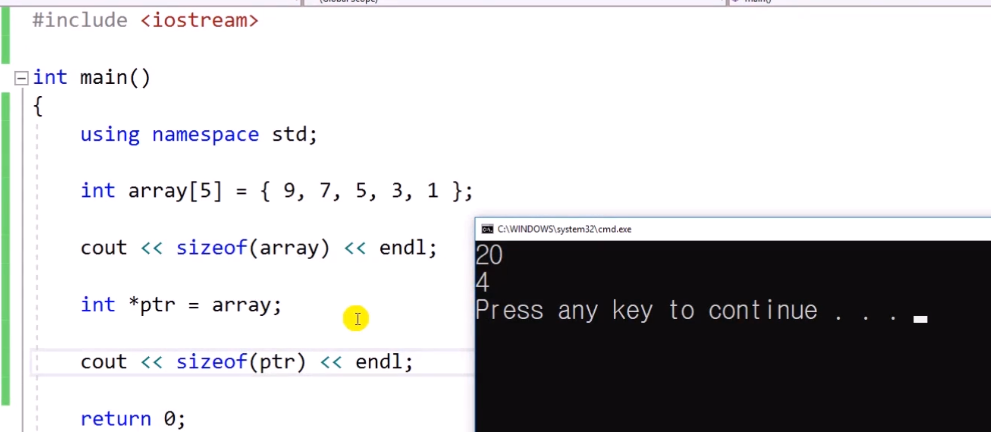
}

**포인터와 정적 배열의 관계 (둘이 같다)**

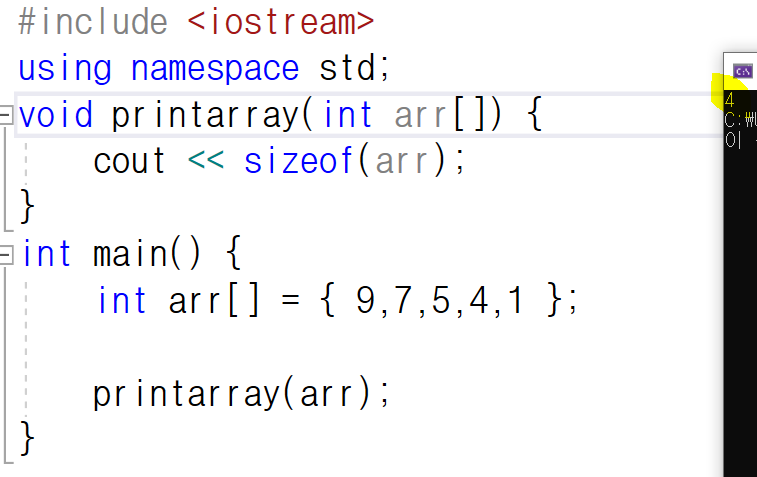
**배열의 이름 = 배열의 포인터이다**

**&arr[0] == arr**

**차이?**



문제점



배열의 사이즈가 안 나오고 포인터 변수의 사이즈를 출력한다

arr이 포인터로 작동한다 => void printarray(int \*arr) 과 같은 역할임

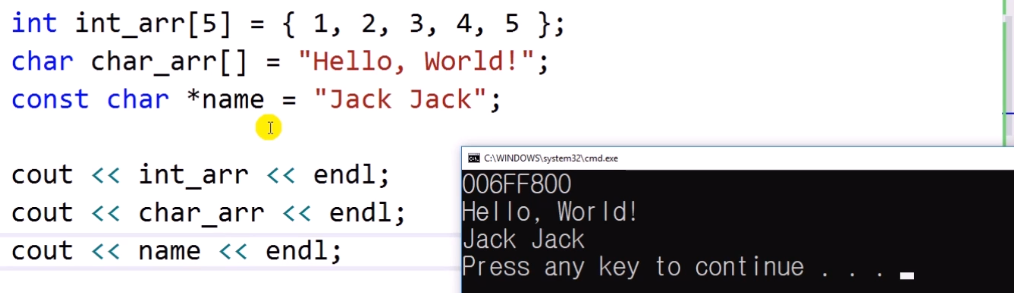
\*ptr , \*(ptr+1)

arr[0] arr[1]

