Introduction to C++ Programming

C++

- 간단한 역사
  - 1980년대 초 Bjarne Stroustrup (http://www.stroustrup.com/)에 의해 개발
  - 1967년 스칸디나비아에서 개발된 Simula-67에 기초한 언어임
  - C++ = C + Simula의 클래스
  - 초기에는 C with Classes라고 불림
  - 향상된 C라는 의미에서 C++로 명명됨
- C++의 특징
  - 객체지향 개념의 장점 + C의 효율성
  - 향상된 C로서의 장점: ANSI/ISO C와 더불어 발전됨

A First Program

```
/*
    첫 C++ 프로그램
*/
# include <iostream>  // 입/출력을 위한 다양한 클래스를 포함함
# include <string>    // STL 라이브러리에 정의된 표준 string 클래스

using namespace std ;

int main() {
    cout << "Enter your name: " ;
    string name ;
    cin >> name ;

    cout << "Hello, " << name << endl ;
}
```

주석(comment)

```
/*
    첫 C++ 프로그램
*/

# include <iostream>  // 입/출력을 위한 다양한 클래스를 포함함
# include <string>    // STL 라이브러리에 정의된 표준 string 클래스

using namespace std ;

int main() {
    cout << "Enter your name: " ;
    string name ;
    cin >> name ;

    cout << "Hello, " << name << endl ;
}
```

헤더파일(header file)

- 변수(variable), 함수(function), 클래스(class)를 선언한 파일

```
/*
  첫 C++ 프로그램
*/
# include <iostream>
# include <string>

using namespace std ;

int main() {
  cout << "Enter your name: " ;
  string name ;
  cin >> name ;

  cout << "Hello, " << name
     << endl ;
}
```

```
#include <streambuf.h>

class ostream : virtual public ios {
  public:
    ostream() { }
    ostream(streambuf* sb, ostream* tied=NULL);

    ostream& write(const char *s, streamsize n);
    ostream& write(const unsigned char *s, streamsize n) ;

    ostream& operator<<(char c);
    ostream& operator<<(unsigned char c) ;
    ostream& operator<<(signed char c) ;
    ostream& operator<<(const char *s);
    ostream& operator<<(const unsigned char *s) ;
  ...
} ;
```

헤더파일(header file)

- C 언어 헤더파일의 사용

```
// #include <string.h>
size_t strlen ( const char * str );
char * strcpy ( char * destination, const char * source );
char * strcat ( char * destination, const char * source );
int strcmp ( const char * str1, const char * str2 );
char * strrchr ( const char * str, int character );
char * strstr ( const char * str1, const char * str2 );
```

```
// #include <stdio.h>
int printf ( const char * format, ... );
int scanf ( const char * format, ... );
```

```
// #include <stdlib.h>
int abs ( int n );
void exit ( int status );
int system ( const char * command );
void * malloc ( size_t size );
void free ( void * ptr );
int atoi ( const char * str );
double atof ( const char * str );
```

헤더파일(header file)

- C 언어 헤더파일의 사용

C 프로그램	C++ 프로그램
# include <string.h> # include <stdlib.h> # include <stdio.h>	# include <cstring> # include <cstdlib> # include <cstdio>

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

int main() {
  char *string = "Hello World";

  printf("%d₩n", strlen(string));

  return 0;
}
```

```
#include <cstdio>
#include <cstring>

int main() {
  char *string = "Hello World";

  printf("%d₩n", strlen(string));

  return 0;
}
```

표준 입/출력

	C 언어의 표준 입/출력	C++ 언어의 표준 입/출력
헤더파일	# include <stdio.h>	# include <iostream>
출력	printf("Hello, %s₩n", name) ;	cout << "Hello, " << name << endl ;
입력	int i ; float f ; char s[10] ; scanf("%d%f%s", &i, &f, s) ;	int i ; float f ; string s ; cin >> i >> f >> s ;

프로그램 인자

- 프로그램인자 예제

