01.C++ 소개

C++ 관련 추천 사이트

https://www.onlinegdb.com/online_c++_compiler : 온라인에서 c++ 코드 실행

왜 객체지향인가?

- 우리가 살고 있는 세상은 거의 모든 사물이 객체(object)로 이루어져 있음
- 실제 세계 반영하는 프로그래밍
- 소프트웨어 확장 및 재사용 기회 증가
-

프로그래밍 기본 용어(1)

원시 프로그램(source program)

- 소스 코드 라고도 하며 프로그래머가 작성한 프로그램
- C 언어로 작성된 프로그램은 .c 확장자 (c++은 .cpp)

컴파일

- 고급 언어들은 기계어로 번역하기 위해 번역가에 해당하는 컴파일러나 통역가에 해당하는 인터프리터 프로그램 필요(번역은 처음부터 끝까지 한꺼번에, 통역가는 매 구간마다)
- C 언어는 컴파일러 방식으로 방식으로 기계어 번역하며 대표적인 컴파일러로는 GCC C, Clang, Visual C++(IDE)
- 컴파팅하면 .obj 하는 오브젝트 파일이 생성되고 실행파일은 얻을 수 없다. ("obj 파일"은 컴파일러가 소스 코드를 기계어로 변환한 후 생성되는 중간 단계의 파일)

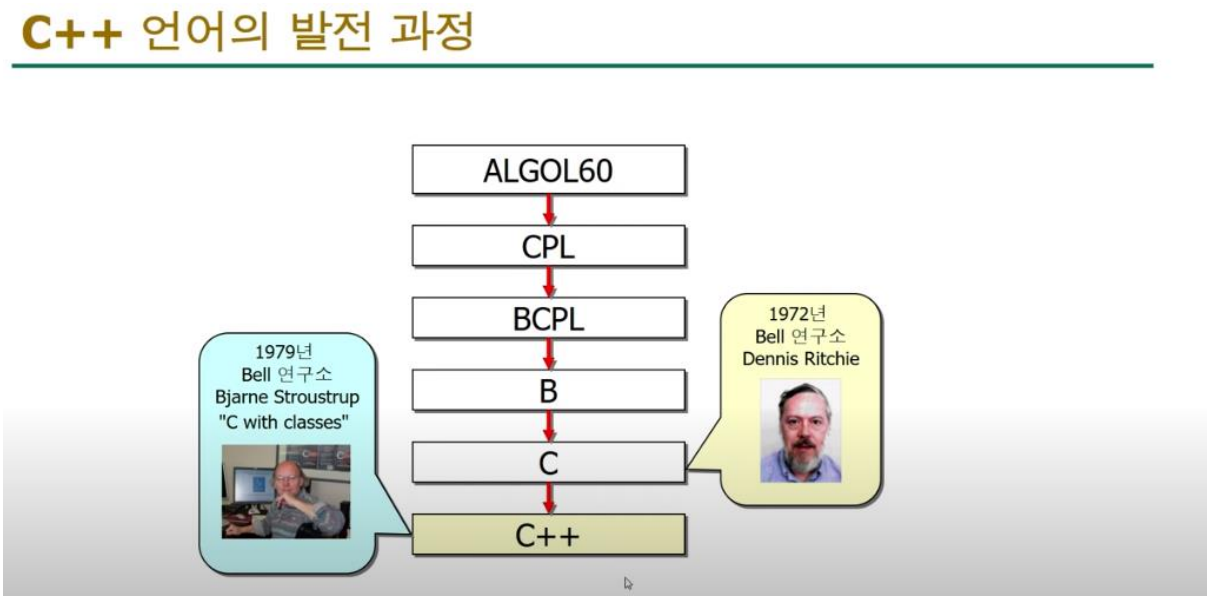
링킹(linking)

- 컴파일하여 만들어진 각 오브젝트 모듈(.obj 파일)을 연결
- 사용한 라이브러리(자주 사용되는 모듈 미리 컴파일해 놓은 것)와 연결하는 작업
- 링킹 후에 실행파일(.exe)를 얻을 수 있다., "linking"은 마치 퍼즐 조각들을 모아 하나의 완전한 그림을 만드는 것과 비슷

디버깅

- 컴파일과 링킹 과정에서 소스 코드에 오류가 있을 때 이런 오류나 버그를 수정하여 프로그램이 실행하도록 하는 작업

C++의 역사



- C 언어는 B 언어로 UNIX 라는 새로운 OS 를 만들었고 되도록 어셈블리 언어를 적게 사용하기 위해 C 언어로 만들었고 UNIX 의 대부분이 C 언어로 이루어져 급성장 이루었다.(어셈블리 언어: 인간보다 컴퓨터에 더 가까운 언어)
- C 컴파일러 개발하는 회사마다 제각각의 C 언어를 만들게 되어 표준 C 언어 문법 필요성 대두
- ANSI(미국 국제 기준 기관)에서 이런 표준 C 언어 만들기 위해 노력하여 ANSI C 와 ISO C 가 탄생
- 표준 C 언어는 계속해서 업데이트 되고있다

