



C++ 스터디

5. 포인터, 벡터, 어레이(배열)

포인터 (& Operator)

- pass by reference에서 사용했던 “& 연산자”를 떠올려보자. 함수에 값을 넘겨줄 때 & 연산자를 사용해 주소 값을 넘겨줌으로써 함수에서도 외부 변수를 수정할 수 있게 하였다.
- 즉 & 연산자는 변수의 주소 값을 반환해주는 **주소 연산자**이다.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int a = 1;
6      cout << "변수의 value : " << a << endl;
7      cout << "변수의 주소값 : " << &a << endl;
8  }
```

Microsoft Visual Studio 디버그 콘솔

```
변수의 value : 1
변수의 주소값 : 00EFF8BC
```

C:\Users\bluej\Desktop\방학프로그래밍

포인터 (주소 값)

- 그렇다면 주소 값은 무엇일까? 데이터의 주소 값은 **해당 데이터가 저장된 메모리의 시작 주소**를 의미한다!
- 예를 들어 int형 데이터(a)는 4바이트의 크기(2^{32} 개의 숫자 표현 가능)를 가진다. 하지만 int형 데이터의 주소 값(&a)은 시작 주소 1바이트만을 가리키게 된다.
- 위의 int형 변수 a를 예시로 들자면 (16진수임)

int a	
00EFF8BC (a의 시작 주소, &a)	(4바이트의 데이터가 나타내는) 값 = 1
00EFF8BD	
00EFF8BE	
00EFF8BF	

그래서 이런
형식은
불가능하다.

```
int a = 1;
int b = &a;
```

"int *" 형식의 값을 사용하여 "int" 형식의 엔터티를 초기화할 수 없습니다.

[온라인 검색](#)

포인터 (포인터 변수)

- C++에서 포인터는 메모리의 시작 주소 값을 저장하는 변수이다. 포인터 변수라고도 하며 사용법은 다음과 같다.
 - 타입* 변수명; // 포인터 변수 (*은 타입에 붙든 변수명에 붙은 상관이 없다.)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int num = 50; // 변수 선언
6      int* address = &num; // num의 주소를
7                          // address 포인터에 주며 선언
8      cout << &num << endl;
9      cout << address << endl;
10 }
11
12 선택 Microsoft Visual Studio 디버그 콘솔
```

```
00C6FCC4
00C6FCC4
```

포인터 (포인터 변수)

- 그렇다면 포인터 변수는 주소 값만 저장하는데 왜 타입이 정해져 있는가 궁금해 할 수 있다.
- 포인터 변수는 시작 주소 값을 저장하고, 타입마다 그 이후로 읽어야 하는 메모리의 크기가 각기 다르기 때문이다!
- 만약 int 정보(4바이트)가 저장되어 있는 메모리를 char 정보(1바이트)를 읽는 방식으로 읽는다면 이상한 값이 나올 것이다.

int a	메모리의 비트 값 (예시)
00EFF8BC (a의 시작 주소, &a)	0001
00EFF8BD	0000
00EFF8BE	0000
00EFF8BF	0000
int가 나타내는 값 : 1	

만약 char* 타입으로
참조한다면
아스키코드 표의
1번인 'SOH'가
읽히게 될 것이다.

포인터 (* operator)

- * 연산자는 **참조 연산자**이다. 포인터나 주소 앞에서 사용해 포인터 저장되어 있는 값을 참조하여 반환한다.
- 주소 연산자(&)의 역산이다.