```
// arg.cpp
# include <iostream>
using namespace std ;

void main(int argc, char* argv[]) {
    cout << argc << endl ;

    for ( int i = 0 ; i < argc ; i ++ ) {
        cout << argv[i] << endl ;
    }
    cout << "End" << endl ;
}
```

```
% arg.exe I am Kim
4
arg.exe
I
am
Kim
End
```

함수(function)

- 함수
  - 여러 문장으로 구성되는 호출의 단위
  - main() 함수에서 프로그램이 시작

```
int main() {
    int intValues[] = {10, 20, 50, 30, 5} ;
    cout << findMax(intValues, 5) << endl ; // 50

    float floatValues[] = {10.5F, 97.3F, 50.1F, 30} ;
    cout << findMax(floatValues, 4) << endl ; // 97.3
}
```

함수 오버로딩(overloading)

- 매개변수의 수 및 타입이 다른 동일한 이름의 여러 함수를 정의

```
int findMax(int numbers[], int size) {       // 함수 오버로딩
    int maxValue = numbers[0] ;
    for ( int i = 1 ; i < size ; i ++ )
        maxValue = myMax (maxValue, numbers[i]) ;
    return maxValue ;
}
float findMax(float numbers[], int size) {   // 함수 오버로딩
    float maxValue = numbers[0] ;
    for ( int i = 1 ; i < size ; i ++ )
        maxValue = myMax (maxValue, numbers[i]) ;
    return maxValue ;
}
```

인라인(inline) 함수

- 인라인 함수
  - inline 키워드로 시작하는 함수
  - 함수의 목적 코드로 분기하는 대신에 인라인 함수의 소스 코드가 호출 코드에 삽입되어 컴파일
  - 일반 함수에 비하여 빨리 수행될 수 있는 이점

```
inline int myMax(int int1, int int2 ) { // 인라인 함수, 함수 오버로딩
    if ( int1 > int2 ) return int1 ;
    return int2 ;
}
inline float myMax(float f1, float f2 ) { // 인라인 함수, 함수 오버로딩
    if ( f1 > f2 ) return f1 ;
    return f2 ;
}
```

클래스(class)

- 객체지향 프로그램의 기본 단위

```
class Person {
private: // 전용 멤버
   string lastName, firstName ;
   int age ;
public: // 공용 멤버
   Person(const string& _first, const string& _last, int age=1) // 생성자
      : lastName(_last), firstName(_first) { // 멤버 초기화 목록
      this->age = age ;
   }
   ~Person() { /* ... */ }   // 소멸자
   void setName(const string& lastName) { this->lastName = lastName ; }
   // 멤버 함수 오버로딩
   void setName(const string& firstName, const string& lastName) {
      this->lastName = lastName ; this->firstName = firstName ;
   }
   string getName() const { return lastName + ", " + firstName ; }
   virtual void print() const {
      cout << lastName << ", " << firstName << ", age of " << age << endl ;
   }
} ;
```

클래스(class)

- 프로그램은 클래스로부터 생성된 객체를 이용하여 동작

```
int main() {
   Person p1("Kildong", "Hong"), p2("Hyunjin", "Yu", 26) ;

   p1.print() ;                    // Hong, Kildong, age of 1
   p2.print() ;                    // Yu, Hyunjin, age of 26
   p2.setName("Ryu") ;
   p2.print() ;                    // Ryu, Hyunjin, age of 26
}
```

Procedure-oriented Code in C

```
# define MAXSIZE 100

/* 데이터 */
char item[MAXSIZE];
int top = 0;

/* 함수 */
void push( char x) {
    if (top+1 < MAXSIZE)
        item[++top] = x;
}
char pop() {
    if (top > 0)
        return item[top--];
}
```

```
void main(void) {
    char x, y;

    /* 함수 호출 */
    push('a');
    push('b');

    x = pop();        /* 'b' */
    y = pop();        /* 'a' */
}
```

Object-oriented Code in C++

