Topic 2 Types
Part 2

내용

- 배열
- std::array, std::vector, std::string 클래스
- iterator와 알고리즘 함수
- 나열형(enum)
- 구조체(struct)
- 공용체(union)
- typedef

배열

- 동일 타입의 변수 집합
- 정의 방법: T 변수명[n]

```
int values[5] = {0, 10, 20, 30, 40} ;
```

values

values[0]	values[1]	values[2]	values[3]	values[4]
0	10	20	30	40

배열의 기본 개념

```
#include <iostream>
using namespace std ;
const int SIZE = 3 ;
int main() {
  // 배열 정의와 초기화
  int intArray[SIZE] = {10, 20, 30} ;
  int sum = 0 ;
  for ( int i = 0 ; i < SIZE ; i ++ ) {
    sum += intArray[i] ; // 원소의 접근
  }
  cout << sum << endl ;
}
```

1차원 배열의 원소 접근 방법

- 배열의 변수 이름은 첫번째 원소에 대한 포인터로 간주

인덱스를 이용한 접근	포인터를 이용한 접근
values[0]	*values
values[1]	*(values+1)
values[2]	*(values+2)
…	…
values[i]	*(values+i)

배열의 초기화

- 배열의 원소 값은 배열을 정의할 때 초기화될 수 있다

```
int intArray1[3] = {1, 2, 3} ;
```

```
int intArray1[3]{1, 2, 3} ; // C++11
```

- 배열의 크기는 초기화 목록의 크기로 결정된다

```
int intArray2[] = {1, 2, 3, 4} ;
// int intArray2[4] = {1, 2, 3, 4}와 동일
```

- 지정되지 않은 원소의 초기값은 0으로 결정

```
int intArray3[3] ={1, 2} ;
// int intArray3[3] = {1, 2, 0}과 동일
```

배열의 초기화

- 문자의 배열은 문자열 리터럴을 이용하여 초기화

```
char strArray1[] = “a string“ ;
// char strArray1[] = {'a', ' ', 's', 't', 'r', 'i', 'n', 'g', 0} ; 과 동일
```

- 클래스 객체의 배열도 가능하다

```
string names[5] = {string(“Kim”), string(“Park”)} ;
```

포인터의 배열

```
#include <iostream>
using namespace std ;

const int SIZE = 3 ;
int main() {
  int intArrayA[SIZE] ;
  for ( int i = 0 ; i < SIZE ; i ++ )
  {
    cin >> intArrayA[i] ;
    intArrayA[i] += 10 ;
  }
```

```
  // 포인터의 배열 정의
  int* pIntArrayB[SIZE] ;
  int intVal1, intVal2, intVal3 ;
  cin >> intVal1 >> intVal2 >> intVal3 ;

  pIntArrayB[0] = &intVal1 ;
  pIntArrayB[1] = &intVal2 ;
  pIntArrayB[2] = &intVal3 ;
  for ( int i = 0 ; i < SIZE ; i ++ ) {
    const int a = intArrayA[i] ;
    const int b = *pIntArrayB[i] ;
    const int sum = a + b ;
    cout << a << " + " << b <<
      " = " << sum << endl ;
  }
}
```

Good Design: 올바른 인덱스의 사용

- 크기가 N인 배열의 인덱스: 0 .. N-1

올바른 인덱스 사용 예	부정확한 인덱스 사용 예
for (int i = 0 ; i < N ; i ++)	for (int i = 0 ; i <= N ; i ++)

2차원 배열

- 2차원 배열: 다른 배열을 원소로 가짐
- 정의 방법: T 변수명[행수][열수]
- 예) int values[3][5]; //선형 메모리 구조를 표로 추상화

	1열	2열	3열	4열	5열
1행	values[0,0]	values[0,1]	values[0,2]	values[0,3]	values[0,4]
2행	values[1,0]	values[1,1]	**values[1,2]**	values[1,3]	values[1,4]
3행	values[2,0]	values[2,1]	values[2,2]	values[2,3]	values[2,4]

```
#include <iostream>
using namespace std ;
const int ROW_SIZE = 3 ;
const int COLUMN_SIZE = 5 ;
int main() {
  int intArrayA[ROW_SIZE][COLUMN_SIZE] ;
  for ( int i = 0 ; i < ROW_SIZE ; i ++ )
    for ( int j = 0 ; j < COLUMN_SIZE ; j ++ )
      intArrayA[i][j] = i + j ;

  int intArrayB[ROW_SIZE][COLUMN_SIZE] ;
  for ( int i = 0 ; i < ROW_SIZE ; i ++ )
    for ( int j = 0 ; j < COLUMN_SIZE ; j ++ )
      intArrayB[i][j] = i * j ;

  cout << "행렬 A + B는" << endl ;
  for ( int i = 0 ; i < ROW_SIZE ; i ++ ) {
    for ( int j = 0 ; j < COLUMN_SIZE ; j ++ ) {
      const int a = intArrayA[i][j] ;
      const int b = intArrayB[i][j] ;
      const int sum = a + b ;
      cout << a << " + " << b << " = " << sum << '₩t' ;
    }
    cout << endl ;
  }
}
```

```
행렬 A + B는
0 + 0 = 0    1 + 0 = 1    2 + 0 = 2    3 + 0 = 3    4 + 0 = 4
1 + 0 = 1    2 + 1 = 3    3 + 2 = 5    4 + 3 = 7    5 + 4 = 9
2 + 0 = 2    3 + 2 = 5    4 + 4 = 8    5 + 6 = 11    6 + 8 = 14
```

