C/C++ Day 1

윤광진, 광주과학기술원 02/05/2018

0 준비

우선 Visual Studio 2017 community 와 C++ 툴킷을 설치합니다 (https://docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation).

윈도우 환경에서 C/C++ 프로그램을 개발한다면 Visual Studio가 가장 강력한 개발 도구입니다. Intellisense 를 비롯해서 편리한 디버깅 환경을 제공해 주기 때문입니다. 그 중에 Visual Studio 2017은 가장 최신 버전으로 최신 C++표준을 지원하며 과거의 표준(C++98, C++03)도 문제없이 사용가능합니다.
(https://docs.microsoft.com/en-us/cpp/cpp-conformance-improvements-2017)

이 강의에서는 C++03 표준을 기준으로 진행될 것이며 C++ 표준 및 최신 C++ 표준 사용법에 관한 이야기는 이 강의의 범주를 벗어나므로 포함하지 않겠습니다. 다만 C++ 표준에 대해 더 알고 싶은 분을 위해 위키 백과의 링크를 포함합니다 (https://en.wikipedia.org/wiki/C%2B%2B#Standardization).

0.1 Project 생성

Visual studio 2017에서 새로운 프로젝트를 생성하는 법은 **File** → **New** → **Project ...** 에서 새로 생성할 프로젝트의 경로 및 이름을 지정해주고 **Empty Project** 를 선택하면 됩니다.

0.2 Visual Studio 2017 준비

Visual Studio 2017설치를 마쳤다면 강의 진행에 알맞게 Visual Studio 2017 을 세팅해야합니다.

첫째로, Microsoft의 C++ 컴파일러에서 Deprecated function을 사용할때 발생하는 에러를 해결해야합니다. Microsoft의 C++ 컴파일러는 일부 POSIX (https://en.wikipedia.org/wiki/POSIX) 함수들을 Microsoft-specific 함수로 변경하였습니다. (https://msdn.microsoft.com/en-us/library/ms235384.aspx) 이것은 코드의 운영체제에 따른 이식성을 떨어뜨릴수 있고 Microsoft-specific 함수를 따로 공부해야하는 불편함을 주기 때문에 POSIX 함수를 그대로 쓸 수 있도록 하겠습니다. 이를 해결하기 위해 **Project** → **Properties** → **C/C++** → **Preprocessor** → **Preprocessor Definitions** 에 **_CRT_SECURE_NO_WARNINGS** 을 입력합니다.

둘째로, 프로그램의 실행이 완료됨과 동시에 콘솔창이 사라지는 것을 막기 위한 세팅입니다. 프로그램의 출력이 콘솔창을 통해 이루어지는 경우 콘솔창 사라지지 않고 남아있어야 결과를 확인할 수 있습니다. 해결법은 **Project** → **Properties** → **Linker** → **System** → **SubSystem** 을 **Console (/SUBSYSTEM:CONSOLE)** 로 변경하면 됩니다.

1 자료형

표 1에 C/C++에서 사용되는 자료형과 그 크기(size)를 나타냈다.

Group	Type names	Notes on size / precision
Character types	char	Exactly one byte in size. At least 8 bits.
Character types	char16_t	Not smaller than char. At least 16 bits.
Character types	char32_t	Not smaller than char16_t. At least 32 bits.
Character types	wchar_t	Can represent the largest supported character set.
Integer (signed)	signed char	Same size as char. At least 8 bits.
Integer (signed)	signed short int	Not smaller than char. At least 16 bits.
Integer (signed)	signed int	Not smaller than short. At least 16 bits.
Integer (signed)	signed long int	Not smaller than int. At least 32 bits.
Integer (signed)	signed long long int	Not smaller than long. At least 64 bits.
Integer (unsigned)	unsigned char	(same size as their signed counterparts)
Integer (unsigned)	unsigned short int	(same size as their signed counterparts)
Integer (unsigned)	unsigned int	(same size as their signed counterparts)
Integer (unsigned)	unsigned long int	(same size as their signed counterparts)
Integer (unsigned)	unsigned long long int	(same size as their signed counterparts)
Floating-point types	float	
Floating-point types	double	Precision not less than float
Floating-point types	long double	Precision not less than double
Boolean type	bool	

Table 1: 자료형

Size	Unique representable values	Notes
8-bit	256	2^8
16-bit	65,536	2^{16}
32-bit	4,294,967,296	2^{32}
64-bit	18,446,744,073,709,551,616	2^{64}

Table 2: 자료형 크기별 표현 가능한 개수

표 2는 데이터 크기별 표현 가능한 값의 갯수다.