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int num = 50; // 변수 선언
6      int* address = &num; // num의 주소를
7      // address 포인터에 주며 선언
8      cout << &num << endl;
9      cout << address << endl;
10
11     cout << num << endl;
12     cout << *(&num) << endl;
13     cout << *address << endl;
14 }
```

Microsoft Visual

008FFEB4
008FFEB4
50
50
50

포인터 (pass by pointer)

- 함수의 인자로 포인터 변수를 줌으로써 pass by reference처럼 사용할 수 있다. (어차피 둘 다 주소 값을 주는 것이 됨으로 동일함)

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int * a, int * b) {
5      int temp = *a; // a의 value를 temp에 넣음
6      *a = *b; // b의 value를 a에 넣음
7      *b = temp;
8  }
9
10 int main()
11 {
12     int num1 = 5, num2 = 19;
13     cout << num1 << ", " << num2 << endl;
14     swap(&num1, &num2); // 포인터 변수에 들어갈
15                          // 주소 값을 줌.
16     cout << num1 << ", " << num2 << endl;
17 }
```

포인터 (함수 포인터)

- 포인터 변수는 함수의 주소마저 담을 수 있다. 함수 포인터는 입력과 출력의 타입이 같은 함수를 대상으로 할 수 있다.
- 함수 포인터 변수 선언법은 **타입 (*변수명) (파라미터);**

```
1  #include <iostream>
2  using namespace std;
3
4  int add(int x, int y) {
5      return x + y;
6  }
7
8  int multiply(int x, int y) {
9      return x * y;
10 }
11
12 int evaluate(int (*f)(int, int), int x, int y) {
13     return f(x, y);
14 }
15
16 int main()
17 {
18     cout << evaluate(&add, 2, 3) << endl;
19     cout << evaluate(&multiply, 2, 3) << endl;
20 }
```

Microsoft Visual Studio

5
6
C:\#\source\#\b...

포인터 (함수 포인터)

- 함수 포인터를 사용함으로써 더 유연하고 동적인 프로그램을 제작 가능해진다.
- 대입시에는 함수의 이름을 그냥 써서 넣어줘도 되고, &를 앞에 붙여줘도 된다.

```
1 #include <iostream>
2 using namespace std;
3
4 int add(int x, int y) {
5     return x + y;
6 }
7
8 int main()
9 {
10     int (*func)(int, int);
11     func = add;
12     cout << func(7, 2) << endl;
13 }
14
15
```

Microsoft Visual Studio 디버그 콘솔

```
1 #include <iostream>
2 using namespace std;
3
4 void func1() {
5     cout << "병준" << endl;
6 }
7
8 void func2() {
9     cout << "재혁" << endl;
10 }
11
12 int main() {
13     void (*a)() = func1;
14     a();
15     a = &func2;
16     a();
17 }
```

Microsoft Visual Studio 디버그 콘솔

배열

- 배열(array)은 같은 타입의 변수들로 이루어진 메모리의 데이터 덩어리이다. 전에 배웠던 포인터에서 요소가 한 개가 아니라 여러 개가 붙어있다고 생각하면 된다.
- 배열을 구성하는 각각의 요소들을 배열 요소(element)라고 하고, 배열에서의 위치를 가리키는 숫자를 인덱스(index)라고 한다. 인덱스는 0 부터 시작한다.
- 배열이란 객체가 아니며 멤버 함수도 가지지 않는다.
- 즉 size() 함수나 elem : array 형태도 지원이 되지 않으므로 배열의 size를 기억해야한다.
- 배열의 크기는 고정되거나 (static array) 바뀔 수도 있다. (dynamic array)

배열 (사용법)

- 배열의 길이를 선언할 때에는 반드시 상수를 사용해야 한다.
- 그러므로 따로 const 키워드를 사용해 이름이 붙은 상수를 이용하거나 숫자를 써줘야 한다.

```
1  #include <iostream>
2  using namespace std;
3  // 어레이는 헤더가 따로 필요없음
4  int main() {
5      int list[3];
6      list[0] = 5;
7      list[1] = 3;
8      list[2] = 12;
9  }
```

```
1  #include <iostream>
2  using namespace std;
3  // 어레이는 헤더가 따로 필요없음
4  int main() {
5      const int num = 1;
6      int list[num];
7  }
```

```
1  #include <iostream>
2  using namespace std;
3  // 어레이는 헤더가 따로 필요없음
4  int main() {
5      int num = 1;
6      int list[num];
7  }
```

배열 (pass by reference)

