# Chapter 1. C++ 기초

박 종 혁 교수
UCS Lab
(http://www.parkjonghyuk.net)

Tel: 970-6702
Email: jhpark1@snut.ac.kr

# Learning Objectives

- C++ 소개
  - 기원, 객체지향 프로그래밍, 용어
- 변수, 식 및 할당문
- 콘솔 입/출력
- 프로그램 스타일
- 라이브러리 및 네임스페이스

# C++ 소개

- C++ 언어의 기원
  - 저급 언어
    - 기계어, 어셈블리어
  - 고급 언어
    - C, C++, ADA, COBOL, FORTRAN
  - C++ 언어를 이용한 객체지향 프로그래밍 (Object Oriented Programming)
- C++ 용어
  - 프로그램(*Programs*) 과 함수(*functions*)
  - cin 과 cout을 이용한 기본 입력/출력 (I/O)

# 디스플레이 1.1
# C++ 프로그램 예제 (1 of 2)

**Display 1.1 A Sample C++ Program**

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

3   int main( )
4   {
5       int numberOfLanguages;

6       cout << "Hello reader.\n"
7            << "Welcome to C++.\n";

8       cout << "How many programming languages have you used? ";
9       cin >> numberOfLanguages;

10      if (numberOfLanguages < 1)
11          cout << "Read the preface. You may prefer\n"
12               << "a more elementary book by the same author.\n";
13      else
14          cout << "Enjoy the book.\n";

15      return 0;
16  }
```

# 디스플레이 1.1
## C++ 프로그램 예제 (2 of 2)

**SAMPLE DIALOGUE 1**

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 0
Read the preface. You may prefer
a more elementary book by the same author.
```

*User types in 0 on the keyboard.*

**SAMPLE DIALOGUE 2**

```
Hello reader.
Welcome to C++.
How many programming languages have you used? 1
Enjoy the book
```

*User types in 1 on the keyboard.*

# C++ 변수

- C++ 식별자
  - 키워드/예약어 vs. 식별자
  - 식별자의 대소문자 구분, 확실한 분별이 가능하게 사용
  - 의미있는 이름을 사용!
- 변수
  - 프로그램을 위한 데이터가 저장된 메모리 위치
  - 프로그램의 모든 변수는 사용되기 전 반드시 선언되어야 함

# 디스플레이 1.2
# 간단한 데이터형 (1 of 2)

Display 1.2 **Simple Types**

| TYPE NAME | MEMORY USED | SIZE RANGE | PRECISION |
|---|---|---|---|
| `short` (also called `short int`) | 2 bytes | −32,768 to 32,767 | Not applicable |
| `int` | 4 bytes | −2,147,483,648 to 2,147,483,647 | Not applicable |
| `long` (also called `long int`) | 4 bytes | −2,147,483,648 to 2,147,483,647 | Not applicable |
| `float` | 4 bytes | approximately $10^{-38}$ to $10^{38}$ | 7 digits |
| `double` | 8 bytes | approximately $10^{-308}$ to $10^{308}$ | 15 digits |

# 디스플레이 1.2
# 간단한 데이터형 (2 of 2)

| | | | |
|---|---|---|---|
| `long double` | 10 bytes | approximately $10^{-4932}$ to $10^{4932}$ | 19 digits |
| `char` | 1 byte | All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.) | Not applicable |
| `bool` | 1 byte | `true`, `false` | Not applicable |

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

# 데이터 할당

- 변수의 선언과 동시에 초기화
  - 초기화하지 않은 변수에는 “정의되지 않은” 값 (쓰레기값)이 들어가 있다
    - int myValue = 0;
- 실행 중 데이터 할당
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues는 반드시 변수이어야 함
    - Rvalues는 어떠한 식도 가능
    - 예:
      distance = rate * time;
      Lvalue: "distance"
      Rvalue: "rate * time"

# 데이터 할당 : 축약 표기

- 디스플레이, 14페이지

| EXAMPLE | EQUIVALENT TO |
|---|---|
| `count += 2;` | `count = count + 2;` |
| `total -= discount;` | `total = total - discount;` |
| `bonus *= 2;` | `bonus = bonus * 2;` |
| `time /= rushFactor;` | `time = time/rushFactor;` |
| `change %= 100;` | `change = change % 100;` |
| `amount *= cnt1 + cnt2;` | `amount = amount * (cnt1 + cnt2);` |