```
const int MAXSIZE = 100;
```

```
// 클래스
class Stack {
private :
    // 데이터 멤버
    char item[MAXSIZE];
    int top;
public :
    // 멤버 함수
    Stack() { top = 0; }
    void push(char x) {
        if (top+1 < MAXSIZE)
            item[++top] = x ;
    }
    char pop() {
        if (top > 0)
            return item[top--];
    }
};
```

```
// 객체 생성
Stack st1;
```

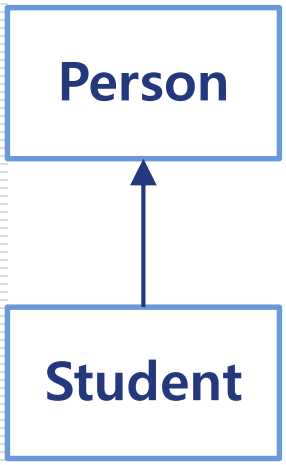
```
void main(void) {
```

```
    // 메시지 전송
    st1.push('a');
    st1.push('b');

    char x = st1.pop();
    char y = st1.pop();
}
```

상속(inheritance)

- 기존에 정의된 클래스의 모든 멤버를 물려 받아서 새로운 클래스를 정의하는 방법




```
class Student : public Person {
 string schoolName ;
 int year ;
public:
 Student(const string& firstName, const string& lastName,
    const string& _schoolName, int _year=1)
       : Person(firstName, lastName, _year+19), schoolName(_schoolName) {
    year = _year ;
 }
 int getYear() const { return year ; }
 void print() const {
    cout << getName() << "is " << year << " grader in "
       << schoolName << endl ;
 }
} ;
```

상속(inheritance)

- 하위 클래스에 대해서 새로 추가한 멤버뿐만 아니라 물려 받은 멤버를 사용

int main() {
  Student hong("Kildong", "Hong", "HK") ;
 	
  cout << **hong.getName()** << endl ;
  hong.**print**() ;
}
	Person으로부터 상속받은 getName()을 호출
	Student에서 새로 정의한 print()를 호출함

연산자 오버로딩(overloading)

- 클래스 객체에 대하여 연산자를 사용

```
int main() {
  Complex c1(10, 10), c2(20, 20) ;
  Complex c3 ;

  c3 = c1 + c2 ;              // + 연산자 오버로딩
  cout << c3 << endl ;        // << 연산자 오버로딩
}
```

```
30 + 30i
```

연산자 오버로딩(overloading)

- 클래스에 대한 연산자의 정의

```
class Complex {
private:
 double re, im ;
public:
 Complex(double re=0, double im=0) { this->re = re ; this->im = im ; }
 // 멤버함수로서 + 연산자 정의
 Complex operator+ (const Complex& c) const {
    return Complex(re+c.re, im+c.im) ;
 }
 double getRe() const { return re ; }
 double getIm() const { return im ; }
} ;
// 비멤버함수로서 << 연산자 정의
ostream& operator << (ostream& os, const Complex& c) {
 os << c.getRe() << " + " << c.getIm() << "i" ;
 return os ;
}
```

템플릿(template) 함수

- 기능은 동일하지만 다른 데이터 타입을 대상으로 동작하는 함수

```
template <class T>
T myMax(T int1, T int2 ) {
    if ( int1 > int2 ) return int1 ;
    return int2 ;
}
template <class T>
T findMax(T numbers[], int size) {
    T maxValue = numbers[0] ;
    for ( int i = 1 ; i < size ; i ++ )
      maxValue = myMax(maxValue, numbers[i]) ;
    return maxValue ;
}
```

```
int main() {
 int intValues[] = {10, 20, 50, 30, 5} ;
 cout << findMax(intValues, 5) << endl ;

 float floatValues[] = {10.5F, 97.3F, 50.1F, 30} ;
 cout << findMax(floatValues, 4) << endl ;
}
```

템플릿(template) 클래스

