C++ 언어

# C++이란?

C++은 기존의 C언어에 객체 지향 프로그래밍의 클래스 개념만을 추가하여 만든 언어

C++은 C언어에서 절차 지향적 언어의 특징을 가져왔습니다.

또한, Simula에서 클래스를 사용하는 객체 지향적 언어의 특징을 가져왔습니다.

그와 동시에 템플릿으로 대변되는 일반화 프로그래밍 방식의 언어이기도 합니다.

C++은 C언어를 기초로 삼아 만든 언어이므로, 기존의 C 표준 라이브러리를 그대로 사용할 수 있습니다.

# C++의 역사

C++은 C언어가 개발된 벨 연구소의 비야네 스트롭스트룹(Bjarne Stroustrup)에 의해 개발됩니다.

# C++의 특징

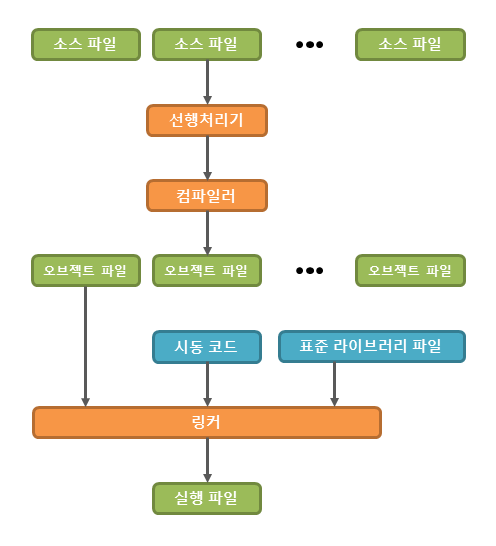
1. C++은 절차 지향적이며 구조적 프로그래밍 언어입니다.

2. C++은 객체 지향 프로그래밍 언어입니다.

3. C++은 일반화 프로그래밍 언어입니다.

# 프로그래밍

C++에서 소스 파일에서 실행 파일을 생성하는 순서



C++을 통해 작성된 소스 파일의 확장자는 대부분 .cpp 가 된다.

\* 선행처리

선행처리(preprocess)란 소스 파일 중에서도 선행처리 문자(#)로 시작하는 선행처리 지시문의 처리 작업을 의미합니다.

이러한 선행처리 작업은 선행처리기(preprocessor)에 의해 처리됩니다.

선행처리기는 코드를 생성하는 것이 아닌, 컴파일하기 전 컴파일러가 작업하기 좋도록 소스를 재구성해주는 역할만을 합니다.

\* 컴파일

컴퓨터는 0과 1로 이루어진 이진수로 작성된 기계어만을 이해할 수 있습니다.

소스 파일은 개발자에 의해 C++ 언어로 작성되기 때문에 컴퓨터가 그것을 바로 이해할 수는 없습니다. 따라서 소스 파일을 컴퓨터가 알아볼 수 있는 기계어로 변환시켜야 하는데, 그 작업을 컴파일(compile)이라고 합니다. 파일은 C/C++컴파일러에 의해 수행되며, 컴파일이 끝나 기계어로 변환된 파일을 오브젝트 파일(object file)이라고 합니다.

이러한 오브젝트 파일의 확장자는 .o 나 .obj 가 됩니다.

\* 링크

컴파일러에 의해 생성된 오브젝트 파일은 운영체제와의 인터페이스를 담당하는 시동 코드(start-up code)를 가지고 있지 않습니다.

또한, 대부분의 C++ 프로그램에서 사용하는 표준 라이브러리 파일도 가지고 있지 않습니다.

하나 이상의 오브젝트 파일과 라이브러리 파일, 시동 코드 등을 합쳐 하나의 파일로 만드는 작업을 링크(link)라고 합니다.

링크는 링커(linker)에 의해 수행되며, 링크가 끝나면 하나의 새로운 실행 파일이나 라이브러리 파일이 생성됩니다.

이처럼 여러 개의 소스 파일을 작성하여 최종적으로 링크를 통해 하나의 실행 파일로 만드는 것을 분할 컴파일이라고 합니다.

\* executable file

소스 파일은 선행처리기, 컴파일러 그리고 링커에 의해 위와 같은 과정을 거쳐 실행 파일로 변환됩니다.

최근 사용되는 개발 툴은 대부분 위에서 소개한 선행처리기, 컴파일러, 링커를 모두 내장하고 있으므로 소스 파일에서 한 번에 실행 파일을 생성할 수 있습니다.