2차원 배열의 원소 접근 방법

인덱스를 이용한 접근	포인터를 이용한 접근
values[0][0]	*values
values[0][1]	*(values+1)
values[i][0]	*(values + (5 * i) + 0)
values[i][j]	*(values + (5 * i) + j)
…	…

values[0][0]	Values[0][1]	Values[0][2]	Values[0][3]	Values[0][4]	**values[1][0]**	Values[1][1]	Values[1][2]	Values[1][3]	Values[1][4]	**values[2][0]**	Values[2][1]	Values[2][2]	Values[2][3]	Values[2][4
0	1	2	3	4	1	3	5	7	9	2	5	8	11	14

동적 1차원 배열

- new와 delete를 이용하여 프로그램 실행 중에 크기를 결정할 수 있는 동적 배열을 정의

```
#include <iostream>
using namespace std ;
int main()
    cout << "1차원 배열의 크기를 입력하시오." << endl ;
    int intSize ;
    cin >> intSize ;
    // new로 입력된 크기만큼의 int를 할당
    int* const intArray = new int[intSize] ;
    for ( int i = 0 ; i < intSize ; i ++ )
        cin >> intArray[i] ;
    delete [] intArray ; // delete로 할당된 메모리 반환
}
```

포인터의 동적 배열

```
#include <iostream>
using namespace std ;

int main() {
  int intArray1[2] = {0, 1} ;
  int intArray2[2] = {2, 3} ;
  int intArray3[2] = {4, 5} ;
   // 3개의 int* 즉 int 배열을 원소로 하는 배열
  int** const ppIntArray = new int*[3] ;
  ppIntArray[0] = intArray1 ;       // 배열의 이름은 포인터로서 사용됨
  ppIntArray[1] = intArray2 ;
  ppIntArray[2] = intArray3 ;

  for ( int i = 0 ; i < 3 ; i ++ ) {
    for ( int j = 0 ; j < 2 ; j ++ )
      cout << ppIntArray[i][j] << '₩t' ;
    cout << endl ;
  }
  delete [] ppIntArray ;
}
```

동적 2차원 배열

```
#include <iostream>
using namespace std ;
int main() {
   cout << "행렬의크기를입력하시오." << endl ;
   int rowSize, columnSize ;
   cin >> rowSize >> columnSize ;

   cout << "행렬A [" << rowSize << " X " << columnSize << "] 의 값을 입력하시오.₩n" ;
   int ** const intArrayA = new int*[rowSize] ;
   for ( int i = 0 ; i < rowSize ; i ++ ) {
      intArrayA[i] = new int[columnSize] ;
      for ( int j = 0 ; j < columnSize ; j ++ ) cin >> intArrayA[i][j] ;
   }
   cout << endl ;
   cout << "행렬B [" << rowSize << " X " << columnSize << "] 의 값을 입력하시오.₩n" ;
   int ** const intArrayB = new int*[rowSize] ;
   for ( int i = 0 ; i < rowSize ; i ++ ) {
      intArrayB[i] = new int[columnSize] ;
      for ( int j = 0 ; j < columnSize ; j ++ ) cin >> intArrayB[i][j] ;
   }
```

동적 2차원 배열

```
  cout << endl << "행렬A + B는" << endl ;
  for ( int i = 0 ; i < rowSize ; i ++ ) {
    for ( int j = 0 ; j < columnSize ; j ++ ) {
      const int a = intArrayA[i][j] ;
      const int b = intArrayB[i][j] ;
      cout << a << " + " << b << " = " << a+b<< '₩t' ;
    }
    cout << endl ;
  }

  // intArrayA, intArrayB를 delete해 주어야 함
  for ( int i = 0 ; i < rowSize ; i ++ ) {
    delete [] intArrayA[i] ;
    delete [] intArrayB[i] ;
  }

  delete [] intArrayA ;
  delete [] intArrayB ;
}
```

$$\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}+\begin{bmatrix}2 & 2\\ 2 & 2\end{bmatrix}$$