Call Graph

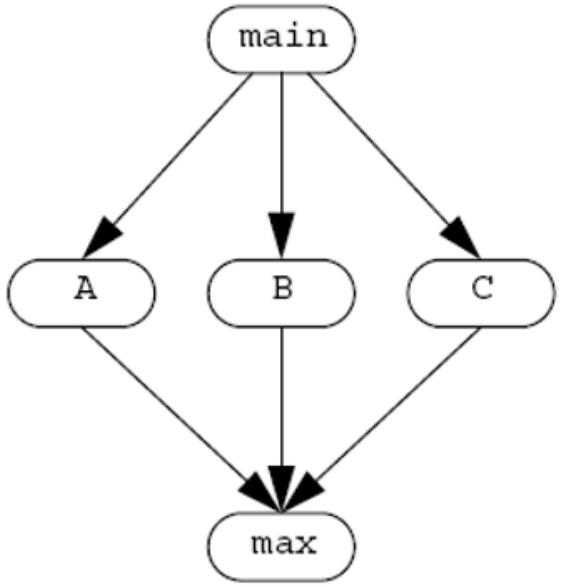
Call graph

- a directed graph that represents the calling relationships between the program's procedures

```
procedure main() {
  return A() + B() + C();
}

procedure A() {
  return max(4, 7);
}
procedure B() {
  return max(4.5, 2.5);
}
procedure C() {
  return max(3, 1);
}
```

(a) Example Program



(b) Context-Insensitive

Call graph

What is for?

- 1)human understanding of programs
- 2)Performance tuning
- 3)Design pattern detection
- 4)Other software maintenance activities,
- such as dead function detection, change impact analysis …

