제 1 장 프로그래밍 입문

**1-1 프로그래머**

### **프로그램**

* 문제를 풀기 위한 명령들의 조직적인 집합
* 현실의 문제를 해결하기 위해 존재한다.
* 최근 그래픽 환경에서 명령 외에 프로그램 외형을 장식하는데 데이터까지도 프로그램의 일부로 포함
* Ex) 프로그램 배경 장식 그림, 사용자 대화상자, 사운드, 아이콘, 메뉴 등 프로그램 사용법을 설명하는 도움말, 주의사항 등

### **프로그래머**

* 프로그램을 제작하는 사람(컴퓨터와 사용자 사이를 중계하는 통역자)
* 문법 🡺 명령을 올바르게 기술하는 규칙

🡺 프로그램을 구성하는 단위가 명령이므로 규칙에 맞게 명령을 정확하게 작성하는 지식이 필요

* 작문 🡺 프로그램은 단순한 명령의 나열이 아니라 조직적인 집합이므로 작성한 명령을 논리적으로 배치하는 기술이 필요함. 또한 인간의 창조력이 요구됨.

### **언어**

* 전기의 흐름, 차단 상태를 숫자로 표기하면 이진수가 되는데 이진수는 컴퓨터가 이해하는 유일한 언어
* 그래서 컴퓨터가 연산을 하도록 하려면 이진수로 된 기계어 명령을 전달 해야함.
* 프로그래머는 기계어 코드를 모르더라도 문법과 형식에 맞게 명령을 작성하면 됨.

**컴파일러** 🡺 중간의 번역 프로그램

* 명령은 컴파일러에 의해 기계어로 변환되며 컴퓨터는 이렇게 변환된 기계어를 실행하는 것.

저급언어(Low Level) 🡺 기계의 언어에 가까우며 사용하기는 어렵지만 성능은 좋음 🡪 어셈블리

고급언어(High Level) 🡺 인간의 언어에 가까우며 배우기 쉽지만 성능이 떨어짐. 🡪 베이직언어

But.. 저급이라 해서 성능이나 효율이 떨어진다는 것은 x

고급이라 해서 모든 것이 좋다는 것 x

인간의 입장에서 배우기 쉽고 쓰기 쉬운가를 기준으로 할 뿐

c/c++은 고급언어에 속함 하지만 섬세한 하드웨어 제어가 가능하기에 중급 언어로 분류하기도 함.

**1-2 C언어 소개**

### **c언어란?**

* 1972년 데니스 리치에 의해 개발됨
* 어셈블리는 특정 기계에 종속적이며 이식성이 없어서 여러 플랫폼에 수정없이 사용하는 것이 목적인 Unix와 어울리지 않았음
* 또한 포트란, 베이직 같은 고급언어들은 효율이 좋지 못해 대형 프로젝트에 쓰기에는 부적합했음
* **어셈블리의 기능과 고급언어의 이식성을 동시에 갖춘 언어** 🡪 C언어

But. 수많은 C컴파일러가 개발됨에 따라 많은 변종들이 생겨나게 되면서 사회적으로 낭비를 초래하였다.

* + 미 표준 위원회(ANSI)가 표준안을 완성해 90년에 ISO 9899 로 승인되었다.
  + 이때 제정된 C 표준을 ANSI C(또는 C90)라고 함.
  + 그 전의 C를 클래식 C(K&R C)라고 한다.

### **C vs C++**

* C언어 🡺 응용 프로그램에서 **기본적으로 처리해주는 작업**(윈도우 클래스, 출력형식)에 **변화를 주거나 프로그램 실행 속도가 빨라야 하는 프로그램**을 만들 때 좋음
* C++ 🡺 프로그램의 **규모가 커지고 유지 보수의 문제가 많이 예상될수록 좋음**