- 배열을 함수의 파라미터로 넘겨주면 메모리의 시작 주소만을 담기에 pass by value로 값을 전달할 수 없다. 즉 함수의 인자로써 배열의 포인터만을 넘겨준다.
- 타입 []을 이용해 파라미터로 배열이 옴을 알려주고 배열의 사이즈도 같이 파라미터로 준다.

```
1  #include <iostream>
2  using namespace std;
3  // 어레이는 헤더가 따로 필요없음
4  void print(int a[], int n) {
5      for (int i = 0; i < n; i++)
6          cout << a[i] << " ";
7      cout << endl;
8  }
9
10 int sum(int a[], int n) {
11     int result = 0;
12     for (int i = 0; i < n; i++)
13         result += a[i];
14     return result;
15 }
16
17 int main() {
18     int list[] = { 2,4,6,8 };
19     print(list, 4);
20     cout << sum(list, 4) << '\n';
21 }
```

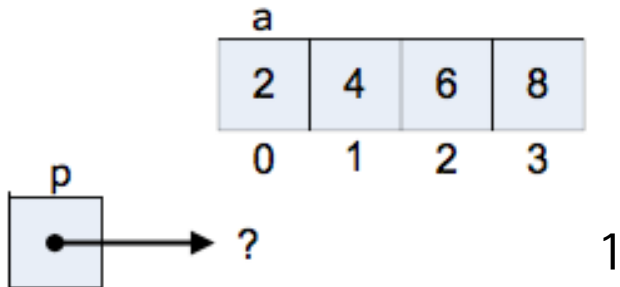
배열 (포인터와 배열)

- 배열은 곧 포인터다!
- $a[n] == *(a + n)$
- $a == \&(a[0])$
- 배열의 시작 주소($\&a[0]$)에 1을 더해줌으로써 (포인터 연산 중 하나) 다음 요소를 볼 수 있다.

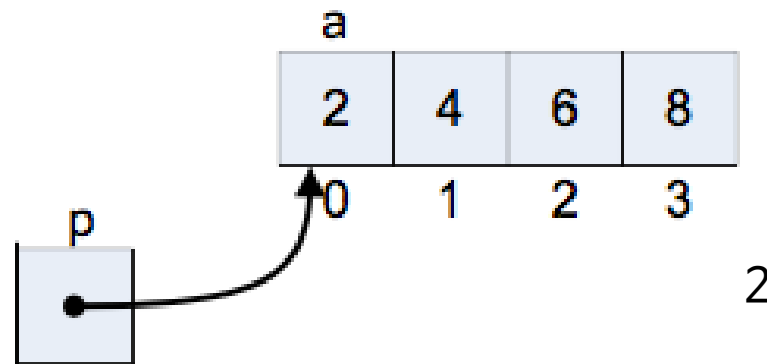
```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int a[] = { 2,4,6,8,10,12,14,16,18,20 }, * p;
6      p = &a[0];
7
8      for (int i = 0; i < 10; i++) {
9          cout << *p << ' ';
10         p++; // 포인터의 값을 1 증가시킬때마다
11              // int의 단위 4바이트 뒤의 메모리 값을 읽음
12              // 즉 배열의 다음값을 나타내게됨
13     }
14     cout << '\n';
15 }
```

배열 (포인터와 배열)