이렇게 생성된 실행 파일의 확장자는 .exe 가 됩니다.

# C++ 프로그램

구조

#include 문

#define 문

int main()

{

명령문;

return 문;

}

ex>

#include <iostream>

#define TEXT "Welcome to C++ Programming!!"

int main()

{

std::cout << TEXT;

return 0;

}

\* main() 함수

C++ 프로그램은 가장 먼저 main() 함수를 찾고, 그곳에서부터 실행을 시작합니다.

따라서 모든 C++ 프로그램은 반드시 하나의 main() 함수를 가지고 있어야 합니다.

만약 main() 함수를 발견하지 못하면 C++ 컴파일러는 오류를 발생시킬 것입니다.

\* 명령문

C++ 프로그램의 동작을 명시하고, 이러한 동작을 컴퓨터에 알려주는 데 사용되는 문장을 명령문(statement)이라고 합니다.

이러한 C++의 모든 명령문은 반드시 세미콜론(;)으로 끝나야 합니다.

실행 결과

std::cout << "C++ 프로그래밍"; // 정상적으로 출력됨.

std::cout << "C++ 프로그래밍" // 오류가 발생함.

\* 반환문 (return)

반환문은 함수의 종료를 의미하며, 함수를 호출한 곳으로 결괏값을 반환하는 역할을 합니다.

특히 main() 함수가 반환되면, 프로그램 전체가 종료됩니다.

\* 선행처리문

#include 문과 #define 문은 모두 선행처리기에 의해 처리되는 선행처리문입니다.

#include 문은 외부에 선언된 함수나 상수 등을 사용하기 위해서 헤더 파일의 내용을 현재 파일에 포함할 때 사용합니다. C언어에서는 헤더 파일에 .h 확장자를 사용했지만, C++에서는 헤더 파일의 확장자를 사용하지 않기로 합니다. 따라서 기존 C언어 헤더 파일들의 이름 앞에 c를 추가하여 C++ 스타일의 헤더 파일로 변환하기도 합니다.

예제

#include <math.h> // C언어에서는 이 스타일만 허용됨.

#include <cmath> // C++에서는 이 스타일뿐만 아니라 위의 스타일도 사용할 수 있음.

물론 원칙은 위와 같이 바뀌었지만 C++ 프로그램에서는 기존의 C언어 스타일로도 헤더 파일을 사용할 수 있습니다. #define 문은 함수나 상수를 단순화해주는 매크로를 정의할 때 사용합니다.

\* 네임스페이스

네임스페이스란 이름이 기억되는 영역을 뜻하며, 이름이 소속된 공간을 의미합니다. 네임스페이스는 C++ 프로그램을 작성할 때 발생하는 이름에 대한 충돌을 방지해 주는 방법을 제공합니다. 이러한 네임스페이스는 C언어에는 없는 C++ 만의 새로운 기능입니다. C++ 프로그램의 표준 구성 요소인 클래스, 함수, 변수 등은 std라는 이름 공간에 저장되어 있습니다. 따라서 C++ 프로그램에서 표준 헤더 파일인 iostream 내의 정의를 사용하려면 다음과 같이 사용해야 합니다.

ex>

#include <iostream>

#define TEXT "Welcome to C++ Programming!!"

int main()

{

std::cout << TEXT;

return 0;

}

위의 예제처럼 std라는 네임스페이스에 있는 정의를 사용하려면, std:: 접두어를 붙여 해당 정의가 std라는 네임스페이스에 있다는 것을 컴파일러에 알려줘야 합니다.

이러한 네임스페이스에 속한 정의를 간단하게 사용하려면 다음과 같은 명령문을 추가하면 됩니다.

using namespace std; // std라는 네임스페이스에 속한 정의들은 네임스페이스 이름을 붙이지 않아도 사용할 수 있음.

앞서 살펴본 예제와 정확히 같은 동작을 수행합니다.

#include <iostream>

#define TEXT "Welcome to C++ Programming!!"

using namespace std;

int main()

{

cout << TEXT;

return 0;

}

# 주석

// C++ 한 줄 주석

/\* 여러 줄

// 두 줄 주석 안에 다른 한 줄 주석 삽입 가능 (단, 여러 줄 안에 여러 줄 주석 삽입 불가)

\*/

# iostream

\* 표준 입출력 클래스

C++에서는 cout 객체로 출력 작업을, cin 객체로 입력 작업을 수행하고 있습니다. 또한, C++에선 기존의 C언어 스타일처럼 printf() 나 scanf() 사용 가능!