### **ANSI C 의 장점 ( 기존 K&R C보다 )**

1. 표준 라이브러리 함수를 규격화했으며 헤더 파일 통일
2. 함수의 원형 선언 기능이 추가되어 컴파일러가 함수 호출부에서 타입 체크를 할 수 있게 됨.
   1. **원형 선언** 이란 ?
      1. 함수의 반환형, 이름, 매개변수만을 먼저 선언해두는 것
      2. 매개변수의 이름을 적지 않고 자료형만 적어두는 것도 가능.
      3. Ex) int get\_result(double) 등
      4. 원형선언을 사용하면 int main() 함수를 먼저 짠 뒤에 나머지 함수들의 코드를 아래에 적을 수 있게 된다.
      5. 왜 쓰나? 🡺 코드 내에 어떤 함수가 있는지 알기 편하고, 함수를 정의하고 나머지 함수를 작업하는 사람의 경우 효과적
3. 정수, 실수 상수 타입을 지정할 수 있는 L, U, F등 접미어가 추가됨.
4. Enum, void 형과 const, volatile 제한자가 추가됨.
   1. Enum 🡺 열거형
   2. Void 🡺 리턴할 값이 없는 함수
   3. Const 🡺 상수형 (변하지않는 값) , 고정값
   4. Volatile 🡺 항상 메모리에 접근을 해야하는 함수에 적용하기.
5. 인접 문자열 상수를 합쳐 주고 확장열의 기능도 추가됨.
6. 함수 내부에서 선언하는 지역 배열이나 구조체를 초기화할 수 있음
7. 구조체끼리 대입할 경우 구조체 크기만큼 메모리 복사를 함.

### **C언어의 장점**

장점보단 언어의 특성 🡺 **성능 지향의 범용 언어**

1. 문법이 간결하다. 예약어가 적고 제공하는 연산자가 다양하기에 짧은 명령으로 일을 할 수 있다.
2. 효율성이 좋아 대규모의 고기능 프로그램을 만들 수 있다. 제작된 프로그램의 크기가 작고 속도가 빠르다.
3. 운영체제 개발을 목적으로 만든 언어이므로 저수준 프로그래밍이 가능하며 어셈블리 수준의 하드웨어 제어를 할 수 있다.
4. 모든 분야에 두루 활용할 수 있을 정도로 범용적이다. 사무 처리, 과학 기술, 상업용 등 거의 대부분 응용 프로그램을 개발 할 수 있다.
5. 성능을 위해 불필요한 에러 처리를 하지 않는다. 타입의 불일치나 배열 경계 점검등은 프로그래머가 직접 해야 하므로 개발자는 항상 주의해야한다.
6. 이식성이 좋다. 대부분 플랫폼에서 사용할 수 있으므로 재컴파일만 하면 타 운영체제에서 실행되는 프로그램을 쉽게 만들 수 있다.

**단점**

1. 배우기가 어렵다. 특히 포인터는 초보자들에게 두려움의 대상으로까지 여겨진다.
2. 가비지컬렉션 기능이 없다.

**입문용 언어로 C 언어가 유리한 이유**

1. C의 대중성 ( 이부분은 모호하다. 너무 옛날 글)
2. 실무에서 바로 쓸 수 있는 언어이다.

C언어는 혼합형 언어( 객체 지향 x )

* 객체를 몰라도 코딩할 수 있다는 장점.

**1-3 개발 툴**

### **컴파일러**

컴파일 하는 프로그램 (중간 번역 프로그램 ) 요즘은 개발 환경 자체를 의미.

문서작업 = 워드

계산 = 스프레드시트

개발 = 컴파일러

언어의 문법에 맞게 명령들을 기술한 파일 🡺 **원시파일(**Source File)

소스를 컴퓨터가 이해할 수 있는 기계어 코드로 번역하는 것 🡺 **컴파일**(Compile)

컴파일러는 소스 파일에 작성된 고급 언어 명령을 해석하여 기계어 코드로 바꾸고 그 결과를 목적파일(Object File)에 써 넣는다.

다시 말해

컴파일러 🡺 소스 파일을 목적 파일로 바꾸는 프로그램

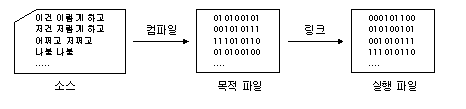
목적파일은 소스명령들을 번역한 기계어 코드를 가진 파일이 되고 이 파일은 곧바로 실행 x

왜냐하면 프로그램은 기계어 코드 외에도 운영체제가 요구하는 코드를 추가로 가져야 하기 때문임.