C++ 언어

- C++ -> C = C+1 -> C+class
- C with Classes 라는 뜻의 언어
- C 언어의 문법을 모두 포함, 개선
- C 문법은 포함하고 있지만 프로그래밍 스타일은 다름
- 객체지향 프로그래밍이 가능하도록 Class 위주로 다양한 문법 추가 (C 언어의 문법만 가지고 프로그래밍 할 수도 있기 때문에 완전한 객체지향 언어는 아니다)

전형적인 C 프로그램

```
#include <stdio.h>   // 표준 입출력 함수들을 사용하기 위한 헤더 파일 포함

#define SIZE 3   // 배열의 크기를 나타내는 매크로 정의

typedef struct {   // WEIGHT 라는 이름의 구조체 정의
    char name[10];   // 이름을 저장하는 문자열 멤버
    double w;          // 체중(w)을 저장하는 실수(double)형 멤버
```

```
} WEIGHT;

void swap(WEIGHT *, WEIGHT *);   // swap 함수의 원형 선언

int main(void) {
    WEIGHT man[SIZE] = {{"한개발", 57.5}, {"엄청군", 125.6}, {"갈비양", 35.7}};
    // man 배열을 선언하고 초기화

    int i, j;
    for (i = 0; i < 2; i++) {   // 체중을 기준으로 버블 정렬
        for (j = i + 1; j < 3; j++) {
            if (man[i].w < man[j].w) {   // 체중 비교 후 교환
                swap(&man[i], &man[j]);
            }
        }
    }

    printf(" 이름 ₩t 체중₩n");
    for (i = 0; i < 3; i++) {
        printf(" %s %5.1f₩n", man[i].name, man[i].w);   // 이름과 체중 출력
    }

    return 0;
}   // main() 함수 끝

void swap(WEIGHT *mani, WEIGHT *manj) {
    WEIGHT temp;   // 임시 구조체 변수 선언
    temp = *mani;
    *mani = *manj;   // 구조체 내용 교환
    *manj = temp;
}
```

토큰

- C 프로그램을 구성하는 기본단위

(1) 예약어

auto: 자동 변수의 선언 및 초기화를 나타냅니다.
break: 반복문이나 switch 문을 중단하고 빠져나올 때 사용됩니다.
case: switch 문에서 조건에 맞는 경우의 분기를 정의합니다.
char: 문자 데이터 타입을 나타내며, 문자를 저장하는 데 사용됩니다.

`const`: 변수가 수정될 수 없음을 나타내며, 상수 선언에 사용됩니다.
`continue`: 반복문의 현재 반복을 중단하고 다음 반복을 진행할 때 사용됩니다.
`default`: switch 문에서 어떤 분기에도 해당하지 않는 경우의 기본 동작을 정의합니다.
`do`: do-while 반복문의 시작을 나타내며, 조건 검사 후 코드 블록을 실행합니다.
`double`: 배정밀도 부동 소수점 데이터 타입을 나타내며, 더 넓은 범위의 실수를 저장합니다.
`else`: if 문에서 조건이 거짓일 때 실행되는 블록을 정의합니다.
`enum`: 열거형 데이터 타입을 정의하며, 일련의 정수 값을 가진 상수를 나타냅니다.
`extern`: 변수가 외부에서 정의되었음을 선언하며, 다른 파일에서 정의된 변수를 참조할 때 사용됩니다.
`float`: 단정밀도 부동 소수점 데이터 타입을 나타내며, 실수를 저장합니다.
`for`: 반복문의 시작을 나타내며, 조건을 만족하는 동안 코드 블록을 실행합니다.
`goto`: 프로그램 내에서 특정한 레이블로 이동할 때 사용됩니다.
`if`: 조건문의 시작을 나타내며, 주어진 조건이 참인 경우 코드 블록을 실행합니다.
`inline`: 함수를 인라인 함수로 선언하여 함수 호출 대신 함수 내용을 복사하여 사용하도록 합니다.
`int`: 정수 데이터 타입을 나타내며, 정수를 저장합니다.
`long`: 큰 정수 데이터 타입을 나타내며, 더 큰 범위의 정수를 저장합니다.
`register`: 레지스터 변수를 선언하여 빠른 접근을 허용합니다.
`restrict`: 포인터가 가리키는 메모리 영역을 제한하여 최적화를 허용합니다.
`return`: 함수에서 반환 값을 나타내며, 함수 실행을 종료하고 값을 반환합니다.
`short`: 작은 정수 데이터 타입을 나타내며, 더 작은 범위의 정수를 저장합니다.
`signed`: 정수 데이터 타입의 부호를 나타내며, 양수와 음수를 표현합니다.
`sizeof`: 데이터 타입이나 변수의 크기를 바이트 단위로 계산합니다.
`static`: 정적 변수를 선언하여 변수의 수명을 프로그램 수명과 동일하게 합니다.
`struct`: 사용자 정의 데이터 타입인 구조체를 정의합니다.
`switch`: 다중 분기 조건문을 나타내며, 특정 조건에 따라 분기를 선택합니다.
`typedef`: 새로운 데이터 타입에 대한 별칭을 정의합니다.
`unsigned`: 부호 없는 정수 데이터 타입을 나타내며, 양수만 표현합니다.

