



C++ 프로그래밍

김 형 기

hk.kim@jbnu.ac.kr

Today

- 배열
- 명령문과 연산자
- 제어문

배열

Array

- 배열

- 개요
- 선언과 초기화
- 접근

Array

- 배열
 - 복합(compound) 데이터 타입
 - **동일한 타입**인 요소들의 집합
- 개별 요소들에 직접적인 접근이 가능
- 사용 목적
 - 아래와 같은 코드를 효율적으로 바꿀 수 있음

```
1 int score_1 = 0;  
2 int score_2 = 0;  
3 int score_3 = 0;  
4 ...  
5 int score_100 0;
```

● 특징

- 고정된(정해진) 길이
- 연속된 메모리 주소에 저장
- 인덱스를 통해 접근 가능
- 인덱스는 0부터 시작하며, 마지막 인덱스는 size-1
- out of bound 체크 하지 않음
- 초기화 필요
- 효율적인 데이터 구조

scores	index
100	[0]
85	[1]
21	[2]
56	[3]
70	[4]
95	[5]

길이가 6인 배열

Arrays

- 직접 확인해보세요!

```
1 int scores[6] = { 100, 85, 21, 56, 70, 95 };
```

메모리 1		
주소: 0x005DF73D		
0x005DF73D	cc	?
0x005DF73E	cc	?
0x005DF73F	cc	?
0x005DF740	64	d
0x005DF741	00	.
0x005DF742	00	.
0x005DF743	00	.
0x005DF744	55	U
0x005DF745	00	.
0x005DF746	00	.
0x005DF747	00	.
0x005DF748	15	.
0x005DF749	00	.
0x005DF74A	00	.
0x005DF74B	00	.
0x005DF74C	38	8
0x005DF74D	00	.
0x005DF74E	00	.
0x005DF74F	00	.
0x005DF750	46	F
0x005DF751	00	.
0x005DF752	00	.
0x005DF753	00	.
0x005DF754	5f	-
0x005DF755	00	.
0x005DF756	00	.
0x005DF757	00	.
0x005DF758	cc	?

0x 00 00 00 64

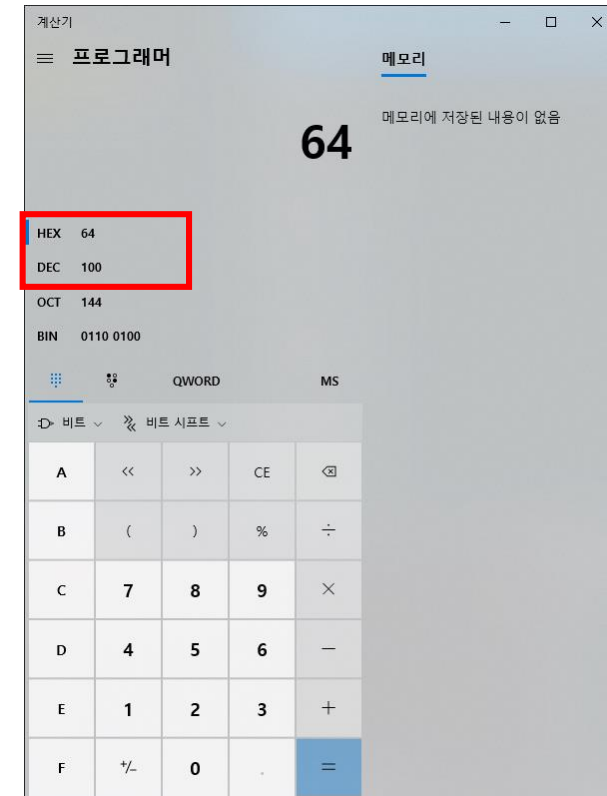
0x 00 00 00 55

0x 00 00 00 15

0x 00 00 00 38

0x 00 00 00 46

연속된(붙어있는 층)
메모리에 값이 쓰여있음



쓰여있는 값이 맞는지
계산기로 확인 가능

Defining Arrays

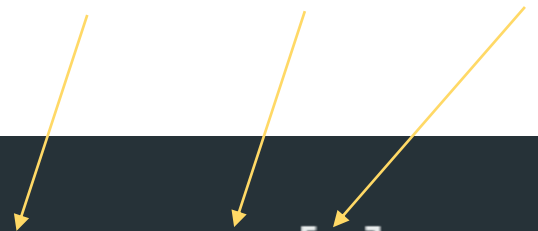
- 배열의 정의

- 기본 형

요소타입 배열이름 [요소개수];