# 데이터 할당 규칙

- 데이터 할당의 호환성
  - 형 불일치
    - 일반 규칙 : 하나의 형의 값은 다른 형의 값으로 저장할 수 없다
  - intVar = 2.99;  // intVar에 2가 할당됨!
    - 정수 부분만 “일치”, 따라서 정수부만 할당됨
    - “묵시적(implicit) 형 변환” 또는 “자동 형 변환” 이라 한다
  - 문자상수(리터럴:Literals)
    - 2, 5.75, "Z", "Hello World"
    - “상수(constants)”로 취급 : 프로그램에서 변경 불가

# 문자상수(Literal) 데이터

- 문자상수(literal)
  - 예:
    - 2 // 문자 상수 int
    - 5.75 // 문자 상수 double
    - "Z" // 문자 상수 char
    - "Hello World" // 문자 상수 string
- 실행 중 값의 변경이 불가
- "문자상수"라고 불리는 이유는 프로그램에서 "문자 그대로 타이핑 되기" 때문이다

# 백슬래시 문자상수 (Escape Sequences)

- "확장된" 문자 세트
- 백슬래쉬 "\" 가 문자 앞에 위치
  - 백슬래쉬 문자상수가 들어온다는 것을 컴파일러에게 알림
  - 뒤에 따라오는 문자는 "백슬래쉬 문자상수의 세트(문자)"로 인식됨
  - 디스플레이 1.3 (다음 슬라이드)

# 디스플레이 1.3
# 백슬래쉬 문자상수의 종류 (1 of 2)

**Display 1.3 Some Escape Sequences**

| SEQUENCE | MEANING |
|---|---|
| \n | New line |
| \r | Carriage return (Positions the cursor at the start of the current line. You are not likely to use this very much.) |
| \t | (Horizontal) Tab (Advances the cursor to the next tab stop.) |
| \a | Alert (Sounds the alert noise, typically a bell.) |
| \\ | Backslash (Allows you to place a backslash in a quoted expression.) |

# 디스플레이 1.3
## 백슬래쉬 문자상수의 종류 (2 of 2)

| | |
|---|---|
| \' | Single quote (Mostly used to place a single quote inside single quotes.) |
| \" | Double quote (Mostly used to place a double quote inside a quoted string.) |
| The following are not as commonly used, but we include them for completeness: | |
| \v | Vertical tab |
| \b | Backspace |
| \f | Form feed |
| \? | Question mark |

# 상수

- 상수 이름 짓기
  - 문자상수(Literal)가 상수이지만, 의미를 내포하지는 않는다
    - 예를 들어, 프로그램에서 보이는 24는 표현하고자 하는 것이 무엇인지 나타나지 않는다
- 명명된 상수의 사용
  - 데이터를 표현하는 의미있는 이름

    const int NUMBER_OF_STUDENTS = 24;
    - "선언된 상수" 또는 "명명된 상수"라고 한다
    - 이제, 프로그램에서 필요로 하는 곳에서 이름이 사용된다
    - 추가적인 이득 : 프로그램의 유지 보수 용이(한번의 수정으로 결과의 변경 가능 – 명명부만 고치면 됨)

# 산술 연산자:
# **디스플레이 1.4** 명명된 상수 (1 of 2)

- 기본 산술 연산자
  - 우선순위 규칙
  - 기본 규칙

**Display 1.4 Named Constant**

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6      const double RATE = 6.9;
7      double deposit;

8      cout << "Enter the amount of your deposit $";
9      cin >> deposit;
```

# 산술 연산자:
# 디스플레이 1.4 명명된 상수 (2 of 2)

```
10        double newBalance;
11        newBalance = deposit + deposit*(RATE/100);
12        cout << "In one year, that deposit will grow to\n"
13             << "$" << newBalance << " an amount worth waiting for.\n";