```
const int SIZE = 200 ;
template <class T> // 템플릿 클래스 Stack의 정의
class Stack {
    T elems[SIZE] ;
    int top ;
 public:
    Stack() { top = 0 ; }
    void push(const T& elem) { elems[top++] = elem ; }
    T pop() { return elems[--top] ; }
    bool isEmpty() const { return top == 0 ; }
} ;
```

```
int main() {
   Stack<int> is; // int 타입의 Stack
   is.push(100) ;
   cout << is.pop() << endl ;

   Stack<Complex> cos ; // Complex 클래스의 Stack
   cos.push(Complex(10, 10)) ;
   cos.push(Complex(20, 20)) ;
   cout << cos.pop() << endl ;
}
```

```
100

20 + 20i
```

예외처리(exception handling)

- 예외의 발생에 대한 조건 검사 및 처리를 지원

```
const char* const E_NON_POSITIVE_LENGTH = "Non positive length" ;
const char* const E_NOT_TRIANGLE = "Not a triangle" ;

int getTriangleLength(int x, int y, int z) {
    if ( x <= 0 || y <= 0 || z <= 0 ) throw E_NON_POSITIVE_LENGTH ;
    if ( x+y <= z || x+z <= y || y+z <= x ) throw E_NOT_TRIANGLE ;
    return x + y + z ;
}
```

```
int main() {
    int x, y, z ;
    cin >> x >> y >> z ;
    try {
       int totalLength = getTriangleLength(x, y, z) ;
       cout << totalLength << endl ;
    }
    catch (const char* const e) { cout << e << endl ; }
}
```

네임스페이스(namespace)

- 변수/함수/클래스 등의 이름에 대한 공간
- 예) std 네임스페이스
  - string
  - vector
  - cin, cout

```
# include <iostream> // 입/출력을 위한 다양한 클래스를 포함함
# include <string> // STL 라이브러리에 정의된 표준 string 클래스

int main() {
    std::cout << "Enter your name: " ;
    std::string name ;
    std::cin >> name ;

    std::cout << "Hello, " << name << std::endl ;
}
```

네임스페이스(namespace)

- using namespace 지시자(directive)

```
# include <iostream> // 입/출력을 위한 다양한 클래스를 포함함
# include <string> // STL 라이브러리에 정의된 표준 string 클래스

using namespace std;

int main() {
    cout << "Enter your name: " ;
    string name ;
    cin >> name ;

    cout << "Hello, " << name << endl ;
}
```

네임스페이스(namespace)

- 필요성
  - 식별자에 대한 유일한 이름을 부여하기 위하여 사용
  - 이름 충돌(name conflict)을 피함

```
namespace NS1 {
    int value ;
    void sort(int numbers[], int size) {
        /*quicksort*/
    }
}

namespace NS2 {
    int value ;
    void sort(int numbers[], int size) {
        /*bubblesort */
    }
}
```

```
void main() {
    NS1::value = 100 ;
    NS2::value = 200 ;

    int values[] = {20, 50, 10} ;
    NS1::sort(values, 3) ;
    NS2::sort(values, 3) ;

