**C++ 문법 정리**

**>> 입출력**

#include <iostream> //입출력 헤더파일

\* C에서는 <stdio.h>

\* 헤더파일의 확장자는 .h이지만 C++에서는 직접 정의한 헤더파일이 아닌 표준 헤더파일 선언에서는 .h를 생략함

출력

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | std::cout << ”출력할내용”; | [cs](http://colorscripter.com/info#e) | |
| 결과 | 출력할내용 |

변수를 넣어도 된다

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | int a = 10;  std::cout << a; | [cs](http://colorscripter.com/info#e) | |
| 결과 | 10 |

연속하여 출력할 수도 있다

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | std::cout << ”Hello ” << ”World”; | [cs](http://colorscripter.com/info#e) | |
| 결과 | Hello World |

줄바꿈은 std::endl

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | std::cout << ”Hello” << std::endl << ”World”; | [cs](http://colorscripter.com/info#e) | |
| 결과 | Hello  World |

\* ‘\n’도 여전히 기능함

입력

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | char str[50];  std::cin >> str;  std::cout << str; | [cs](http://colorscripter.com/info#e) | |
| 입력 | Hello World |
| 결과 | Hello |

\* 변수는 미리 선언되어 있어야 하며, 공백 문자로 입력이 끊김

연이어 입력 받을 수 있다

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | char str1[50], str2[50];  std::cin >> str >> str2;  std::cout << str << str2; | [cs](http://colorscripter.com/info#e) | |
| 입력 | Hello World |
| 결과 | HelloWorld |

**>> 지역변수 선언**

변수 선언 위치에 제한을 두지 않는다

|  |  |
| --- | --- |
| for (int i = 0; i < 10; i++)      std::cout << “Hello World” << std::endl;    [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**>> 함수 오버로딩**

함수의 이름이 같아도 매개변수의 선언형태가 다르다면 C++은 구분할 수 있다

|  |  |
| --- | --- |
| int func(int a) {     return a;  }    int func(int a, int b) {      return a + b;  } | [cs](http://colorscripter.com/info#e) |

\* C언어는 함수의 이름만 이용하여 호출 대상을 찾고 C++은 이름뿐만 아니라 매개변수의 정보도 활용하기 때문

\* 매개변수의 자료형 또는 개수가 달라야 함. 반환형은 아무 상관없음

**>> 매개변수의 디폴트 값**

함수의 매개변수의 디폴트 값을 설정할 수 있다

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | #include <iostream>    int func(int num = 1) {      return num;  }    int main(void) {      std::cout << func() << std::endl;      std::cout << func(1);  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) | |
| 결과 | 1  1 |

\* 인자 수가 선언된 매개변수 수보다 작으면 왼쪽에서부터 채워져 나감. 즉, 오른쪽 매개변수부터 순서대로 디폴트 값을 채워줘야 함

\* 함수의 원형을 별도로 선언하는 경우 디폴트 값은 원형 선언에만 위치시켜야 함

|  |  |
| --- | --- |
| #include <iostream>    int func(int num = 1);    int main(void) {      std::cout << func() << std::endl;      std::cout << func(1);  }    int func(int num) {      return num;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

**>> 인라인 함수**

함수의 인라인화: 함수의 몸체부분이 함수의 호출문을 대체

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | inline int func(int num) {      return num + num;  }    int main(void) {      func(1);  } | [cs](http://colorscripter.com/info#e) | |
| 컴파일 후 | |  |  | | --- | --- | | inline int func(int num) {      return num+num;  }    int main(void) {      1 + 1;  } | [cs](http://colorscripter.com/info#e) | |

\* 컴파일러가 함수의 인라인화가 오히려 성능에 해가 된다고 판단되면 inline 키워드를 무시하기도 함

\* 매크로 함수는 자료형에 의존적이지 않지만 인라인 함수는 자료형이 다르면 데이터 손실이 발생하기도 함

**>> 이름공간**

이름공간이 다르면 같은 이름, 매개변수의 함수와 변수를 선언할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | #include <iostream>    namespace Ns1 {      void func(void) {          std::cout << "Ns1" << std::endl;      }  }    namespace Ns2 {      void func(void) {          std::cout << "Ns2" << std::endl;      }  }    int main(void) {      Ns1::func();      Ns2::func();  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) | |
| 결과 | Ns1  Ns2 |