목적파일을 실행파일로 바꾸기 위해선 운영체제의 요건에 맞도록 형태를 바꾸고 스타트업(StartUp)이라는 추가 코드를 가져야 함.

목적파일에 이런 처리를 하여 실행파일로 만드는 동작 🡺 **링크(**Link)

링크 해주는 프로그램 🡺 **링커**(Linker)



해당과정 🡺 **빌드(**Build) 라고 함

빌드 하는 이유 🡪 여러 소스를 합쳐 하나의 실행 파일을 만들 수 있어야 하기에 함

최근은 IDE(통합개발환경)을 통해  개발 환경 내에서 편집, 개발, 디버깅, 최적화까지 다 할 수 있고 무척 편리하며 생산성 향상에 크게 기여하고 있다.

**1-4 컴파일러 종류**

텍스트이(가) 표시된 사진

자동 생성된 설명

실무에서 가장 많이 사용되는 컴파일러는 마이크로소프트사의 비주얼 C++

2-2 First 예제

First 분석

#include <stdio.h>

이 명령은 stdio.h 라는 헤더 파일을 포함하라는 뜻

앞으로 어떤 함수를 사용하겠다는 의사 표시

헤더 파일에는 C 컴파일러가 제공하는 표준 함수들에 대한 선언이 작성

Ex) stdio.h를 포함함으로써 printf가 무엇을 하는 함수인지 컴파일러가 알 수 있게 됨.

void main()

main은 함수 이름 C 프로그램은 항상 main 함수에서부터 실행을 함. 괄호()는 main이 함수라는 것을 알려주는 역할

void는 리턴값이 없다.

자바와 다른점

* 자바의 경우 줄바꿈은 println 으로 하면 되지만 C언어는 printf(“~~ \n”)으로 \n을 적어서 줄바꿈을 실행한다는 점을 기억할 것

제 2 장 첫 번째 예제

**2-3 예제 실습**

### 나. 숫자 맞추기 게임

텍스트이(가) 표시된 사진

자동 생성된 설명

1번줄🡺 Turboc.h 헤더파일은 요새는 잘 쓰지 않는 추세이다. 실습을 위해 강제로 헤더파일을 복사해 집어넣어서 실습을 진행했다.

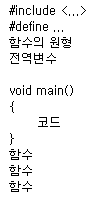
8번줄🡺 1번줄의 Turboc.h 헤더파일 안에 있는 함수로 기존의 rand 함수를 쓰면 난수가 시스템을 재실행할때 항상 같은 값으로 난수 설정이 되는데 randomize 함수를 활용하면 난수 값이 계속 바뀐다.

14번줄🡺 기존의 scanf 는 버퍼 오버플로우(BOF)등의 보안 취약점을 가지고 있어 에러가 난다. 그래서 안정적인 scanf\_s를 활용한다.

2-4 프로그램의 구성 요소

나. 프로그램의 구조

1. c언어는 대소문자를 구분한다.
2. 항상 main 함수로 시작한다. 함수 중에 반드시 있어야 하는 함수가 main이다. 컴파일러는 main부터 실행한다.
3. 세미콜론은 자연어의 마침표에 해당한다. 컴파일러는 세미콜론을 기준으로 문장을 구분하므로 세미콜론을 달아줘야한다.
4. 프리포맷(Free Format) 을 지원한다. 문법에만 맞으면 줄을 나눠서도 작성가능한다.
5. C 프로그램의 구조는 대체적으로 아래와 같다.



선두에 전처리문이 오고 이어서 함수 원형, 전역변수 선언이 온다.

제 3 장 변수

3-1.변수

### 변수의 정의

컴퓨터의 주 기억 장치는 메모리(RAM)이고 주된 일은 메모리에 기록된 값을 처리해 입출력한다.

컴퓨터는 연산 대상 메모리의 위치를 구분한다.

포인터 관련

**포인터변수**

### 포인터 변수 크기

포인터 변수의 크기는 무조건 4byte가 아니다.

**포인터 변수의 크기**는 시스템 운영체제의 영향보단 **컴파일러 운영체제의 영향이 크다.**

우선 컴퓨터 windows 운영체제는 64bit로 설정 되어있다.