- 정의 예

```
1 int scores[5];  
2  
3 const int daysInYear = 365;  
4 double temperature[daysInYear];
```



Initializing Arrays

- 배열의 초기화

```
1 int scores[5] = {100, 85, 21, 56, 70};  
2  
3 int highScores[10] = {3, 5}; //3,5 and remaining 0  
4  
5 const int daysInYear = 365;  
6 double temperature[daysInYear] = {0}; //all to zero  
7  
8 int myArray[] = {1, 2, 3, 4, 5}; //automatic sizing
```

Accessing Array Elements

- 배열 내의 요소들을 사용(= 읽고 쓰기)

```
1 int scores[] = {100, 85, 21, 56, 70};  
2  
3 cout << "first score : " << scores[0] << endl;  
4 cout << "second score : " << scores[1] << endl;  
5 cout << "third score : " << scores[2] << endl;  
6 cout << "fourth score : " << scores[3] << endl;  
7 cout << "fifth score : " << scores[4] << endl;
```

- 요소들은 개별적인 주소를 갖고 있으므로 변수와 동일하게 값 수정 가능

```
1 cin >> scores[0];  
2  
3 scores[1] = 40;
```

(Summary) Arrays

- 배열 (동일한 타입 요소들의 집합, 고정된 길이)
 - 개요
 - 정의와 초기화 – (고정 크기 선언, 리스트 초기화 "{ }")
 - 사용 – (인덱스로 개별 요소에 접근하여 사용)

명령문과 연산자

Statements and Operators

- 표현식
- 명령문
- 연산자
 - 대입 연산자
 - 산술 연산자
 - 증감 연산자
 - 관계 연산자
 - 논리 연산자
 - 멤버 접근 연산자
 - 기타

Expressions

- 표현식

- 코드의 가장 작은 구성요소

- Ex, 연산을 통해 값을 계산

```
1 34 // literal
2 favorite_numver //variable
3 3+5 // addition
4 3*5 // multiplication
5 a>b // relational
6 a = b // assignment
```

Statements

- 명령문

- 명령을 수행하는 코드 단위
- 세미콜론(";") 으로 끝나는 문장
- 표현식의 집합

```
1 int favorite_number; //declaration
2 favorite_number = 20; // assignment
3 3 + 5; // expression
4 favorite_number = 3 * 5; // assignment
5 if (a>b) cout << "a is greater than b"; //if
```

Operations (1)

- 연산자

- 단항(unary), 이항(binary), 삼항(ternary) 연산자

- 대입 연산자

lhs = rhs

- l-value & r-value
- 오른쪽의 값을 계산하여 왼쪽에 대입
- 컴파일러가 대입이 가능한지 체크함

```
1 int num1=0;  
2 num1 = "Hyunki"; // ??
```

- 왼쪽은 대입이 가능해야 함(ex, 리터럴, 상수는 될 수 없음)

Operations (2)

- 산술 연산자

- `+, -, *, /, %(mod)`

- 증감 연산자

- Prefix (`++num, --num`) (대입 전 증감)
- Postfix (`num++, num--`) (대입 후 증감)

- 비교 연산자

- `==, !=`
- 결과는 Boolean 타입의 `true` or `false`

Operations (3)

- 관계 연산자

- $<$, $>$, $<=$, $>=$

- 논리 연산자

- $!$, $\&\&$, $\|\$ (not, and, or)
- Short-circuit evaluation

➤ 결과 파악이 가능한 경우 나머지 연산을 하지 않음 (최적화)

```
1 if(expr1 && expr2 && expr3) // if expr1 is false?  
2 if(expr1 || expr2 || expr3) // if expr1 is true?
```

Operations (4)