다음 Listing 1을 실행하여 결과를 확인하자.

Listing 1: 자료형

```
#include <iostream>
using namespace std;
void main() {
    cout << "Size of char : " << sizeof(char) << endl;
    cout << "Size of int : " << sizeof(int) << endl;
    cout << "Size of short int : " << sizeof(short int) << endl;
    cout << "Size of long int : " << sizeof(long int) << endl;
    cout << "Size of float : " << sizeof(float) << endl;
    cout << "Size of double : " << sizeof(double) << endl;
}
```

2 표준 입출력

여기서는 C/C++에서 사용되는 표준 입출력인 printf, scanf, cout, cin에 대해서 알아본다.

2.1 printf

Listing 2: printf

```
1 int printf( const char* format, ... );
```

Listing 2에 **printf** 함수의 정의가 있다. **const char*** 타입의 **format** 변수를 입력받아 그 내용을 출력하고 출력한 **문자열의 갯수**를 반환한다. 또한, 만약 **format**이 *format specifiers*를 포함(% 기호로 시작하는 문자열)하고 있다면 **printf**의 인자들(arguments)로부터 해당 내용을 대입받아 출력한다.

specifier	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal	integer 7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

Table 3: format specifier

위 표에 format specifier들이 있다. format specifier의 사용법은

%[flag][width][.precision][length][format specifier]

이다. (http://www.cplusplus.com/reference/cstdio/printf/)

다음 Listing 3을 타이핑하여 실행해 보고 결과를 확인하자.

Listing 3: printf example

```
1 #include <stdio.h>
2
3 void main()
4 {
5     printf("%d\n", printf("hello world\n"));
6 }
```

다음 Listing 4의 실행결과를 예측해 보자.

Listing 4: printf example

```
1 #include <stdio.h>
```

```
2
3  int main()
4  {
5      printf ("Characters: %c %c \n", 'a', 65);
6      printf ("Decimals: %d %ld\n", 1977, 650000L);
7      printf ("Preceding with blanks: %10d \n", 1977);
8      printf ("Preceding with zeros: %010d \n", 1977);
9      printf ("Some different radices: %d %x %o %#x %#o \n",
10             100, 100, 100, 100, 100);
11     printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
12     printf ("Width trick: %*d \n", 5, 10);
13     printf ("%s \n", "A string");
14     return 0;
15 }
```

2.2 scanf

Listing 5에 **scanf** 의 정의가 있다. (http://www.cplusplus.com/reference/cstdio/scanf/)

Listing 5: scanf

```
1  int scanf( const char* format, ... );
```

scanf는 표준 입력 함수로 *stdin(표준 입력)*[1] (http://www.cplusplus.com/reference/cstdio/stdin/)으로부터 **format**의 형태로 데이터를 읽어서 주어진 인자에 저장하는 함수다. 반환값은 입력에 성공한 인자의 갯수다.

다음을 실행해보고 결과를 확인하자.

Listing 6: scanf example 1

```
1  #include <stdio.h>
2
3  void main()
4  {
5      int a, b, c;
6      char txt[10];
7      printf("input three numbers with whitespace between them: ");
8      int i = scanf("%d %d %d", &a, &b, &c);
9      printf("you've entered: %d %d %d\ni: %d\n", a, b, c, i);
10     printf("input text: ");
11     i = scanf("%9s", txt);
12     printf("you've entered: %s\ni: %d\n", txt, i);
```

[1] 간단하게는 키보드로부터 받은 입력이라고 보면됨

```
13 }
```

format 안의 하나의 공백문자(whitespace character)는 **stdin**으로부터 특정 입력을 찾을때까지 여러개의 공백문자를 대치 할수 있다. 공백문자(whitespace character)는 blank, newline, tab 등이 있다. **format** 안에는 **printf**와 비슷하게 format specifier를 이용해 **stdin**으로부터 들어온 입력의 *type*을 지정해 줄수 있다.

