...그냥 빈 프로젝트로 만들어도 되는 듯 ...

C언어 - 클래스 라이브러리 제공 및 사용

JetBrains - C Lion cf> visual studio

c언어 + 객체지향 = c++
c with classes
- 호환성이 좋음

프로그램 = 명령과 데이터의 집합

자료 - 변수, 배열 등
연산 - 기본연산, 사용자정의
묶어서 자료구조...
스택 - LIFO (함수호출)
리카시브콜? recursive call

배열 - 같은 자료형 여러개 쓸 때 사용
한두개 쓸땐 그냥 변수 사용

어떤 자료를 쓸건지 - 배열, 변수 내용

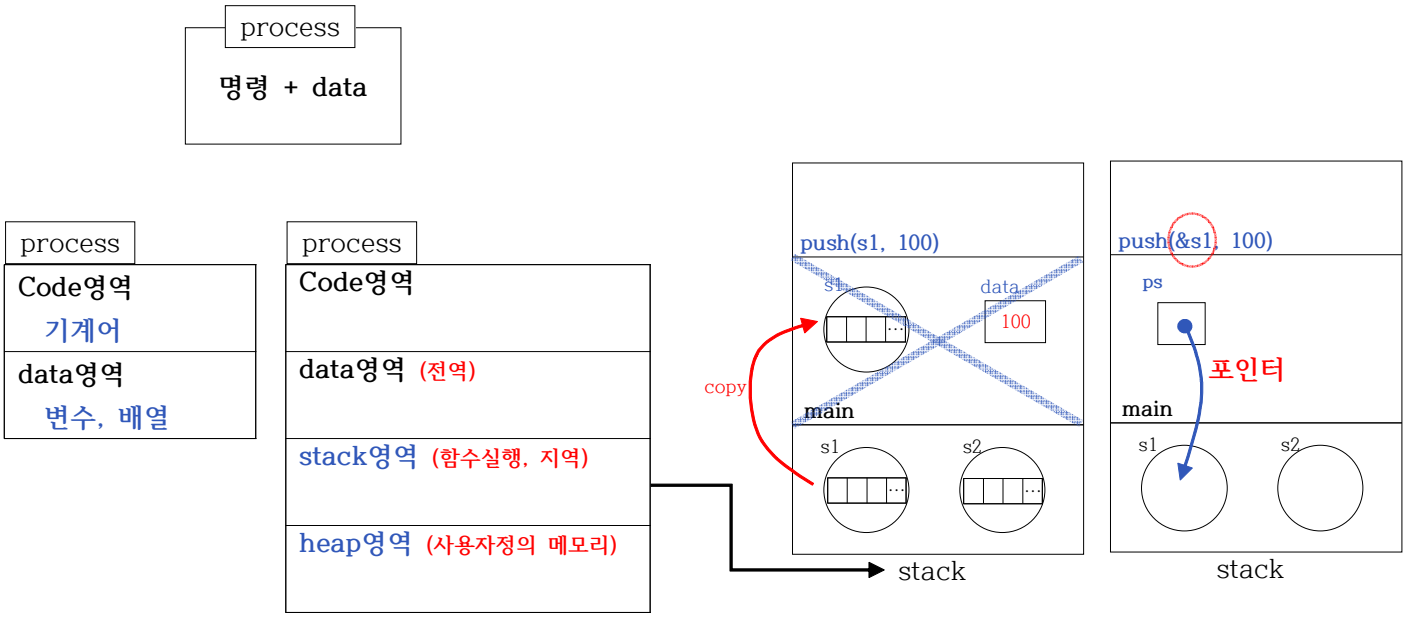
자주쓰는 언어 - 1. 자바, 2. c/c++, 3. 파이썬

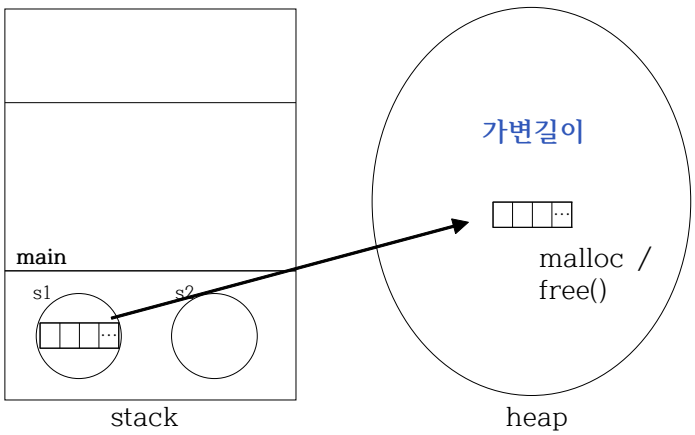
int -> -21억 ~ 21억

void 함수 입력 or 결과값이 없을 때 사용, void포인터

[Stack.c]

프로그램실행 : 주기억장치 -> 보조기억장치 ?
프로세스(실행중인 프로그램) / 서비스-백그라운드로 돌아가는 프로그램들





C언어 포괄형 프로그램(PG)
(Generic)
※ 메모리상의 모든 데이터는
(시작주소, data size)를 알면,,
모든 데이터를 다룰 수 있음

제네릭 프로그래밍(영어: generic programming)은 데이터 형식에 의존하지 않고, 하나의 값이 여러 다른 데이터 타입들을 가질 수 있는 기술에 중점을 두어 재사용성을 높일 수 있는 프로그래밍 방식이다.

제네릭 프로그래밍은 여러가지 유용한 소프트웨어 컴포넌트들을 체계적으로 융합하는 방법을 연구하는 것으로 그 목적은 알고리즘, 데이터 구조, 메모리 할당 메커니즘, 그리고 기타 여러 소프트웨어적인 장치들을 발전시켜 이들의 재사용성, 모듈화, 사용 편이성을 보다 높은 수준으로 끌어올리고자 하는 것이다.

<string.h>

- memset() , memcpy() , memcmp()

```
int a;
int b;

a = 0;
b = a;

if (a == b) { }
```

```
double d;
double f;

d = 0.0;
f = d;

if (d == f) { }
```

```
int a;
int b;

memset(&a, 0, sizeof(int));      // a = 0;
memcpy(&b, &a, sizeof(int));   // b = a;

//if (a == b) { }
if (memcmp(&a, &b, sizeof(int)) == 0) { }
```

```
double d;
double f;

memset(&d, 0, sizeof(double));     // d = 0.0;
memcpy(&f, &d, sizeof(double));   // f = d;

//if (d == f) { }
if (memcmp(&d, &f, sizeof(double))==0)
```

포인터 - 인자 전달할 때 쓰임

```
void* p;
int a;

a = 100;
p = &a;
*p = 200;
// void*는 역참조 안됨,,,
// *(int*)p  타입캐스팅 후 역참조
// a
```

void* int int*

(int)p

```
void* p;
double d;

d = 3.14;
p = &d;
*p = 2.718;
// d
```

void* double double*

(double)p