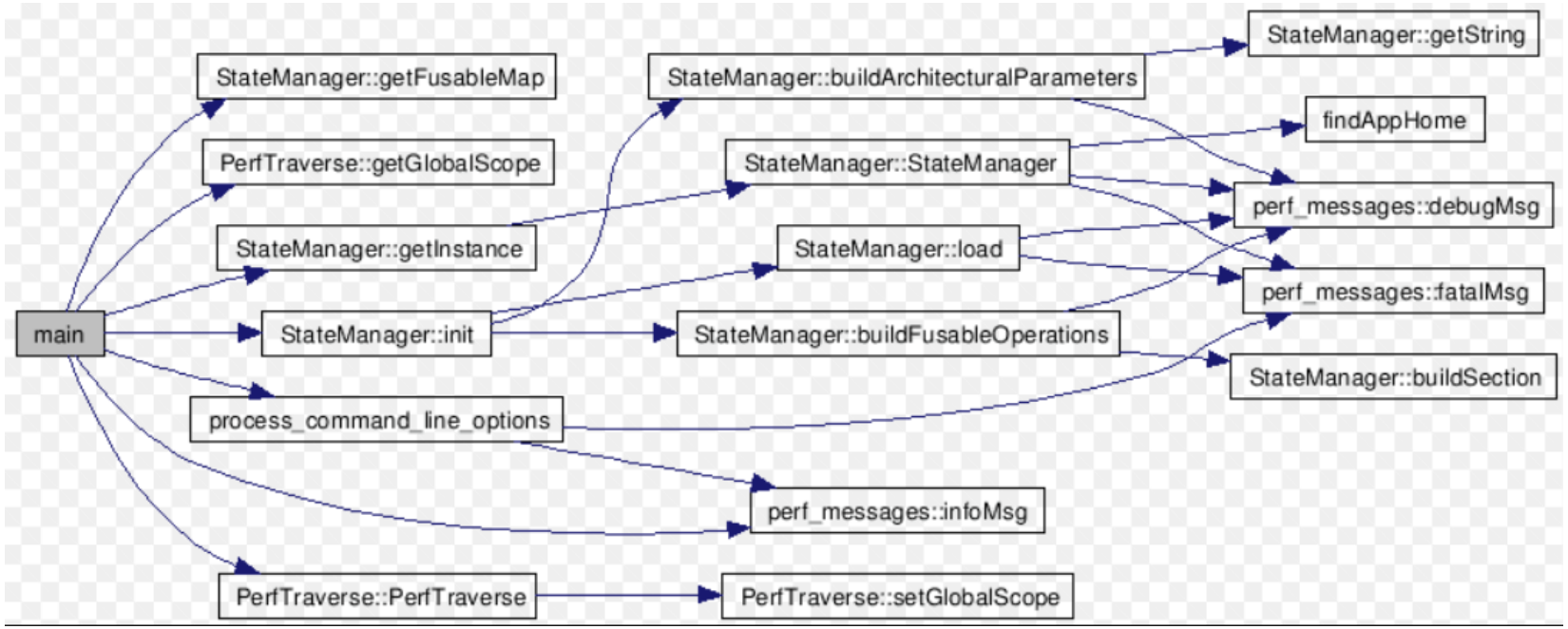
Idea for making call graph

- 어떤 문법으로 함수 선언 및 호출하는지 파악
- 해당하는 문법에서 함수 이름을 저장
- 같은 함수가 여러번 호출 된다면 함수 호출 횟수를 증가
- 원하는 자료구조에 저장 (배열, 리스트.. 등)
- 모든 것이 완료되면 graphviz 등 콜 그래프를 시각화

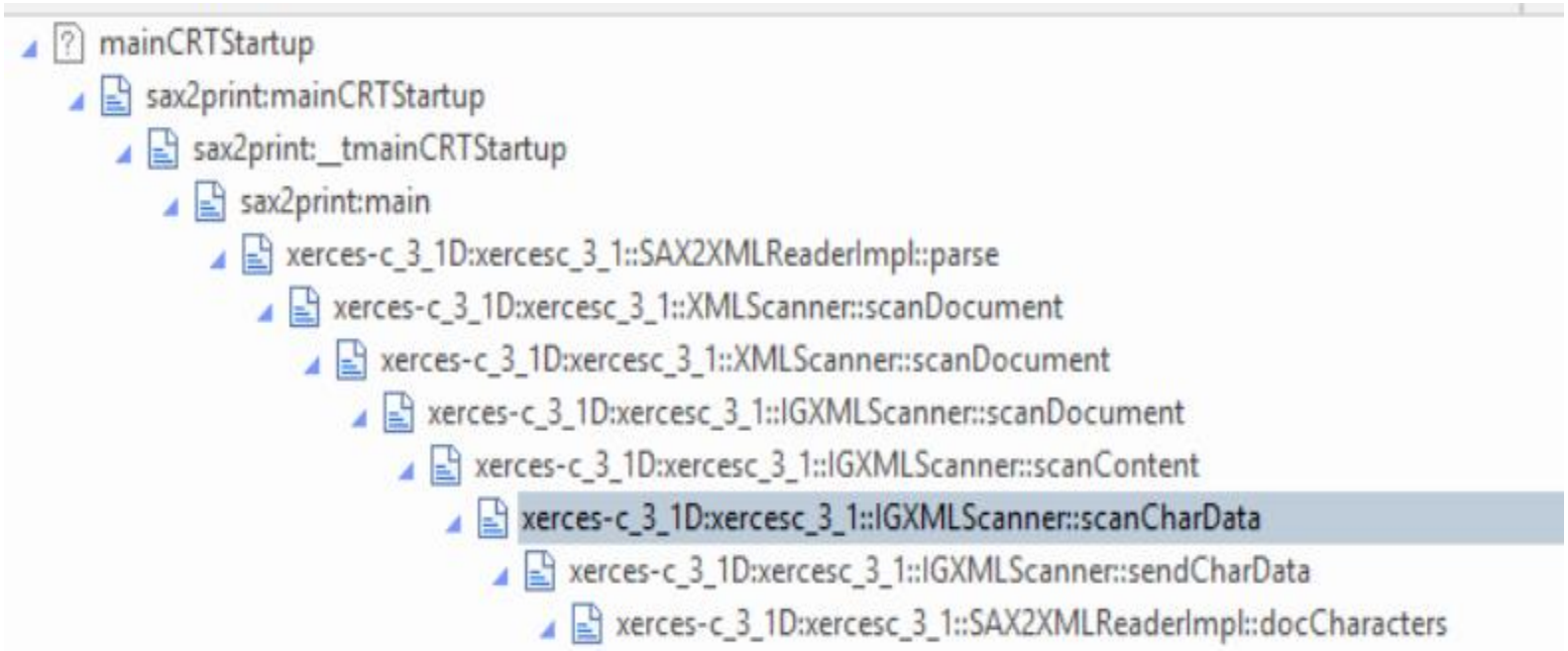
과제 간단 설명

- 렉스와 야크를 이용해 코드를 파싱
- 함수 호출과 관련된 내용 (어떤 함수가 어떤 함수를 호출하는지, 몇번 호출하는지, 코드의 몇 번째 줄에서 호출하는지 등)을 자료구조에 저장
- 자유롭게 원하는 방법(ex. C라이브러리 등)을 사용하여 자료구조에 저장된 내용을 콜 그래프로 출력
- 자세한 사항은 과제설명 파일 참고