● 복합 연산자

```
1 a += 1; // a = a+1;  
2 a *= b+c; // a = a*(b+c);
```

연산자	의미
=	첫 번째 피연산자에서 지정한 개체에 두 번째 피연산자의 값을 저장합니다(단순 할당).
*=	첫 번째 피연산자의 값과 두 번째 피연산자의 값을 곱하여 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
/=	첫 번째 피연산자의 값을 두 번째 피연산자의 값으로 나누어 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
%=	두 번째 피연산자의 값에서 지정한 첫 번째 피연산자의 모듈러스를 가져와서 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
+=	두 번째 연산자의 값과 첫 번째 연산자의 값을 더하여 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
-=	첫 번째 피연산자의 값에서 두 번째 피연산자의 값을 빼서 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
<<=	두 번째 피연산자의 값에서 지정한 비트 수만큼 첫 번째 피연산자의 값을 왼쪽으로 이동하여 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
>>=	두 번째 피연산자의 값에서 지정한 비트 수만큼 첫 번째 피연산자의 값을 오른쪽으로 이동하여 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.
&=	첫 번째 및 두 번째 피연산자의 비트 AND를 구하여 첫 번째 피연산자가 지정한 개체에 결과를 저장합니다.

Operations (5)

● 연산자 우선순위

- 혼동을 없애기 위해 우선순위 괄호표기 하는 것이 좋은 코드!

순위	연산자	결합순서
1	() [] -> .	왼쪽 우선
2	! ~ ++ -- + -(부호) *(포인터) & sizeof 캐스트	오른쪽 우선
3	*(곱셈) / %	왼쪽 우선
4	+ -(덧셈, 뺄셈)	왼쪽 우선
5	<< >>	왼쪽 우선
6	< <= > >=	왼쪽 우선
7	== !=	왼쪽 우선
8	&	왼쪽 우선
9	^	왼쪽 우선
10		왼쪽 우선
11	&&	왼쪽 우선
12		왼쪽 우선
13	? :	오른쪽 우선
14	= 복합대입	오른쪽 우선
15	,	왼쪽 우선

(Summary) Statements and Operators

- 표현식 (가장 작은 단위, 리터럴, 변수 등등)
- 명령문 (코드의 구성 단위, 세미콜론으로 끝을 표기)
- 연산자
 - 대입 연산자 (l-value = r-value)
 - 산술 연산자 (사칙연산)
 - 증감 연산자 (++ , --)
 - 관계 연산자 (> , < , >= , <=)
 - 논리 연산자 (! , && , ||)
 - 멤버 접근 연산자 (., ->)
 - 기타

제어문

Controlling flow

- 조건문

- If-else문 (블록, nested)
- Switch문
- ?: 연산자

- 반복문

- For문
- While, do-while문
- Continue, break

Controlling flow

- 조건문

- If-else문 (블록, nested)
- Switch문
- ?: 연산자

- 반복문

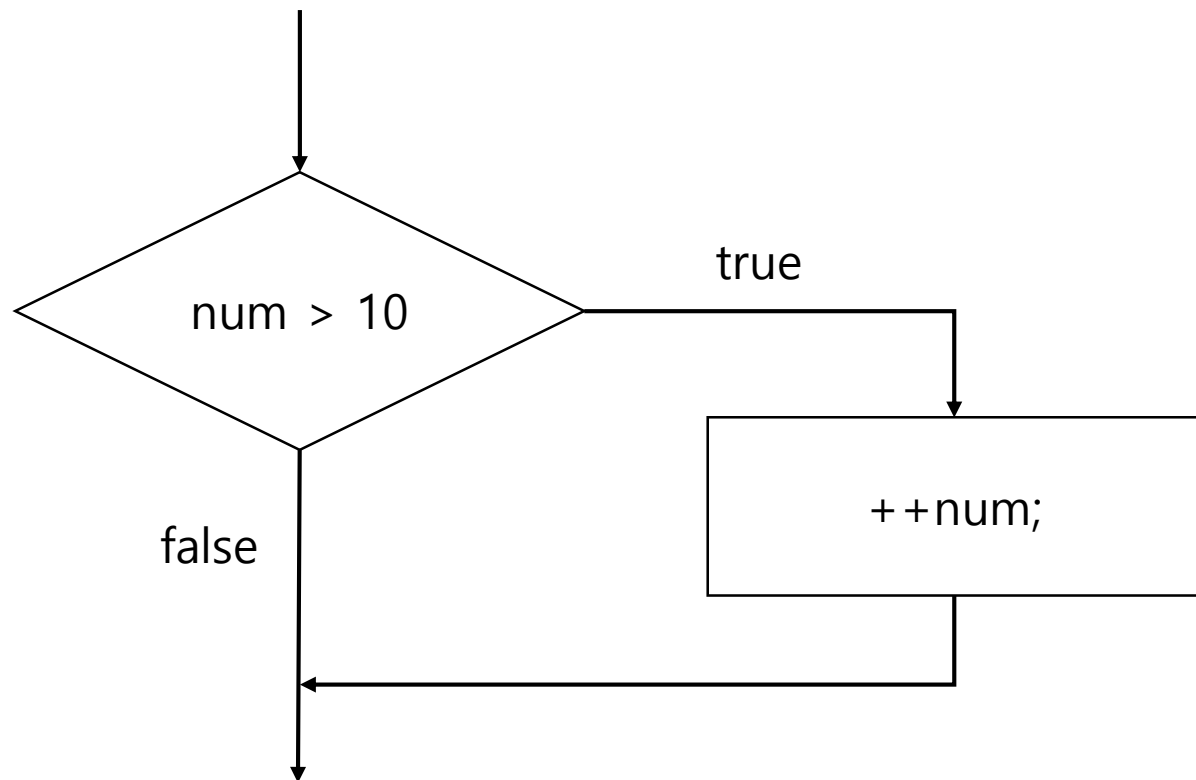
- For문
- While, do-while문
- Continue, break