14        return 0;
15    }
```

**SAMPLE DIALOGUE**

Enter the amount of your deposit $**100**
In one year, that deposit will grow to
$106.9 an amount worth waiting for.

# 산술 정밀도

- 연산의 정밀도
  - **매우** 중요한 고려사항!
    - C++에서 수식은 예측한 대로 계산되지 않을 수도 있다!
  - "최상위 순위 피연산자"가 수행되는 연산의 "정밀도"를 결정
  - 일반적인 함정!

# 산술 정밀도의 예

- 예:
  - C++에서 17 / 5는 3으로 계산!
    - 두 개의 피연산자는 integer 형
    - Integer 형 정밀도로 나눗셈 연산 수행!
  - C++에서 17.0 / 5는 3.4와 같다!
    - 최상위 순위 피연산자는 “double 형”
    - double 형 정밀도로 나눗셈 연산 수행!
  - int intVar1 =1, intVar2=2;
    intVar1 / intVar2;
    - Integer 형 정밀도로 나눗셈 연산 수행!
    - 결과: 0!

# 부분적인 산술 정밀도

- 연산은 "하나씩" 수행된다
  - 1 / 2 / 3.0 / 4 는 3개의 부분 연산이 수행된다
    - 첫 번째 → 1 / 2  equals 0
    - 두 번째 → 0 / 3.0 equals 0.0
    - 세 번째 → 0.0 / 4 equals 0.0!
- 복잡한 식에서 하나의 피연산자를 수정하는 것으로는 충분하지 않다
  - 연산이 수행되는 동안 모든 부분 연산이 고려되어야 한다!

# 형 변환(Type Casting)

- 변수의 형 변환
  - 문자상수(literal)은 ".0"을 추가하여 정확한 연산을 강제할 수 있지만 변수는?
    - "myInt.0"을 사용할 수는 없다!
  - static_cast<double>intVar
  - 명시적으로 intVar를 double 형으로 변환 ("casts" 또는 "converts"
    - 변환의 결과는 다음에 사용된다
    - 예제 식:
      doubleVar = static_cast<double>intVar1 / intVar2;
      - 강제적인 변환으로 두 정수형 변수 사이에 double 형 정밀도의 나눗셈이 발생한다!

# 형 변환(Type Casting)

- 두 가지 형태
  - 묵시적(Implicit) 형 변환 — 또는 자동("Automatic")
    - 자동적으로 실행
      17 / 5.5
      이 식은 묵시적 형 변환이 발생하여 17 → 17.0 으로 변환
  - 명시적(Explicit) 형 변환
    - 프로그래머가 변환 연산자를 이용하여 변환을 명기
      (double)17 / 5.5
      위의 식과 같은 식이지만, 명시적 형 변환을 사용
      (double)myInt / myDouble
      더욱 일반적인 사용; 변수에 변환 연산자가 사용됨

# 축약 연산자(Shorthand Operators)

- 증가 & 감소 연산자
  - 적절한 축약 표현
  - 증가 연산자, ++
    intVar++; 다음의 식과 같다
    intVar = intVar + 1;
  - 감소 연산자, --
    intVar--; 다음의 식과 같다
    intVar = intVar – 1;

# 축약 연산자(Shorthand Operators): 두 가지 형태

- 사후 증가(Post-Increment)
  intVar++
  - 현재 변수의 값을 사용한 후, 변수의 값이 증가됨
- 사전 증가(Pre-Increment)
  ++intVar
  - 변수의 값이 먼저 증가하고, 변경된 값이 사용됨
- 현재 사용되는 값이 무엇인가에 따라 결정
- 단독으로 사용되면 두 표현의 차이는 없다:
  intVar++; 와 ++intVar; → 동일한 결과

# 사후 증가

- 사후 증가의 예:

```
int           n = 2, valueProduced;
valueProduced = 2 * (n++);
cout << valueProduced << endl;
cout << n << endl;
```

  - 이 코드 부분은 다음의 결과를 출력:

    4
    3

  - 사후 증가가 사용됨

# 사전 증가

- 사전 증가의 예:

```
int           n = 2, valueProduced;
valueProduced = 2 * (++n);
cout << valueProduced << endl;
cout << n << endl;
```

  - 이 코드 부분은 다음의 결과를 출력:

    6
    3

  - 사전 증가가 사용됨

# 콘솔 입/출력

- 입출력(I/O) 객체 : cin, cout, cerr
- <iostream> 이라 불리는 C++ 라이브러리에 정의되어 있다
- 전처리기 지시자(processor directives)라 불리는 라인이 파일의 시작부에 명기되어야 함:
  - #include <iostream>
    using namespace std;
  - C++에게 적합한 라이브러리를 사용한다고 알려서 I/O 객체인 cin, cout, cerr을 사용한다

# 콘솔 출력

- 출력 가능한 것이 무엇인가?
  - 어떠한 데이터도 디스플레이 스크린에 출력 가능
    - 변수
    - 상수
    - 문자상수(Literals)
    - 수식(위의 모두를 포함 가능)
  - cout << numberOfGames << " games played.";
    두 가지 값이 출력:
    변수 numberOfGames의 값과,
    문자상수 문자열 “games played.” 출력
- Cascading: 하나의 cout으로 다양한 값을 출력

# 새로운 라인에서의 출력

- 출력에서의 줄 바꿈
  - 상기: "\n" 은 “줄바꿈” 문자를 나타내는 백슬래쉬 문자상수(escape sequence)
- 두 번째 방법: endl 객체
- 예:

```
cout << "Hello World\n";
```

  - 디스플레이에 문자열 “Hello World”를 보내고, 백슬래쉬 문자상수 “\n”에서 다음 라인으로 줄바꿈