**문자열 기호적 상수**

const char\* name = "Jack jack";

char\* name2 = "jack jack"; //x



문자열 포인터를 출력하면 문자열이 출력됨

**메모리 동적 할당**

int \*ptr = new int; //int 크기의 메모리를 할당 받은 후 그 주소를 알려줌

\*ptr = 7;

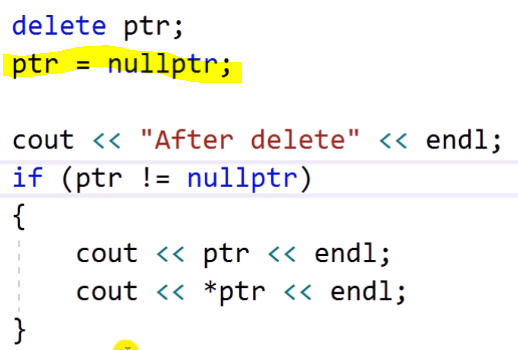
int var;

var = 7;

위 둘이 같은 의미 임

int\* ptr = new int{7};

delete ptr; //메모리 돌려줌



**동적 할당 배열**

미리 배열의 크기가 정해져 있는 정적 할당 배열과 달리

그때 그때 배열을 할당함

int length;

cin >> length;

int\* arr = new int[length];

int\* arr = new int[length](); //모두 0으로 초기화

int\* arr = new int[length] {1, 2, 3, 4};

**포인터와 const**

const int value = 5;

int\* ptr = &value; //x

const int\* ptr = &value; //o

value 자체도 바꾸지 못하므로 pointer도 const로 !

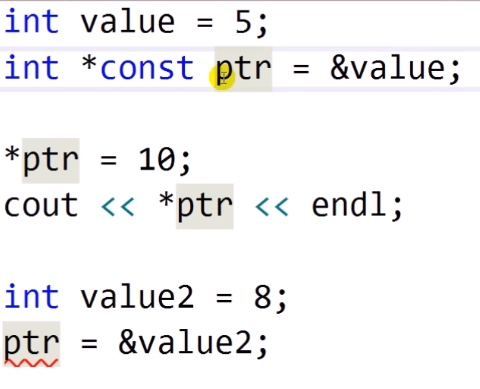
int v1 = 5;

const int\* ptr = &v1;

int v2 = 6;

ptr = &v2;

const int \*ptr 의미는 ptr이 가리키는 값을 변경 시키지 않겠다는 뜻이지 ptr 값은 변경 가능 하므로 ptr=&v2가 가능, 또한 v1=7로 하면 \*ptr이 7로 qusrudehlas

\

포인터 값 자체를 못바꾼다.

**참조변수 reference variable**

int value = 5;

int& ref = value; //ref가 value와 같아짐 즉, &ref == &value (그냥 완전히 같아짐)

cout << ref << endl; //5

value = 6;

cout << ref << endl; //6

int &ref는 반드시 초기화를 해야하고 리터럴이 들어가지 못한다.

#include <iostream>

using namespace std;

void change(int& n) {

n = 20;

}

int main() {

int n = 10;

cout << n << endl; //10

change(n);

cout << n << endl; //20

}

위에 void change(int& n) 은 change(1) 로 할 수 없다,

하지만 void change(const int& n) 으로 하면 change(5), change(a+2) 다 가능함.

int& ref = 5; // x

const int& ref = 5; //o

#include <iostream>

using namespace std;

struct Person {

string name;

int age;

};

int main() {

Person person;

person.age = 39;

person.name = "hi";

Person& ref = person;

ref.age = 10;

ref.name = "you";

Person\* ptr = &person;

ptr->age = 20;

ptr->name = "hey";

}

**for-each 반복문**

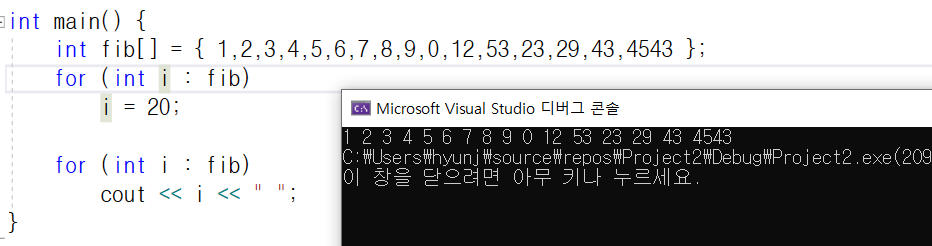
array를 동적할당으로 하면 for-each를 사용할 수 없다.