이럴 경우에 컴파일러 운영체제의 크기로 포인터변수의 크기가 달라진다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 경우 컴파일러를 **64bit**로 설정하고 포인터 pi의 크기를 구해보았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 결과 포인터변수의 크기는 **8** 로 나타난다.

하지만 컴파일러를 **32bit**로 바꾸고 포인터변수 pi의 크기를 구해보면

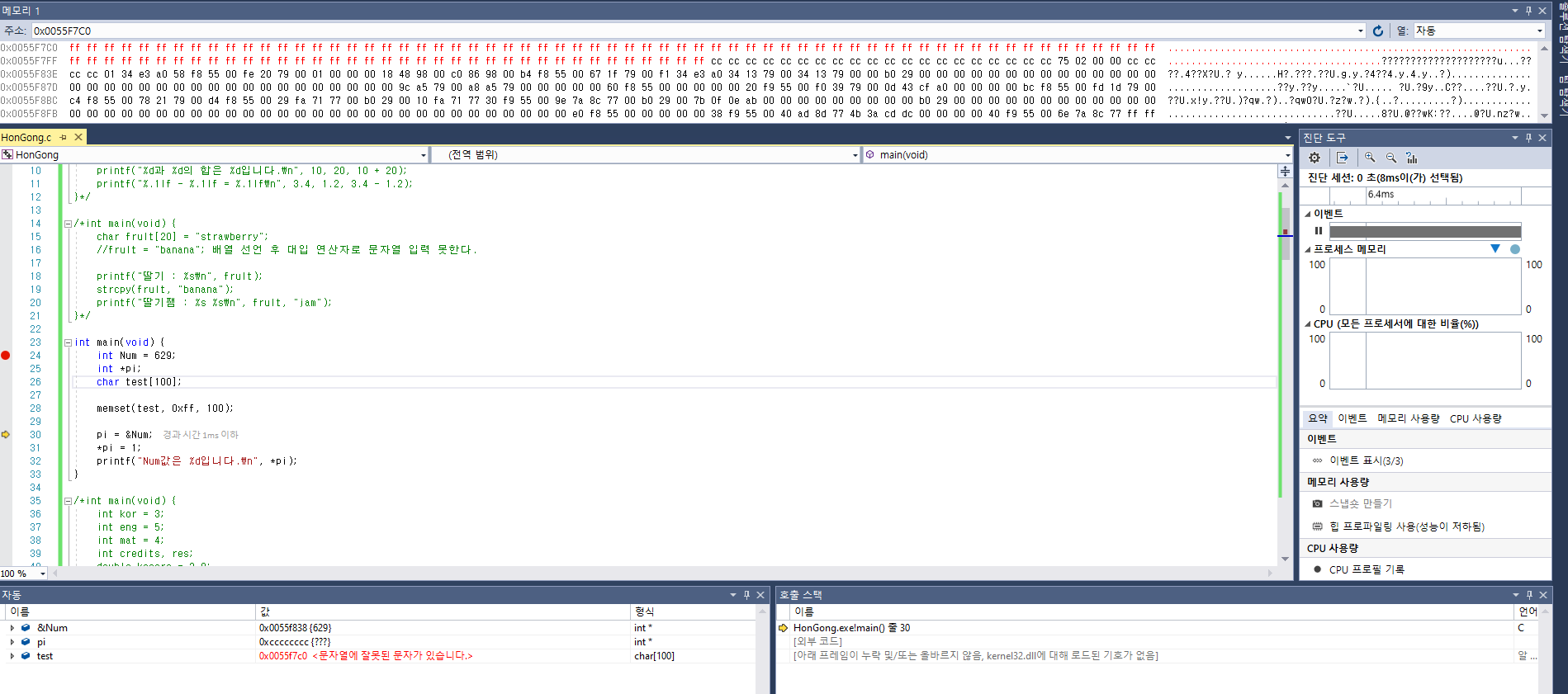
텍스트이(가) 표시된 사진

자동 생성된 설명

위의 사진과 같이 **4byte**로 나타나진다.

그렇기 때문에 포인터변수의 크기는 컴파일러 운영체제의 환경에 따라 달라진다는 점을 기억해야한다.