    NS1::Person p1 ;
    NS2::Person p2 ;
}
```

Coding Style

- The styles address
  - file organization
  - Indentation
  - Comments
  - Declarations
  - Statements
  - white space
  - naming conventions

언어	대표적인 코딩 스타일 정의
Java	● Oracle: Code Conventions for the Java Programming Language ● Google Android: Code Style Guidelines for Contributors
C++	● Google C++ Style Guide ● GCC Coding Conventions

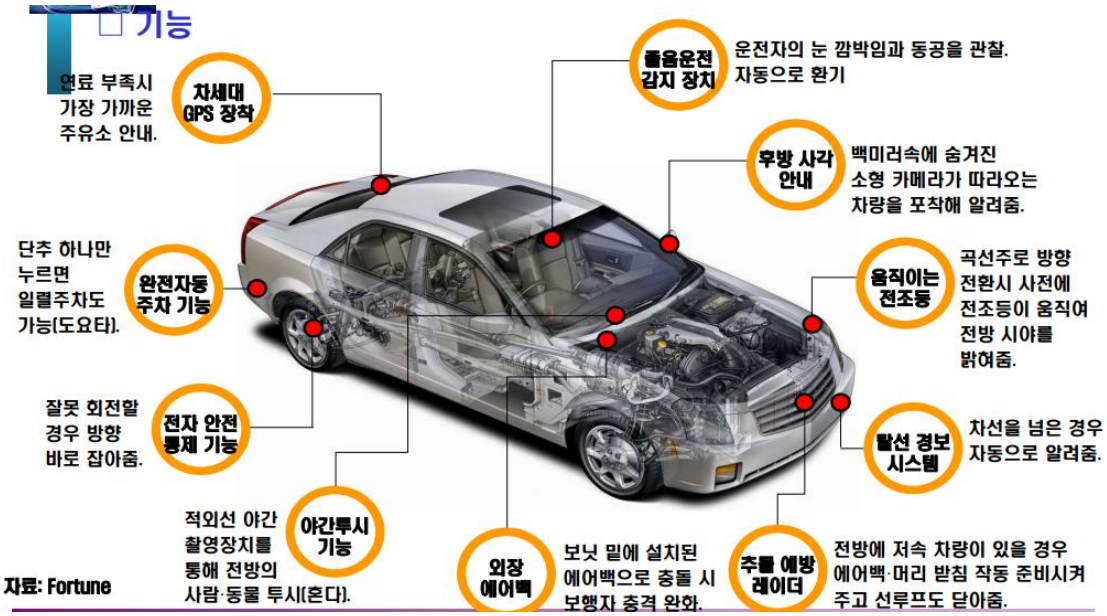
Coding Style

종류	설명	예
단일 문장	하나의 행은 오직 하나의 문장만을 정의한다	size = 10 ; minimumSize = 20 ; (X)
대입문	임베디드 대입문을 사용하지 않는다.	d = (a = b + c) + e ; (X)

복합문	if, while, for, do 문 등은 반드시 { } 로서 복합문을 가진다.	if (..) {
		
		}

공백	키워드 와 '(' 사이에는 공백을 넣는다. 연산자와 피연산자 사이에는 공백을 넣는다.	for (...) size += minimumSize ; size ++ ;

Safety-Critical Software



Coding Standards

- MISRA: The Motor Industry Software Reliability Association
  - C:1998, MISRA C:2004
  - MISRA C++:2008
- Flight
  - JSF C++: Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, Lockheed Martin Corporation, 2005
  - JPL: NASA/JPL Laboratory for Reliable Software
- The Power of 10