정적할당 일 때만 사용 가능하다.( 대신 vector를 사용)

const int fib[] = { 1,2,3,4,5,6,7,8,9,0,12,53,23,29,43,4543 };

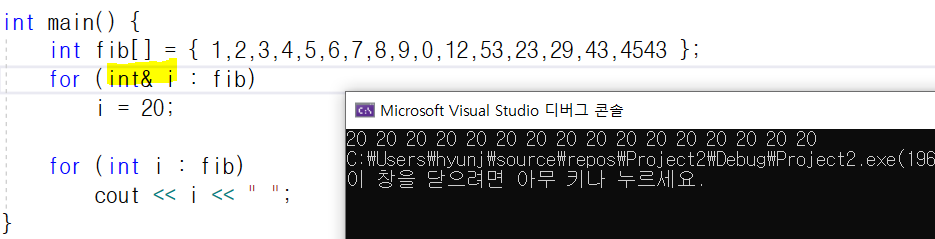
for (int i : fib)

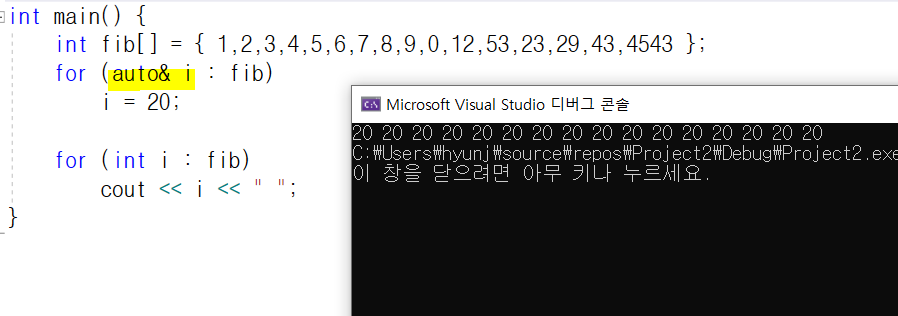
cout << i << endl;



변수 형태 이므로 값이 바뀌지 않음

=> reference 사용





보통 auto로 많이 사용한다.

**void 포인터 : 자료형이 정해지지 않은 포인터**

int a = 2;

double b = 3;

void\* ptr=nullptr;

ptr = &a;

ptr = &b;

다만, pointer의 자료형이 정해지지 않았기 때문에 \*(ptr+1) 이런 연산은 할 수 없다.

cout << \*ptr; //x

cout << (double\*) ptr; //o

ptr에 저장된 값을 알고 싶을 때도 형변환을 해준 뒤 알 수 있다.

**std::array 정적 배열**

#include <iostream>

#include<array>

using namespace std;

int main() {

int arr[5] = { 1,2,3,4,5 };

array<int, 5> arr2 = { 1,2,3,4,5 };

//위 아래 같은 의미

}

#include <iostream>

#include<array>

using namespace std;

int main() {

array<int, 5> arr2 = { 1,2,3,4,5 };

for (int& c : arr2) //& 붙이면 같은 메모리를 쓰므로 메모리양을 줄일 수 있다.

cout << c<<" ";

}

**정렬**

#include <iostream>

#include<array>

#include<algorithm>

using namespace std;

int main() {

array<int, 5> arr2 = { 70,22,3,14,5 };

sort(arr2.begin(), arr2.end());

for (int c : arr2) {

cout << c << " ";

}

}

**reverse 정렬**

#include <iostream>

#include<array>

#include<algorithm>

using namespace std;

int main() {

array<int, 5> arr2 = { 70,22,3,14,5 };

sort(arr2.rbegin(), arr2.rend());

for (int c : arr2) {

cout << c << " ";

}

}

#include <iostream>

#include<array>

using namespace std;

int main() {

array<int, 5> arr2 = { 70,22,3,14,5 };

cout << arr2.size(); //배열 크기 (5)

}

**std::vector** **동적배열**

#include <iostream>

#include<vector>

using namespace std;

int main() {

vector<int> arr = { 1,2,3,4 };

cout << arr.size() << endl; //배열 크기(4)

}

arr[1] 과 arr.at(1) 로 index 1번 원소 호출 가능 => .at은 out of index를 예외 처리 해줌

vector로 동적 할당을 하면 메모리를 delete할 필요가 없다.