다음의 결과가 어떻게 나올지 예상해 보자.

Listing 7: scanf example 2

```
1  #include <stdio.h>
2
3  int main ()
4  {
5    char str [80];
6    int i;
7
8    printf ("Enter your family name: ");
9    scanf ("%79s",str);   // input to GIST
10   printf ("Enter your age: ");
11   scanf ("%d",&i);  // input to 20
12   printf ("Mr. %s , %d years old.\n",str,i);
13   printf ("Enter a hexadecimal number: ");
14   scanf ("%x",&i);    // input to ff
15   printf ("You have entered %#x (%d).\n",i,i);
16
17   return 0;
18 }
```

2.3 cout, cin

C++에는 *stream*이라 불리는 추상 객체가 있는데 키보드나 모니터(출력), 파일 등으로부터(으로) 데이터를 입력받는(출력하는) 작업을 한다. 입력의 경우 stream은 키보드 또는 파일로부터 문자 혹은 데이터들을 순차적으로 읽어올 것이고, 출력의 경우 stream은 문자 혹은 데이터를 모니터 또는 파일로 순차적으로 표시한다. 다음 표 4 는 C++ 표준 라이브러리에 정의되어있는 stream 객체들이다.

stream	description
cout	standard input stream
cin	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

Table 4: stream objects

여기서는 **cout**과 **cin** 에 대해서 자세히 알아본다.

일반적인 표준 출력 장치는 스크린이며, **cout**을 이용해 스크린에 출력을 할 수 있다. 여기서 말하는 스크린이란 모니터를 말한다기 보다 콘솔 창을 말한다. **cout** (**cin** 포함)은 **std namespace**에 정의되어 있기 때문에 **std** 지정자(identifier)를 사용해야한다. **cout**을 이용한 출력은 *insertion operator* («)를 사용한다.

Listing 8: cout example 1

```
#include <iostream>
using namespace std;  // using std namespace
int main ()
{
  cout<<"Hello world\n";
  return 0;
}
```

Listing 9: cout example 2

```
#include <iostream>
using namespace std;  // using std namespace
int main ()
{
  int a = 2018;
  cout<<"Happy "<<a<<" !!!"<<endl;
  return 0;
}
```

Listing 9의 **endl**은 newline을 의미하며 마친가지로 **std namespace**에 정의되어 있다.

표준 입력 stream 객체는 **cin**이며, 키보드로부터 입력을 받을수 있다. *extraction operator* (»)를 사용해 입력을 받을 수 있다.

Listing 10: cin example 1

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

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

3 Control flow

아주 간단한 프로그램으로써 주어진 명령들을 순차적으로 실행하고 끝마치게 할수도 있지만, 그런 프로그램으로는 복잡한 작업을 처리하기 어렵다. 이번 절에서는 분기(**if, else, switch**)와 반복(**for, while**) 또 그것들의 조합으로 다양한 작업을 할 수 있는 프로그램을 만들어 보자.

3.1 if, else

if 키워드는 조건문(condition)이 만족되면 해당되는 문장 또는 블럭이 실행되게 한다. 다음 Listing 11에 if의 사용법이 나와있다.

Listing 11: if

```
1 if (condition) statement
```

if의 **contition**의 조건이 만족(**true**)된다면 **statement**가 실행되는 구조다. 만약 만족되지 않는다(**false**)면 **statement**는 실행되지 않고 무시된다. 조건문(**condition**)에는 비교 연산자 또는 논리 연산자를 사용할 수 있다 (표 5, 6).