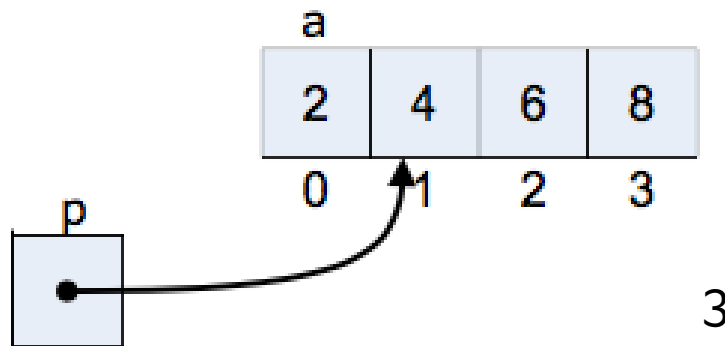
```
int a[] = { 2, 4, 6, 8 }, *p
```



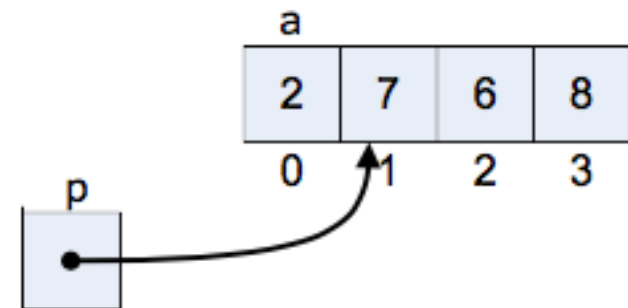
```
p = a;
```



```
p++;
```



```
*p = 7;
```



배열 (포인터와 배열)

- 즉 아래의 두 용법은 형태는 다르지만 같은 것을 나타내게 된다. (어차피 시작 주소 값만 전달되기 때문)

Sometimes pointer notation is used to represent an array parameter to a function. The array print function that begins

```
void print(const int a[], int n)
```

could instead be written

```
void print(const int *a, int n)
```

where a is a pointer to an array. The compiler treats the two forms identically in the machine language it produces.

배열 (포인터와 배열 예제)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int a[] = { 2,4,6,8,10,12,14,16,18,20 },
6          * begin, * end, * cursor;
7      begin = a;
8      end = a + 10;
9      cursor = begin;
10     while (cursor != end) {
11         cout << *cursor << ' ';
12         cursor++;
13     }
14     cout << '\n';
15 }
```


배열 (dynamic array)

- Static array (여태까지 해온 배열들)의 크기는 처음 선언한대로 고정 되어있어 크기를 바꿀 수가 없다.
- 하지만 dynamic array는 size를 마음대로 할당하고 지울 수 있다. -> 동적할당을 통해
- 동적 할당에 대한 자세한 내용은
<https://m.blog.naver.com/cache798/130033385486> 참조
 - new 가 동적 메모리 할당을 위해 사용된다.
double *numbers;
numbers = new double[size];
 - delete 는 동적 메모리 해제를 위해 사용된다. 사용 후에는 파일i/o처럼 반드시 해제 해줘야 한다!
delete[] numbers;

배열 (dynamic array 예시)

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      double sum = 0.0;
6      double* numbers;
7      int size;
8      cout << "처리할 value들의 수를 입력해주세요 : ";
9      cin >> size;
10
11     if (size > 0) {
12         cout << size << "개의 숫자를 입력해 주세요 : ";
13         numbers = new double[size];
14         for (int i = 0; i < size; i++) {
15             cin >> numbers[i];
16             sum += numbers[i];
17         }
18         for (int i = 0; i < size; i++) {
19             cout << numbers[i] << ' ';
20         }
21         cout << "의 평균은 " << sum / size << '\n';
22         delete[] numbers;
23     }
24 }
```

배열 (다차원 배열)

- 다차원 배열은, 2차원 이상의 배열이다. 배열 요소로 또 다른 배열을 가지게 하면 된다.
- 2차원 배열의 **요소** -> 1차원 배열
- 3차원 배열의 **요소** -> 2차원 배열
- 메모리는 입체적 공간이 아닌 선형이므로 메모리에 다음과 같이 정보가 저장된다.

int arr[2][3] (한 칸당 4Byte)



배열 (다차원 배열 예시)



```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  const int ROWS = 3, COLUMNS = 5;
6
7  using Matrix = double[ROWS][COLUMNS];
8
9  void populate_matrix(Matrix m) {
10     cout << ROWS << "개의 열을 채우세요." << endl;
11     for (int row = 0; row < ROWS; row++) {
12         for (int col = 0; col < COLUMNS; col++) {
13             cout << "열 #" << row <<
14                  "행 - " << col << "번째 값 :";
15             cin >> m[row][col];
16         }
17     }
18 }
19
20 void print_matrix(const Matrix m) {
21     for (int row = 0; row < ROWS; row++) {
22         for (int col = 0; col < COLUMNS; col++)
23             cout << setw(5) << m[row][col];
24         cout << '\n';
25     }
26 }
27
28 int main() {
29     Matrix mat;
30     populate_matrix(mat);
31     print_matrix(mat);
32 }
```