\* ::은 범위지정 연산자라 부름. 이름공간을 지정할 때 사용함

\* 동일한 이름공간에 정의된 함수를 동일한 이름공간에서 호출할 땐 이름공간을 명시할 필요 없음

\* 이름공간 안에 또다른 이름공간을 만들 수 있음 외부이름공간::내부이름공간::변수/함수 이런 식으로 이어서 적으면 됨

\* 별칭을 정해줄 수도 있음 namespace 별칭=이름공간;

\* 함수 선언만 삽입할 경우

|  |  |
| --- | --- |
| #include <iostream>  namespace Ns1 {      void func(void);  }  namespace Ns2 {      void func(void);  }  int main(void) {      Ns1::func();      Ns2::func();  }    void Ns1::func(void) {      std::cout << "Ns1" << std::endl;  }    void Ns2::func(void) {      std::cout << "Ns2" << std::endl;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

using을 사용하면 이름공간을 지정하지 않고 함수나 변수를 호출할 수 있게 된다. (선언된 함수 내에서만 효력을 발휘함)

|  |  |
| --- | --- |
| #include <iostream>  namespace Ns {      void func(void) {          std::cout << "Ns" << std::endl;      }  }    int main(void) {      using Ns::func;      func();  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) |

\* using namespace 이름공간의이름 선언은 그 이름공간에 선언된 모든 것에 대해 이름공간 생략이 가능해짐

**>> 범위지정 연산자 ::**

|  |  |  |  |
| --- | --- | --- | --- |
| 코드 | |  |  | | --- | --- | | #include <iostream>    int a = 10;    int main(void) {      int a = 1;      a += 1; // 지역변수 a을 증가시킴      ::a += 10; // 전역변수 a를 증가시킴      std::cout << a << std::endl << ::a;  }  [*Colored by Color Scripter*](http://colorscripter.com/info#e) | [cs](http://colorscripter.com/info#e) | |
| 결과 | 2  20 |

>> 참조자

자신이 참조하는 변수를 대신할 수 있는 또 하나의 이름

int a = 10;

int &b = a;

이렇게 하면 a의 메모리 공간을 b로도 부를 수 있게 된다. 별칭 같은 것이다.

\* 참조자의 수에는 제한이 없음

\* 참조자를 대상으로 참조자를 선언하는 것 가능. 배열 원소 하나도 가능

\* 참조자 c가 포인터 변수 b를 선언하는 경우

int a = 0;

int \*b = &a;

int \*(&c) = b;

\* 무조건 선언과 동시에 NULL이 아닌 변수를 참조하도록 해야함

\* 함수 매개변수 선언 위치에 참조자 사용 가능 (이때 함수 내에서 참조자를 통한 값의 변경이 없을 경우 const를 사용해서 선언하는 게 좋음)

\* 함수의 반환형이 참조자일 수도 있음 int& func(const int &a) (지역변수를 반환하지 않도록 주의)

\* const로 선언된 변수를 참조하려면 참조자도 const로 선언해야 함. const로 선언된 참조자는 상수도 참조가 가능함

**>> new & delete**

new는 malloc과 비슷하고 delete는 free와 비슷하다.

|  |  |
| --- | --- |
| #include <iostream>  #include <string.h>    int main(void) {      int len = 20;        char \*str = new char[len];        strcpy(str, "Hello World");      std::cout << str;        delete []str;  } | [cs](http://colorscripter.com/info#e) |

\* 객체 생성에는 반드시 new & delete를 사용해야 함

\* 참조자 선언도 가능함 (결과로 1 출력됨)

|  |  |
| --- | --- |
| int \*ptr = new int;      int &ref = \*ptr;      ref = 1;      std::cout << \*ptr << std::endl; | [cs](http://colorscripter.com/info#e) |

**>> C언어의 표준함수 호출하기**

C언어의 헤더파일의 .h를 생략하고 앞에 c를 붙인다.

#include <stdio.h> => #include <cstdio>

#include <math.h> => #include <cmath>

\* 가급적이면 C++의 표준 헤더를 이용하는 게 좋음