**디버깅 하는 방법** = 라인 앞 줄 번호 왼쪽 공간에다가 왼쪽마우스 클릭을 하면 빨간 동그라미 모양이 나온다. 그리고 나서 f5를 누르면 디버깅을 하면서 볼 수 있다.



그리고 모든 포인터 변수 자료형은 똑 같은 크기를 가지게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

의 결과로 32Bit 컴파일을 할 시에는 모두 4가 나오고 64bit 컴파일을 하면 8이 나온다.

왜 모두 같은 크기를 가지게 되는 것일까?

모든 자료형이 그렇듯 각 자료형은 해당 자료 표현을 위해 그 범위만큼 크기를 가진다.

예를 들어 char 자료형은 1 byte크기를 가진다.

테이블이(가) 표시된 사진

자동 생성된 설명

그리고 1byte 즉 8bit로 표현할 수 있는 범위는 위의 그림과 같다. 8bit는 256의 경우의 수를 표현 할 수 있고 이를 ascii 코드에 대응해 지정문자를 출력한다.

테이블이(가) 표시된 사진

자동 생성된 설명

위의 그림은 int 자료형이다. (4byte)

변수의 저장용량을 넘어서는 행위 🡺 오버 플로우

단어 및 코드 풀이

## **ㄱ**

### 구조체(Struct) – 사용자 정의 자료형

* 여러 자료형들을 조합해 내가 원하는 자료형으로 재생성하는 것
* 쓰는 이유 : 구조체자료형으로 변수 선언하게 되면 해당 변수는 자료형을 지정하지 않아도 사용가능하다 🡪 개발할때 불필요한 선언을 줄일 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 사진처럼 배열에서도 활용이 가능하다. Main 함수 내에서 Job자료형으로 job[] 배열의 형식을 선언한 후 배열 내의 값들을 대입하기만 하면 된다. ( 구조체를 쓰지 않았으면 각각 배열마다 자료형을 선언해줘야 하는 번거로움이 있었을 것이다.)

## ㄴ

## ㄷ

### 동적할당(malloc)

* 프로그램이 실행할 동안 사용할 메모리 공간을 할당한다. 사용이 끝나면 공간을 반납한다.
* 반대개념 <-> 정적할당(사용할 메모리 크기를 고려해 미리 할당이 이루어지는 것)
* Stdlib.h 표준 라이브러리에 정의 되어 있기 때문에 사용하기위해서 include 해줘야한다.

테이블이(가) 표시된 사진

자동 생성된 설명

## ㄹ

## ㅁ

## ㅂ

## ㅅ

### 시프트 연산자

## ㅇ

## ㅈ

## ㅊ

## ㅋ

## ㅌ

## ㅍ

## ㅎ

## a

## b

## c

## d

## e

## f

## g

## h

## i

## j

## k

## l

## m

### Memset 함수

* 메모리의 값을 원하는 크기만큼 특정 값으로 세팅한다.
* 함수원형 🡪 void\* memset(void\* ptr, int value, size\_t num);
  + Void\* ptr 🡪 세팅하고 하는 메모리의 시작주소(즉, 그 주소를 가리키고 있는 포인터가 위치하고 있는 자리
  + Int value 🡪 메모리에 셋팅하고자 하는 값 (int 타입으로 받지만 내부는 unsigned char로 변환 즉, ‘a’를 넣어도 무방
  + Size\_t num 🡪 길이 (byte 단위, 메모리 크기 단위의 길이 / “ 길이\* sizeof(데이터타입)” 형태로 작성)

## n

## o

## p

## q

## r

## s

### static

* 다른 파일에서 호출하지 못하는 정적함수( 해당 파일 내에서만 쓰게 처리할 수 있다.)
* static으로 선언을 하게 되면 해당 위치에서만 사용이 가능하다. 이 말은 즉, 다른 파일에선 동일한 이름으로 함수를 구성해 사용이 가능하다. 또한 static을 표시함으로써 코드분석시 해당함수가 어디서 사용되어있는지 빠르게 확인이 가능하다. 마지막으로 협업시 동일 함수명으로 중복되어버릴 경우를 미리 막을 수 있다.

## t

## u

## v

## w

## x

## y

## z