배열 (예시)

Listing 11.21: findchar.cpp

```
#include <iostream>

bool find_char(const char *s, char ch) {
    while (*s != '\0') { // Scan until we see the null character
        if (*s == ch)
            return true; // Found the matching character
        s++; // Advance to the next position within the string
    }
    return false; // Not found
}

int main() {
    const char *phrase = "this is a phrase";
    // Try all the characters a through z
    for (char ch = 'a'; ch <= 'z'; ch++) {
        std::cout << '\'' << ch << '\'' << " is ";
        if (!find_char(phrase, ch))
            std::cout << "NOT ";
        std::cout << "in " << "\"" << phrase << "\"\n";
    }
}
```

벡터 (소개 및 사용법)

- C++에서의 벡터는 물리의 스칼라, 벡터 그런 것이 아니고 여러 개의 value를 동시에 담을 수 있는 **메모리 블록(메모리 덩어리)**을 관리하는 객체이다. 즉 Python의 리스트와 비슷하다고 생각하면 된다. - 배열을 자동으로 관리해준다고 생각해도 됨
- 대신 벡터의 내용물은 전부 같은 타입이어야 한다. 그 이유는 벡터가 배열을 바탕으로 하기 때문이다.
- 사용법은 다음과 같다. `vector<타입>` 변수명;
 - `#include <vector>` // 벡터 라이브러리를 include 해준다.
 - `vector<int> vec_a;` // 이름만 가진 벡터 생성
 - `vector<int> vec_b(10);` // 이름과 초기 크기를 가진 벡터 생성
 - `vector<int> vec_c(10,8);` // 이름과 초기 크기, 그리고 초기 값을 가진 벡터 생성('8'이 10개)
 - `vector<int> vec_d{10,20,30,40};` // 이름과 특정 값을 가진 벡터 생성

벡터 (컴퓨터 메모리에서)

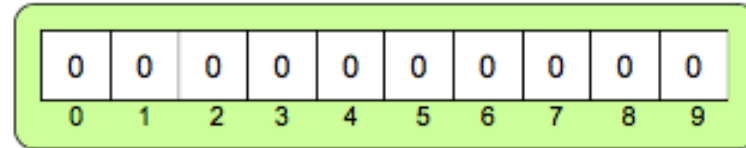
- 벡터는 선형 저장소이다.
 - 벡터는 원소들을 연속적인 메모리 블록에 저장한다.
- 우측 그림의 한 칸이 4바이트이다. (int)

```
vector<int> vec_a;  
vector<int> vec_b(10);  
vector<int> vec_c(10, 8);  
vector<int> vec_d{ 10, 20, 30, 40 };
```

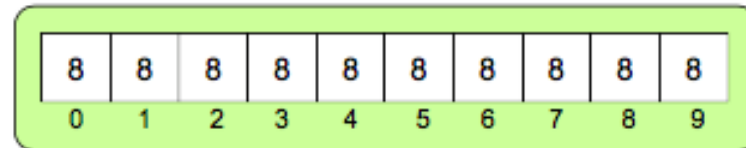
vec_a



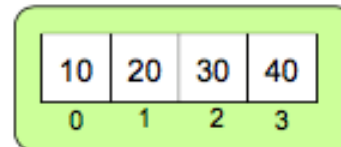
vec_b



vec_c



vec_d



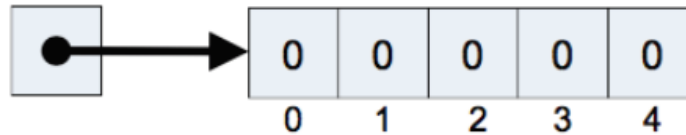
벡터 (벡터 vs 동적 어레이 vs 정적 어레이)

```
int list_1[5];  
int *list_2 = new int[5];  
vector<int> list_3(5);
```