Conditional statement

- if문

- 표현식이 참일 경우에만 실행하는 명령문

```
1 if (Expr)
2     statement;
3
4 if(num > 10)
5     ++num;
```



Conditional statement

- if문과 블록

- 조건문 내에서 하나 이상의 명령문을 실행하기 위해서는 블록 내에 명령문을 작성
- 블록은 "{"로 시작해서 "}"로 끝남
- 블록 내에서 선언된 변수는 "**지역변수**"라 하며, 블록 내에서만 접근 가능

```
1 if(expr)
2 {
3     int a = 10; //local variable
4 }
5
6 cout << a << endl; //ERROR!!
```

```
1 {
2     int a = 10; //local variable
3 }
4
5 cout << a << endl; //ERROR!!
```

꼭 if에만 해당되는 이야기가 아님! 블록 안에서
정의한 변수는 블록 안에서만 사용 가능

Conditional statement

- if – else 문

- 표현문의 참/거짓 여부에 따라 명령문 실행 분기
- else if 키워드를 통해 다양한 조건을 기술 가능

```
1 if(expr)
2 {
3     statement1;
4 }
5 else
6 {
7     statement2;
8 }
```

```
1 if (score > 90)
2 {
3     cout << "A";
4 }
5 else if (score > 80)
6 {
7     cout << "B";
8 }
9 else if (score > 70)
10 {
11     cout << "C";
12 }
13 else
14 {
15     cout << "F";
16 }
```

Conditional statement

- nested if문

- 블록과 if-else문을 중첩하여 복잡한 조건을 효율적으로 기술 가능

```
1 if (my_score ≠ your_score)
2 {
3     if(my_score > your_score)
4     {
5         cout << "I win!";
6     }
7     else
8     {
9         cout << "You win!";
10    }
11 }
12 else
13 {
14     cout << "Tie!";
15 }
```

← "!=" 로 작성

Conditional statement

- switch문

- switch, case, default를 사용한 분기문
- Switch 표현문의 결과는 정수형 리터럴이어야 함(정수, char, 열거형)

```
1 switch(selection)
2 {
3     case 1: cout << "1 selected";
4         break;
5     case 2: cout << "2 selected";
6         break;
7     case 3:
8     case 4: cout << "3 or 4 selected";
9         break;
10    default: cout << "Not 1,2,3,4";
11 }
```

```
1 switch (selection)
2 {
3     case 1: cout << "1 selected";
4     case 2: cout << "2 selected";
5     case 3: cout << "3 selected";
6     case 4: cout << "4 selected";
7         break;
8     default: cout << "Not 1,2,3,4";
9 }
```

Conditional statement

- ?: 연산자

```
(conditional_expr) ? expr1 : expr2
```

- conditional_expr은 boolean 표현식
 - 표현식이 참이라면 expr1의 값을 리턴
 - 표현식이 거짓이라면 expr2의 값을 리턴
- If-else 문의 사용과 유사함
- 삼항 연산자