Operator name	syntax
Equal to	a == b
Not equal to	a != b
Greater than	a > b
Less than	a < b
Greater than or equal to	a >= b
Less than or equal to	a <= b

Table 5: 비교 연산자

Operator name	syntax
Logical negation(NOT)	!a
Logical AND	a && b
Logical OR	a || b

Table 6: 논리 연산자

Listing 12: if

```
1 if (x == 100)
2   cout << "x is 100";
```

Listing 12에서 **x**의 값이 100 이라면 그 밑의 2번줄이 실행된다. **statement**는 한 줄의 실행 코드가 될 수도 있고 몇 줄에 걸친 *block*이 될 수도 있다. block의 시작과 끝은 중괄호({, })를 이용한다.

Listing 13: if

```
1 if (x == 100)
2 {
3     cout << "x is ";
4     cout << x;
5 }
```

Listing 13에서 **x**의 값이 100이라면 그 다음 중괄호에 포함된 2번에서부터 5번줄이 실행된다. 또한, 문법에 어긋나지 않는 선에서 들여쓰기(indentation), 띄어쓰기 및 줄 바꾸기 등은 코드에 아무 영향이 없다.

Listing 14: if

```
1 if (x == 100){    cout << "x is ";    cout << x; }
```

즉, 13와 14은 동일한 동작을 하는 코드이다.

if는 **else**와 함께 분기문을 더 세분화 할 수 있다. 다음은 **if, else**의 사용법이다.

Listing 15: if

```
1 if (condition) statement1 else statement2
```

Listing 15에서 **condition**이 **true**면 **statement1**을 실행하고 **false**면 **statement2**를 실행한다. 그리고 **if**와 **else** 사이에 **else if**를 이용해 분기문을 더 세분화 할 수 있다.

Listing 16: if

```
1 if (condition1) s1 else if (condition2) s2 else s3
```

Listing 16에서 **condition1**이 **true**면 **s1**을 실행하고 아니면 다시 **condition2**를 체크하여 **true**라면 **s2**를 실행하고 **false**라면 **s3**를 실행한다.

Listing 17: if

```
1 if (x > 0)
2   cout << "x is positive";
3 else if (x < 0)
4   cout << "x is negative";
5 else
6   cout << "x is 0";
```

Listing 17는 **x**가 양수인지 음수인지 또는 0인지를 체크하는 코드이다.

3.2 while, for

while, **for**를 통해 반복문의 사용법을 알아보자. 반복문은 보통 조건문이 주어지고 해당 조건문이 만족(**true**)하는 동안 반복문 안의 구문을 실행하자. **while**은 **expression**이 주어지고 **true**인 동안 **while**의 **statement**를 반복 실행한다 (Listing 18).

Listing 18: while

```
while (experssion) statement
```

바로 예제를 보자.

Listing 19: while

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

int main ()
{
  int n = 10;
  while (n>0) {
    cout << n << ", ";
    --n;
  }
  cout << "liftoff!\n";
}
```

Listing 19의 작동 순서를 나열해보자:

1. **n**을 10으로 초기화
2. **while**의 조건문 체크 (**n>0**)
   - **true**: statement 실행 (3 번으로)
   - **false**: 5 번으로
3. statement 실행
   - **cout**으로 **n** 값 출력
   - **n** 값 하나 줄이기 (**--n;**)
4. **while** 블럭의 끝 도달, 자동으로 2 번으로
5. **while** 블럭 바로 다음의 코드로 이동
   - **cout**으로 liftoff! 출력

다음은 **for** 반복문 이다.

Listing 20: for

```
for(initialization; condition; increase) statement
```

for 반복문도 마찬가지로 조건이 만족하는 동안 구문(statement)를 실행한다. **initialization**은 **for**문이 수행되기전 한번 실행되고, **increase**는 한번의 반복이 완료될때마다 실행된다. 즉, 다음과 같이 **for**문이 작동한다:

1. 최초 **for**문이 수행되기 직전 **initialization** 구문이 실행됩니다.
2. **condition** 조건문을 점검합니다. **true**라면 3번으로 이동해 **statement**가 실행되고 **false**라면 5번으로 이동해 **for**문을 빠져나옵니다.
3. **statement**를 실행합니다.
4. **increase**를 실행합니다. 그리고 2번으로 이동합니다.
5. **for**문을 빠져나와 다음 코드로 이동합니다.

다음 예제를 실행해 보자.

Listing 21: for