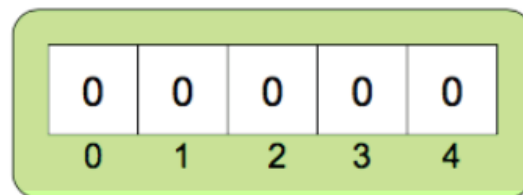
list_1



list_2



list_3



벡터 (요소 접근 및 사용)

- 벡터는 Python의 리스트처럼 한 번에 출력할 수가 없다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<int> vec{ 1,2,3,4 };
7     cout << vec << endl;
8 }
```

error가 발생하면서 컴파일이 안됨

- 대신 각각의 요소에 [] 연산자를 사용해 접근 가능하다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<int> vec{ 1,2,3,4 };
7     cout << vec[0] << endl;
8     cout << vec[1] << endl;
9 }
10
11 Microsoft Visual Studio 디버그
```

벡터 (요소 접근 및 사용)


- 마찬가지로 각각의 요소에 [] 연산자를 사용해 값을 할당할 수 있다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<int> vec{ 1,2,3,4 };
7     cout << vec[0] << endl;
8     cout << vec[1] << endl;
9 }
10
11 Microsoft Visual Studio 디버그
```

대신 벡터의 범위를 벗어난 접근에 대해서는 컴파일은 되지만 실행 도중에러가 발생한다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<int> vec{ 1,2,3,4 };
7     cout << vec[4] << endl;
8 }
9
10
```

Microsoft Visual C++ Runtime Library

 Debug Assertion Failed!

Program: C:\Users\bluej\Desktop\방학프로그램\Tutorial\Project1\Debug\Project1.exe
File: C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.21.27702\include\vector
Line: 1469

Expression: vector subscript out of range

For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.

(Press Retry to debug the application)

중단(A) 다시 시도(R) 무시(I)

벡터 (벡터 순회)

- 그렇다면 벡터의 내용물 전체를 순회하기 위해서는 어떻게 해야 할까?
- 벡터의 size와 [] 연산자를 가지고 for문을 통해 전체를 순회하며 출력할 수 있다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     const int vec_size = 5;
7     vector<char> vec{ 'c','h','a','r','!' };
8     for (int i = 0; i < vec_size; i++) {
9         cout << vec[i];
10    }
11    cout << endl;
12 }
13
14
```

Microsoft Visual Studio 디버그 콘솔

char!

C:\Users\blue\Desktop\방학프로

벡터 (벡터 순회)

- 하지만 저것보다 더 단순한 방법이 존재하는데 Python의 for문 사용법과 비슷하게 (순회하는원소의타입 [auto로 대체 가능] 원소변수명 : 벡터변수명)처럼 적어주면 for문 내에서 원소변수명에 벡터의 원소가 하나씩 들어가 접근 가능하다. 단 벡터의 값에 for문을 통해 변화를 주기 위해서는 &을 붙여 pass by reference 방식으로 가져와야 한다.

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<char> vec{'c','h','a','r','!'};
7     for (char elem : vec) {
8         cout << ++elem;
9     }
10    cout << endl;
11
12    for (auto elem : vec) {
13        cout << elem;
14    }
15    cout << endl;
16 }
17
```

Microsoft Visual Studio

dibs"
char!

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     vector<char> vec{'c','h','a','r','!'};
7     for (char& elem : vec) {
8         cout << ++elem;
9     }
10    cout << endl;
11
12    for (auto elem : vec) {
13        cout << elem;
14    }
15    cout << endl;
16 }
17
```

Microsoft Visual Studio

dibs"
dibs"

벡터 (메소드 or 멤버함수)

- 벡터도 객체이다. 그러므로 메소드를 가진다. 더 많은 메소드들은 <https://hyeonstorage.tistory.com/324> 참조

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6      vector<double> vec;
7      vec.push_back(10.0); // 벡터 맨 뒤에 파라미터로 들어간 요소 추가
8      vec.push_back(20.0);
9      vec.push_back(30.0);
10     vec.pop_back(); // 벡터의 마지막 요소를 벡터에서 제거한다.
11     vec.at(0); // == v[0]
12     cout << vec.size() << endl; // 벡터의 현재 길이를 반환한다.
13     cout << vec.empty() << endl; // 벡터가 비었는지의 여부 반환
14     vec.clear(); // 벡터 초기화
15     cout << vec.empty() << endl;
16 }
17
18
```