  - Rules for Developing Safety-Critical Code. IEEE Computer, 2006. 39(6): pp. 95-97
- Security
  - CERT C Secure Coding Standard
  - MITRE CWE(Common Weakness Enumeration)

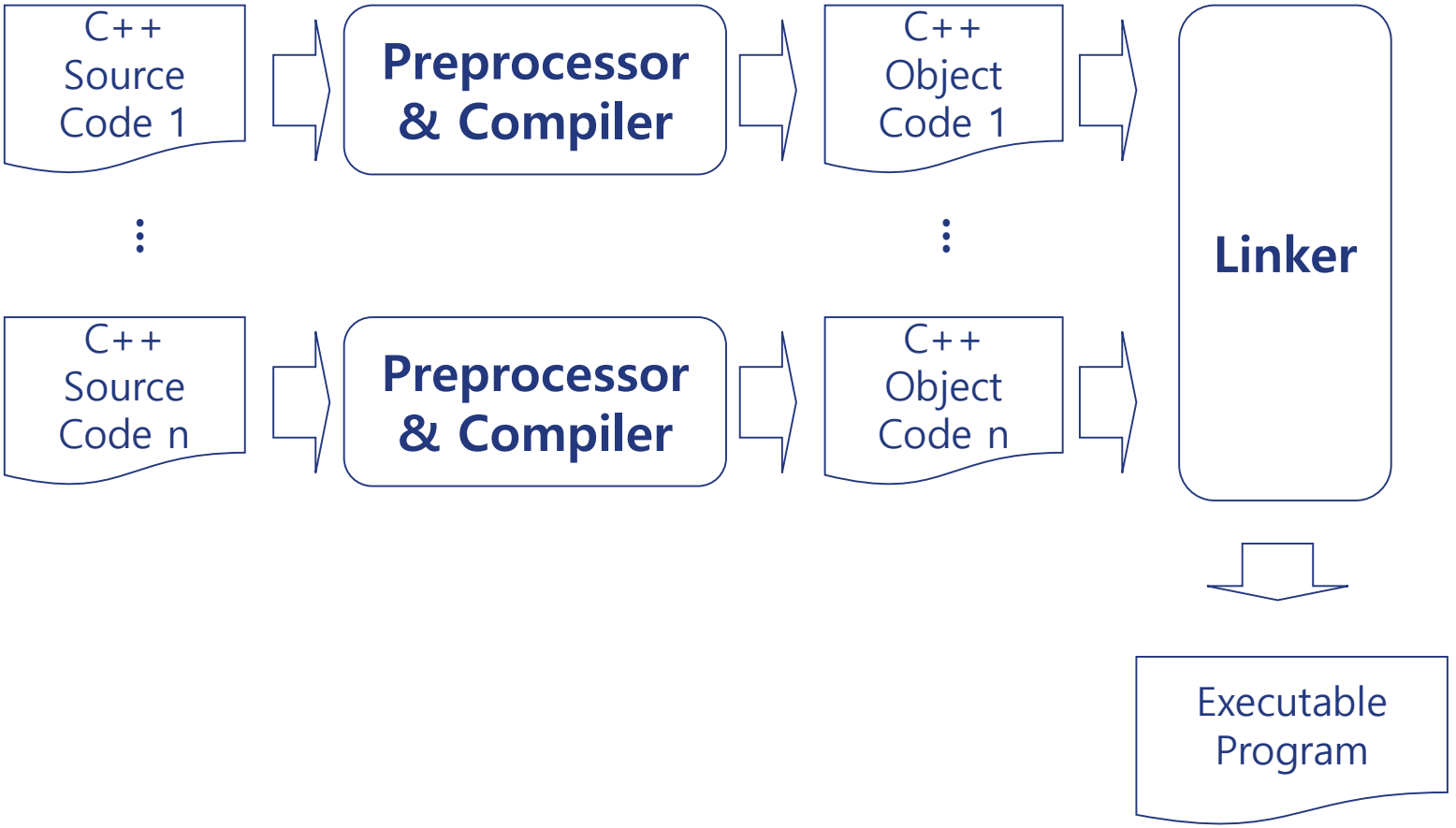
MISRA-C:2004 Rules Examples

- Find the violations of MISRA-C rules.

```
1: float result ;
2: float Program(char* option, int x, int y) {
3:    float result ;
4:    if ( x+y >= 10 && x+y <= 100 ) {
5:        char* option ;
6:        result = x*x + y++ ;
7:        option = (char*) malloc(100) ;
8:    } else if ( x == y*y || (! option) ) {
9:        result = Program(option, x*y, x+y) ;
10:       return result ;
11:  } else if (x = 50 )
12:       option++ ;
13:  return result ;
}
```

Build Process

- The Build Process



References

- Tutorials on Web
  - C++ Language Tutorial: http://www.cplusplus.com/doc/tutorial/
  - C++ Tutor: http://www.cpptutor.com/
- Advanced
  - Effective C++: 55 Specific Ways to Improve Your Programs and Designs
  - More Effective C++
  - C++ Gotchas: Avoiding Common Problems in Coding and Design; http://www.semantics.org/cpp_gotchas/index.html