Conditional statement

- ?: 연산자 예시

```
1 int a=10, b=20;  
2 int score=92;  
3 int result{};  
4  
5 result = (a>b) ? a:b;  
6 result = (a<b) ? (b-a):(a-b);  
7 result = (b!=0) ? (a/b):0;  
8 cout << ((score>90) ? "Excellent!":"Good!");
```

Controlling flow

- 조건문

- If-else문 (블록, nested)
- Switch문
- ?: 연산자

- 반복문

- For문
- While, do-while문
- Continue, break

Looping statement

- 반복문(iteration / repetition)
 - 반복 조건 + 명령문
- 사용 예
 - 특정 횟수만큼 반복이 필요할 때
 - 집합의 각 요소에 대한 연산이 필요할 때
 - 특정 조건이 참일 동안 명령의 수행이 필요할 때
 - 입력 스트림의 끝까지 반복을 수행해야 할 때
 - 무한 반복이 필요할 때
 - 등등...

Looping Statement

- for문

```
1 for(초기화 ; 종료조건 ; 증감)
2 {
3     명령문;
4 }
```

```
1 int i=0;
2
3 for(i=0;i<5;++i)
4 {
5     cout << i << endl;
6 }
```

```
1 for(int i=0;i<5;i++)
2 {
3     cout << i << endl;
4 }
5
6 i=10; // ERROR!
```

Looping Statement

- for문

- 배열 루프

```
1 int scores[] = {100,90,50};  
2  
3 for(int i=0;i<3;i++)  
4 {  
5     cout << scores[i] << endl;  
6 }
```

- 콤마 연산자

```
1 for(int i=0, j=5;i<5; i++,j++)  
2 {  
3     cout << i << " * " << j << " : " << i*j << endl;  
4 }
```

Looping Statement

- for문, 모든 조건이 항상 존재해야 할 필요는 없음

```
1 int main()
2 {
3     int i = 0;
4     for (; true; )
5     {
6         i++;
7         if (i ≤ 5)
8         {
9             cout << i << endl;
10        }
11        else
12        {
13            break;
14        }
15    }
16 }
```

← "≤" 로 작성

Looping Statement

- while문

```
1 while(expr)
2 {
3     statements;
4 }
```

```
1 int i=0;
2
3 while(i<5)
4 {
5     cout << i << endl;
6     i++; //Important!
7 }
```

```
1 bool is_done=false;
2 int number=0;
3
4 while(!is_done)
5 {
6     cout << "enter number under 10" << endl;
7     cin >> number;
8     if(number ≥ 10)
9     {
10         cout << "wrong number" << endl;
11     }
12     else
13     {
14         cout << "OK!" << endl;
15         is_done = true;
16     }
17 }
```

Looping Statement

- do while문

```
1 do
2 {
3     statements;
4 } while(expr);
```

```
1 int number;
2
3 do
4 {
5     //int number{};
6     cout << "enter number under 10" << endl;
7     cin >> number;
8 } while(number ≤ 10);
9
10 cout << "OK!" << endl;
```

Looping Statement

- continue

- Continue문 이후의 문장은 실행되지 않음
- 다음 iteration으로 곧바로 넘어가기 위해 사용

- break

- Break문 이후의 문장은 실행되지 않음
- 루프 밖으로 바로 빠져나가기 위해 사용

```
1 int values[] = { 1,2,-1,3,-1,-99,7,8,9 };
2
3 for (int i = 0; i < 9; i++)
4 {
5     if (values[i] == -99)
6         break;
7     else if (values[i] == -1)
8         continue;
9     else
10         cout << values[i] << endl;
11 }
```

- continue

- Continue문 이후의 문장은 실행되지 않음
- 다음 iteration으로 곧바로 넘어가기 위해 사용

- break

- Break문 이후의 문장은 실행되지 않음
- 루프 밖으로 바로 빠져나가기 위해 사용

```
1 int values[] = { 1,2,-1,3,-1,-99,7,8,9 };
2
3 for (int i = 0; i < 9; i++)
4 {
5     if (values[i] == -99)
6         break;
7     else if (values[i] == -1)
8         continue;
9     else
10         cout << values[i] << endl;
11 }
```


(Summary) Controlling flow

- 조건문

- If-else문 (블록, nested)
- Switch문
- ?: 연산자

- 반복문

- For문
- While, do-while문
- Continue, break