`void`: 함수의 반환 값이 없음을 나타내며, 데이터 타입을 지정하지 않을 때 사용됩니다.
`volatile`: 변수가 예상치 못한 방식으로 변경될 수 있음을 나타냅니다.
`while`: while 반복문의 시작을 나타내며, 주어진 조건이 참인 경우 코드 블록을 반복 실행합니다.
`_Bool`, `_Complex`, `_Imaginary`: C99 표준에 도입된 불리언, 복소수, 가상 허수 타입을 나타냅니다.

(2) 연산자

`+`: 덧셈 연산을 수행합니다.
`-`: 뺄셈 연산을 수행하거나, 부정 연산자로 사용됩니다.
`*`: 곱셈 연산을 수행합니다.
`/`: 나눗셈 연산을 수행합니다.
`%`: 나머지 연산을 수행합니다.
`=`: 대입 연산자로, 오른쪽 값을 왼쪽 변수에 할당합니다.
`+=`, `-=`, `*=`, `/=`, `%=` 등: 대입 연산과 함께 연산을 수행하고 결과를 왼쪽 변수에 할당합니다.
`==`: 두 값이 같은지 비교하고, 참(1) 또는 거짓(0)을 반환합니다.
`!=`: 두 값이 다른지 비교하고, 참(1) 또는 거짓(0)을 반환합니다.
`<`, `>`, `<=`, `>=`: 두 값을 비교하여 작거나 크거나 같은지를 판별하고, 참(1) 또는 거짓(0)을 반환합니다.
`&&`: 논리 AND 연산을 수행하여 두 조건이 모두 참인지 확인합니다.
`||`: 논리 OR 연산을 수행하여 두 조건 중 하나라도 참인지 확인합니다.
`!`: 논리 NOT 연산을 수행하여 조건을 부정합니다.
`&`: 비트 AND 연산을 수행합니다.
`|`: 비트 OR 연산을 수행합니다.
`^`: 비트 XOR 연산을 수행합니다.
`~`: 비트 NOT 연산을 수행합니다.
`<<`: 비트를 왼쪽으로 시프트하여 값을 증가시킵니다.
`>>`: 비트를 오른쪽으로 시프트하여 값을 감소시킵니다.
`&=`, `|=`, `^=`, `<<=`, `>>=` 등: 비트 연산과 함께 대입 연산을 수행하고 결과를 변수에 할당합니다.
`sizeof`: 피연산자의 크기를 바이트 단위로 반환합니다.
`->`: 구조체나 공용체 포인터를 사용하여 멤버에 접근합니다.
`.`: 구조체나 공용체의 멤버에 접근합니다.
`? :`: 삼항 연산자로, 조건을 검사하여 참일 때와 거짓일 때의 값을 반환합니다.
`,`: 연속적인 표현식을 구분합니다.
`&`: 변수의 주소를 반환하거나, 비트 AND 연산을 수행합니다.
`*`: 포인터 변수를 선언하거나, 포인터를 역참조하여 값에 접근합니다.
`sizeof`: 피연산자의 크기를 바이트 단위로 반환합니다.

++: 변수의 값을 1 증가시킵니다.
—: 변수의 값을 1 감소시킵니다.

(3) 구두점

;: 문장의 끝을 나타내며, 각 문장을 구분합니다.
:: 레이블을 정의하거나, 조건문 및 반복문에서 사용됩니다.
(,): 함수나 연산의 인자를 둘러싸고, 그룹을 나타내며, 함수 호출을 표시합니다.
{, }: 코드 블록의 시작과 끝을 정의하며, 여러 문장을 하나의 그룹으로 묶습니다.
[,]: 배열의 인덱스를 지정하거나, 배열 변수를 정의할 때 사용됩니다.
,: 연속적인 값이나 표현식을 구분합니다.
.: 구조체나 공용체의 멤버에 접근하는 데 사용됩니다.
–>: 구조체나 공용체 포인터를 사용하여 멤버에 접근하는 데 사용됩니다.

(4) 식별자

식별자: 변수, 함수, 타입 등을 명명하는 데 사용되는 이름

(5) 상수

(6) 문자열