객체 지향 프로그래밍
(Object Oriented Programming)

9장

강사 – 강대기

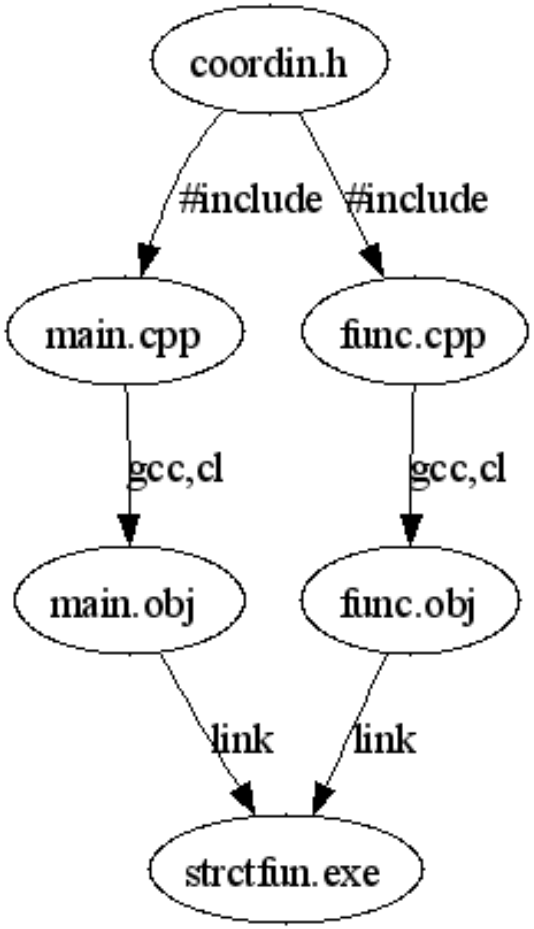
차례 (Agenda)

- 분할 컴파일
- 기억 존속 시간
- 사용 범위
- 링크
- 위치 지정 new
- 이름 공간

분할 컴파일

- Unix의 경우, make (Windows에서도 가능)
- 전체 프로그램을 분할한다.
  - 분할을 위한 좋은 방법은 아무도 모른다
  - OOP에서는 클래스로 만들어 나간다
- 함수들을 분할하고, 공통으로 필요한 부분은 헤더로 넣는다
- 실행 파일의 메모리에 들어가야 하는 것은 헤더에 넣을 수 없다. 따라서, global.c 와 같은 파일이 필요하다

분할 컴파일 예


coordin.h
#include
#include
main.cpp
func.cpp
gcc,cl
gcc,cl
main.obj
func.obj
link
link
strctfun.exe

Make

- Unix – make, gmake
- Windows – make, nmake (VC 디렉토리에 있음), gmake
- Java는? – Ant

- SE 충고 – 일일 빌드!
- 코드 공유 및 관리 도구 – CVS, VSS, SVN

Makefile의 가장 쉬운 예

all:(탭 입력)hello.c

(탭 입력)gcc -o hello hello.c

- 주의: Makefile은 반드시 탭만 넣어서 띄어야 할 부분이 있다. 공백이나 다른 건 안통함

make –f Makefile

http://kldp.org/KoreanDoc/html/GNU-Make/GNU-Make-7.html

헤더에 들어가는 것들

- 함수 프로토타입
- #define, const를 이용해 정의된 상수
- 구조체 선언
- 클래스 선언
- 템플릿 선언
- 인라인 함수

- 함수 정의나 변수 선언은 들어가선 안된다

#include 헤더에는 ifndef 를 활용

- 언제나 바보같은 상황에서 헤더를 중복하여 include 할 수 있는 위험성이 존재한다
- 이를 방어하기 위해 #ifndef, #define, #endif 로 include 하는 코드를 감싸야 한다

기억 존속 시간

- 특정 변수나 객체는 얼마나 오래 메모리에 머무는가?
- 자동(오토), 정적(스태틱), 동적(다이내믹)
- 자동 – 자동, 레지스터 – 블록이나 함수와 생성/사멸을 같이 함
- 정적 – 외부링크, 내부링크, 링크 없음 – 일단 생기면, 프로그램 끝날 때까지
- 동적 – new/delete로 알아서 생성/사멸

자동 변수

- 자동
  - 함수나 블록 안에서 int index; 또는 auto int index;
  - 초기화 안되어 있으면 쓰레기값
- 레지스터
  - 메모리 대신 레지스터에 저장하라는 요청
  - 포인터 변수로 받지 못함

정적변수

- 프로그램이 실행되는 전체 시간 동안 존속
- 이미 0이나 NULL로 초기화되어 있음
- 외부링크 – 전역 변수
- 내부링크 – 전역 변수 앞에 static
- 링크없음 – 자동 변수 앞에 static

제한자 (523쪽)

- 기억 공간 제한자
  - auto, register, static, extern, mutable
- cv 제한자
  - const , volatile
- 기억공간: const 는 #define 처럼 행동한다

함수 링크 / 언어 링크

- 하나의 함수 안에서 다른 함수 정의 안됨
- 정적 기억 존속 시간
- 외부 링크 – static 으로 내부 링크 부여 가능
- 단일 정의 규칙
- inline도 #define 처럼 행동
- 이름 장식
  - “C” – _spiff, _printf, _scanf
  - “C++” – spiff_d_d@v (이름 장식)
- Calling Convention
  - __cdecl, __stdcall, __fastcall
  - 옛날꺼 – __pascal, __fortran, __syscall

동적 저장 공간

- 동적 기억 공간 관리의 복잡도
  - new / delete 와 GC는 동일
- int* p = new int[10];
  - p 자체는 자동/정적, 가리키고 있는 것은 동적
  - 참조나 Java 는 설계에서 근본적으로 NULL 포인터 에러를 막는다 → 실제 상황은 알 수 없다

위치 지정 new

- 이미 할당된 기억 공간을 같이 사용하는 방식
- 예를 들어

```
char buffer[100];
double* pd = new (buffer) double[10];
```

- #include <new> 필요
- delete를 하면 안됨

이름 공간 (536쪽)

- 과거의 C++ 이름 공간 (534쪽)
  - 선언 영역, 잠재 사용 범위, 사용 범위
- 전역 위치에 놓이며 중첩 가능 (543쪽)
- 외부 링크
- 전역 이름 공간 – 전역 변수
- 항상 열려 있다 – 새로 추가 가능
- 이름 공간 제한자 – :: (qualified name)

using 선언과 using 지시자

- using 선언 (declaration)
  - using Jill::fetch
  - 같은 게 있으면 충돌!
  - 사용이 더 권장됨
- using 지시자 (directive)
  - using namespace Jill;
  - 같은 게 있으면 가리워짐
- 이름을 명명하지 않는 기억 공간 – 전역 변수 (545쪽)