\* cout 객체

cout 객체는 다양한 데이터를 출력하는 데 사용되는 C++에서 미리 정의된 출력 스트림을 나타내는 객체입니다.

문법 : std::cout << 출력데이터;

<<(삽입 연산자)

\* cin 객체

cin 객체는 다양한 데이터를 입력받는 데 사용되는 C++에서 미리 정의된 입력 스트림을 나타내는 객체입니다.

문법 -> std::cin >> 저장할변수;

추출연산자(>>)를 통해 사용자가 입력한 데이터를 입력 스트림에서 추출하여, 오른쪽에 위치한 변수에 저장. 이때 cin 객체는 자동으로 사용자가 입력한 데이터를 오른쪽에 위치한 변수의 타입과 동일하게 변환시킨다.

Ex>

#include <iostream>

using namespace std;

int main()

{

int age;

cout << "여러분의 나이를 입력해 주세요 : ";

cin >> age; // age에 string type 입력 시 0이 저장됨.

cout << "여러분의 나이는 " << age << "살 입니다." << endl;

return 0;

}

\* endl : 줄바꿈 문자

# 변수

\* 변수란 데이터를 저장하기 위해 프로그램에 의해 이름을 할당 받은 메모리 공간

숫자표현 관련 변수 - 정수형, 실수형

정수형 : char형, int형, long형 long long형 변수

실수형 : float형, double형

\* bit & byte

비트(bit) : 컴퓨터가 데이터를 처리하기 위해 사용하는 데이터의 최소 단위

바이트(byte) : 위와 같은 비트가 8개 모여서 구성되며, 한 문자를 표현할 수 있는 최소 단위

1byte = 8bit

\* 변수와 메모리 주소

변수는 기본적으로 메모리 주소를 기억하는 역

메모리 주소 : 물리적인 메모리 공간을 서로 구분하기 위해 사용되는 일종의 식별자

변수를 참조할 때는 메모리의 주소를 참조하는 것이 아닌, 해당 주소에 저장된 데이터를 참조하게 된다. 따라서 변수는 데이터가 저장된 메모리의 주소 + 저장된 데이터의 길이와 형태에 관한 정보도 같이 기억해야 한다.

\* 변수 선언 방법

변수 선언만 or 변수 선언후 초기화

Int num; / int num = 20;

반드시 선언한 변수의 타입과 일치하는 타입의 데이터만 할당해야 함.

Int num1, num2; / double num3 = 3.14, num4 = 4.56;

# 상수 (constant)

상수 : 변수와 다르게 값을 변경할 수 없는 데이터를 저장하는 메모리 공간

\* literal constant (리터럴 상수)

데이터가 저장된 메모리 공간을 가리키는 이름을 가지고 있지 않다.

적절한 메모리 공간을 할당받기 위해 변수든 상수든 타입을 가지고 있다.

1. 정수형 리터럴 상수 : 아라비아 숫자와 부호로 직접 표현 (8,10,16진수로도 표현 가능

Cout객체는 dec(10진수), hex, oct 조정자를 제공함.

이 세 가지 조정자를 cout 객체에 전달하면 변경 전까지 진법의 형태를 유지 가능

int a = 10;

cout << "숫자 10을 10진수로 표현하면 " << a << "이며, " << endl;

cout << oct;

cout << "숫자 10을 8진수로 표현하면 " << a;

\* 정수형 리터럴 상수는 두 가지 경우를 제외하고 모두 int형으로 저장됨.

1. 데이터 값이 너무 커서 int로 저장할 수 없는 경우

2. 정수형 상수에 접미사를 사용하여, 해당 타입을 직접 명시하는 경우

두 가지 경우를 제외하고 모두 int형으로 저장됨.

C++에서 접미사를 추가하여 나타낼 수 있는 정수형 타입을 나타냅니다.

|  |  |
| --- | --- |
| **접미사** | **타입** |
| 기본 설정 | (signed) int형 |
| u 또는 U | unsigned int형 |
| l 또는 L | (signed) long형 |
| ul 또는 uL 또는 Ul 또는 UL | unsigned long형 |
| long long 또는 u11 또는 U11 또는 uLL 또는 ULL | unsigned long long형 (C++11부터 제공) |

2. 실수형 리터럴 상수

실수형 리터럴 상수는 모두 부동 소수점 방식으로 저장됨.

|  |  |
| --- | --- |
| **접미사** | **타입** |
| f 또는 F | float형 |
| 기본 설정 | double형 |
| l 또는 L | long double형 |

3. 문자형 리터럴 상수