Microsoft Visual Studio 디버그 콘솔

벡터 (pass by value 예시)

- 하지만 pass by value 방식으로 함수의 파라미터를 전달하는 것은 객체의 모든 값을 복사해서 전달해주게 된다.
- 그러므로 벡터의 규모가 엄청나게 크다면 동작 시간이 오래 걸리게 된다.
- 이를 방지하기 위해 보통 벡터를 함수의 파라미터로 전달해줄 때는 주소 값을 전달해준다.



```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 void print(vector<int> v) {
6     for (int elem : v)
7         cout << elem << " ";
8     cout << '\n';
9 }
10
11 int sum(vector<int> v) {
12     int result = 0;
13     for (int elem : v)
14         result = elem;
15     return result;
16 }
17
18 int main() {
19     vector<int> list{ 2,4,6,8 };
20     print(list);
21     cout << sum(list) << '\n';
22     int n = list.size();
23     for (int i = 0; i < n; i++)
24         list[i] = 0;
25     print(list);
26     cout << sum(list) << '\n';
27 }
```

벡터 (pass by reference 예시)

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath> //수학관련 라이브러리
4  //sqrt -> 제곱근 반환 함수 사용 위해
5  using namespace std;
6
7  void print(const vector<int> &v) { // const를 붙여
8      // 수정이 없음을 명시해줌
9      for (int elem : v)
10         cout << elem << " ";
11     cout << '\n';
12 }
13
14 bool is_prime(int n) {
15     if (n < 2)
16         return false;
17     else {
18         bool result = true;
19         double r = n, root = sqrt(r);
20         for (int trial_factor = 2; result && trial_factor <= root;
21             trial_factor++)
22             result = (n % trial_factor != 0);
23         return result;
24     }
25 }
26
27 vector<int> primes(int begin, int end) {
28     vector<int> result;
29     for (int i = begin; i <= end; i++)
30         if (is_prime(i))
31             result.push_back(i);
32     return result;
33 }
34
35 int main() {
36     int low, high;
37     cout << "범위에서 가장 작은 값과 큰 값을 입력해주세요 : ";
38     cin >> low >> high;
39     vector<int> prime_list = primes(low, high);
40     print(prime_list);
41 }
```

벡터 (2차원 벡터)

- vector 객체의 요소로 vector 객체를 받아서 2차원 벡터를 만들 수 있다.
- 사용법은 다음과 같다.

`vector<vector<타입>>객체명(row, vector<타입>(col));` //row와 col은 각각 행과 열이다.

- 어렵게 보일 수 있는데 기존의 vector <타입> 객체명 (숫자, 초기화타입)에서 타입과 초기화 타입을 vector로 바꿔준 것 뿐이다.

예시로 `vector<vector<int>> a(2, vector<int>(3));`

		row(행)			-> 벡터 하나
		0	1	2	
col (열)	0	a[0][0]	a[1][0]	a[2][0]	
	1	a[0][1]	a[1][1]	a[2][1]	

벡터 (2차원 벡터 예시)

- 2차원 이상의 3차원 4차원 벡터도 응용하면 가능하다!

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void print(const vector<vector<double>>& m) {
6      for (int row = 0; row < m.size(); row++) {
7          for (int col = 0; col < m[row].size(); col++)
8              cout << m[row][col] << ' ';
9          cout << '\n';
10     }
11 }
12 //단순하게는
13 void print(const vector<vector<double>>& m) {
14     for (auto row : m) {
15         for (double elem : row)
16             cout << elem << ' ';
17         cout << '\n';
18     }
19 }
20
21
```

과제 5

- 랩 5 ㄱ