```
cout << "Hello World" << endl;
```

  - 위와 동일한 결과

# 출력 형식

- 숫자 값의 출력 형식
  - 값이 프로그래머의 의도대로 출력되지 않을 수도 있다!

    cout << "The price is $" << price << endl;
    - 변수 price 의 값이 78.5 이면 다음과 같은 형태로 출력 :
      - The price is $78.500000 또는:
      - The price is $78.5
- 프로그래머는 C++에게 어떻게 숫자를 출력할 것인가를 명시적으로 알려줘야 한다!

# 숫자의 형식

- “Magic Formula”를 이용하여 소수점의 크기를 결정:

  ```
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);
  ```

- 이들 표현으로 값을 원하는 대로 출력 가능:
  - 정확하게 소수점 이후 두 자리까지 출력
  - 예:

    ```
    cout << "The price is $" << price << endl;
    ```

    - 이에 따른 결과:
      The price is $78.50
- 정밀도를 프로그래머의 의지대로 수정 가능!

# 오류(error) 출력

- cerr을 이용한 출력
  - cerr 은 cout과 동일하게 동작
  - 일반 출력과 오류 출력을 구분하는 매커니즘을 제공
- 출력 스트림의 재지정
  - 대부분의 시스템은 cout과 cerr 출력문을 파일과 같은 다른 장치로 "재지정"할 수 있도록 허용
    - 예: line printer, output file, error console 등

# cin을 이용한 입력

- cin은 입력, cout은 출력에 사용
- 차이점 :
  - “>>” (추출 연산자:extraction operator) 방향이 반대
    - “데이터가 가는 방향”을 가리킨다고 생각
  - “cout” 대신 “cin” 을 사용
  - cin에서 문자 상수 입력은 허용하지 않음
    - 변수만 입력해야 한다
- cin >> num;
  - 키보드에서 스크린 상에 입력될 때까지 대기
  - 키보드에서 입력된 값은 num 변수에 “할당’됨

# 입력을 위한 프롬프트: cin 과 cout

- 사용자 입력을 위해 항상 “대기”
  cout << "Enter number of dragons: ";
  cin >> numOfDragons;
  - cout에 “\n”이 없다. 프롬프트는 다음과 같이 같은 라인에서 사용자의 입력에 대기한다:

    Enter number of dragons: ____

    - 키보드 입력의 위치는 배경 위의 밑줄에 만들어진다
- 모든 cin은 cout 프롬프트를 가져야 한다
  - 사용자 친화적인 input/output

# 프로그램 스타일

- 기본 정책: 프로그램을 쉽게 읽고 수정할 수 있도록 작성해야 한다
- 주석, 두 가지 방법:
  - // 두 슬래쉬가 표시되는 하나의 라인 전체가 무시된다
  - /*주석 기호표시 사이의 모든 라인이 무시된다*/
  - 두 가지 방법 모두 일반적으로 사용된다
- 식별자 명명
  - 상수 : ALL_CAPS
  - 변수 : lowerToUpper
  - 가장 중요한 점: **의미있는 이름!**

# 라이브러리

- C++ 표준 라이브러리
- #include <Library_Name>
  - 현재의 프로그램에 라이브러리 파일의 내용을 "추가"하는 지시자
  - 전처리기 지시자(preprocessor directive)
    - 컴파일러가 실행되기 전, 라이브러리의 파일을 현재의 프로그램에 "복사"
- C++ 많은 라이브러리가 포함되어 있다
  - Input/output, math, strings, etc.

# 네임스페이스(Namespaces)

- 네임스페이스의 정의:
  - 정의된 이름의 집합
- 현재: “std” 네임스페이스에만 초점을 둔다
  - 필요로 하는 모든 표준 라이브러리 정의를 가지고 있음
- 예:

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

  - 정의된 이름의 모든 표준 라이브러리를 포함
- 

```
#include <iostream>
using std::cin;
using std::cout;
```

  - 원하는 객체만 선택할 수 있다

# 요약 1

- C++ 은 대소문자를 구분
- 의미있는 이름을 사용
  - 변수와 상수에 대하여
- 변수는 사용되기 전 반드시 선언되어야 함
  - 또한 초기화 되어야 한다
- 숫자의 사용에 있어서 주의할 점
  - 정밀도, 괄호, 연산 우선순위 등
- #include를 사용하여 필요할 때 C++ 라이브러리를 사용

# 요약 2

- cout 객체
  - 콘솔 출력에 사용
- cin 객체
  - 콘솔 입력에 사용
- cerr 객체
  - 오류 메시지 출력에 사용
- 주석을 사용하는 목적은 프로그램의 이해를 돕는 것
  - 남용하지 말 것

# Q&A