```
1  #include <iostream>
2  using namespace std;
3
4  int main ()
5  {
6    for (int n=10; n>0; n--) {
7      cout << n << ", ";
8    }
9    cout << "liftoff!\n";
10 }
```

Listing 21은 Listing 19과 동일한 출력결과를 낸다.

3.3 Jump statements

이 절에서는 Jump statements로써 **break**와 **continue**를 설명합니다. **break**는 반복문을 강제로 빠져나오게 한다.

Listing 22: break

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5    for (int n=10; n>0; n--)
6    {
7      cout << n << ", ";
8      if (n==3)
9      {
10       cout << "countdown aborted!";
11       break;
12     }
13   }
```

```
14 }
```

Listing 22의 **for**문은 그것의 조건문과 관계없이 **n**이 3일때 강제로 **for**를 빠져나온다.

continue는 반복문의 **statement** 중간에 위치해서 나머지 **statement**들이 실행되지 못하게 하고 바로 반복문의 제일 끝으로 이동하게 한다.

Listing 23: continue

```
1  #include <iostream>
2  using namespace std;
3  int main ()
4  {
5    for (int n=10; n>0; n--) {
6      if (n==5) continue;
7      cout << n << ", ";
8    }
9    cout << "liftoff!\n";
10 }
```

Listing 23는 카운트 다운 중 숫자 5를 빠뜨리게 된다.

4 파일 입출력

여기서는 파일 입출력의 사용법을 알아보자. 프로그래밍을 통해 작업을 하다보면 메모리에 저장된 데이터를 파일로 출력해 저장을 할 필요가 있다. 또는 연구나 프로그램 구동에 필요한 데이터가 파일 형태(텍스트파일, 바이너리파일)로 주어져서 메모리에 데이터를 로드를 해야할 필요가 있다. 파일 입출력을 하기위해서는 **fstream** 헤더가 필요하다. **fstream** 헤더에는 파일 입력을 위한 **ifstream** 클래스와 파일 출력을 위한 **ofstream**이 있다. 두 클래스 모두 **std namespace**에 속해 있다.

첫번째로 프로그램을 통해 생성된 데이터를 텍스트 파일로 저장하는 방법을 알아보자.

Listing 24: fout

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  void fileout()
5  {
6      const int n_data = 10;
7      int data[n_data] = { 0, };
8      for (int i = 0; i < n_data; ++i)   // create data
9          data[i] = (i+1) * 10;
10
11     ofstream out("input.txt");
12     if (out.fail())
```

```
13      {
14          cout << "Error: can not open file\n";
15          return;
16      }
17
18      for (int i = 0; i < n_data; ++i) // save data
19          out << data[i] << " ";
20      out.close();
21  }
```

Listing 24은 **n_data**개의 정수형 데이터(**data**) 를 **input.txt** 에 저장하는 코드이다.

다음 Listing25은 **input.txt**파일로부터 데이터를 읽어들이는 코드다. 미리 마련해둔 **data** 메모리에 불러온 데이터들을 저장한다. 그리고 20-21번 줄에서 불러온 데이터의 값을 출력한다.

Listing 25: fin

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  void filein()
5  {
6      ifstream input("input.txt");
7      if (input.fail())
8      {
9          cout << "Error: can not open file\n";
10         return;
11     }
12
13     const int n_data = 10;
14     int data[n_data] = { 0, }; // data memory to load
15     int i = 0;
16     while (!input.eof())
17     {
18         input >> data[i]; // load data
19         ++i;
20     }
21     input.close();
22     for (i = 0; i < n_data; ++i) // print data
23         cout << data[i] << endl;
24 }
```