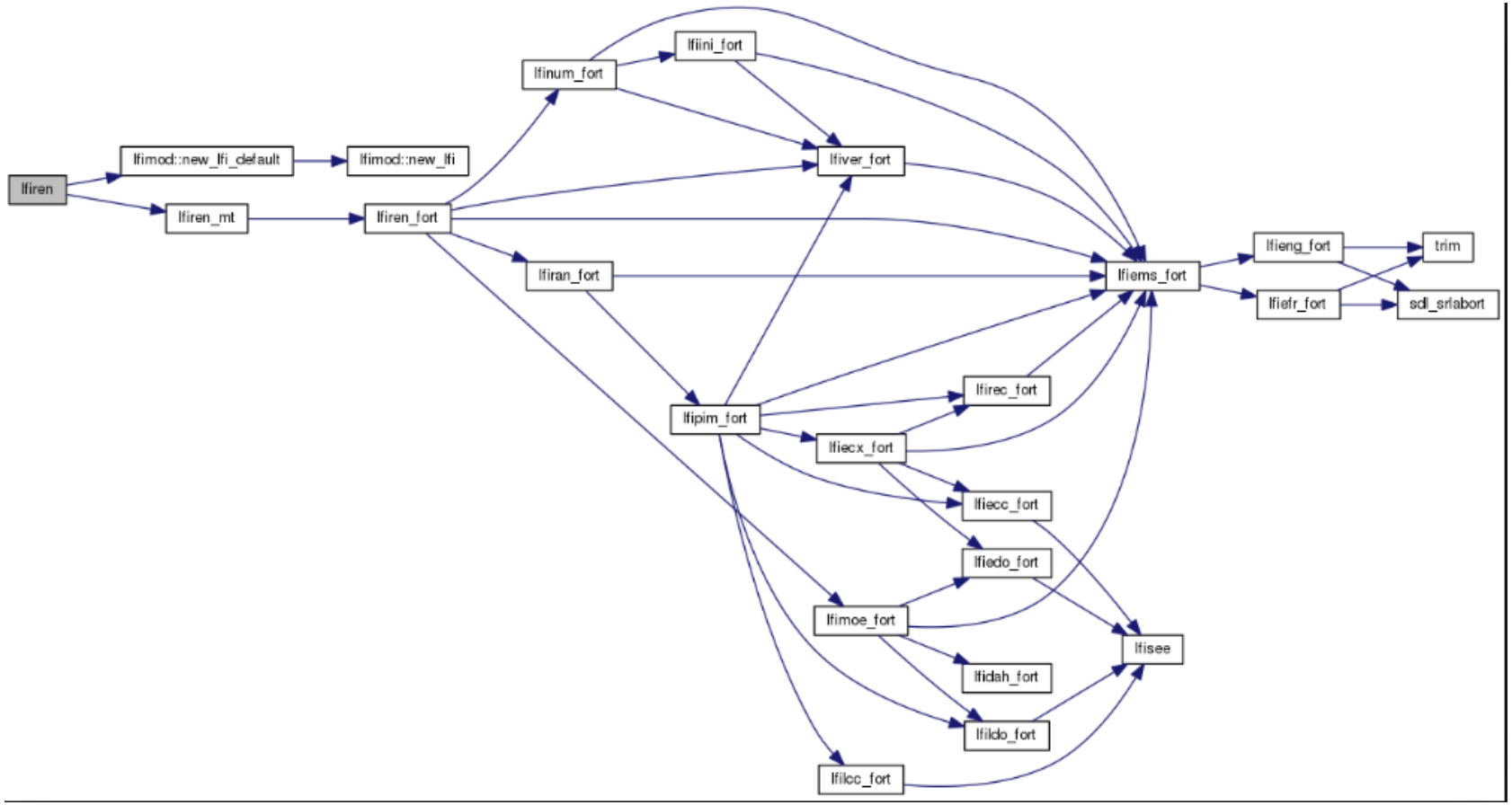
Call graph 예시



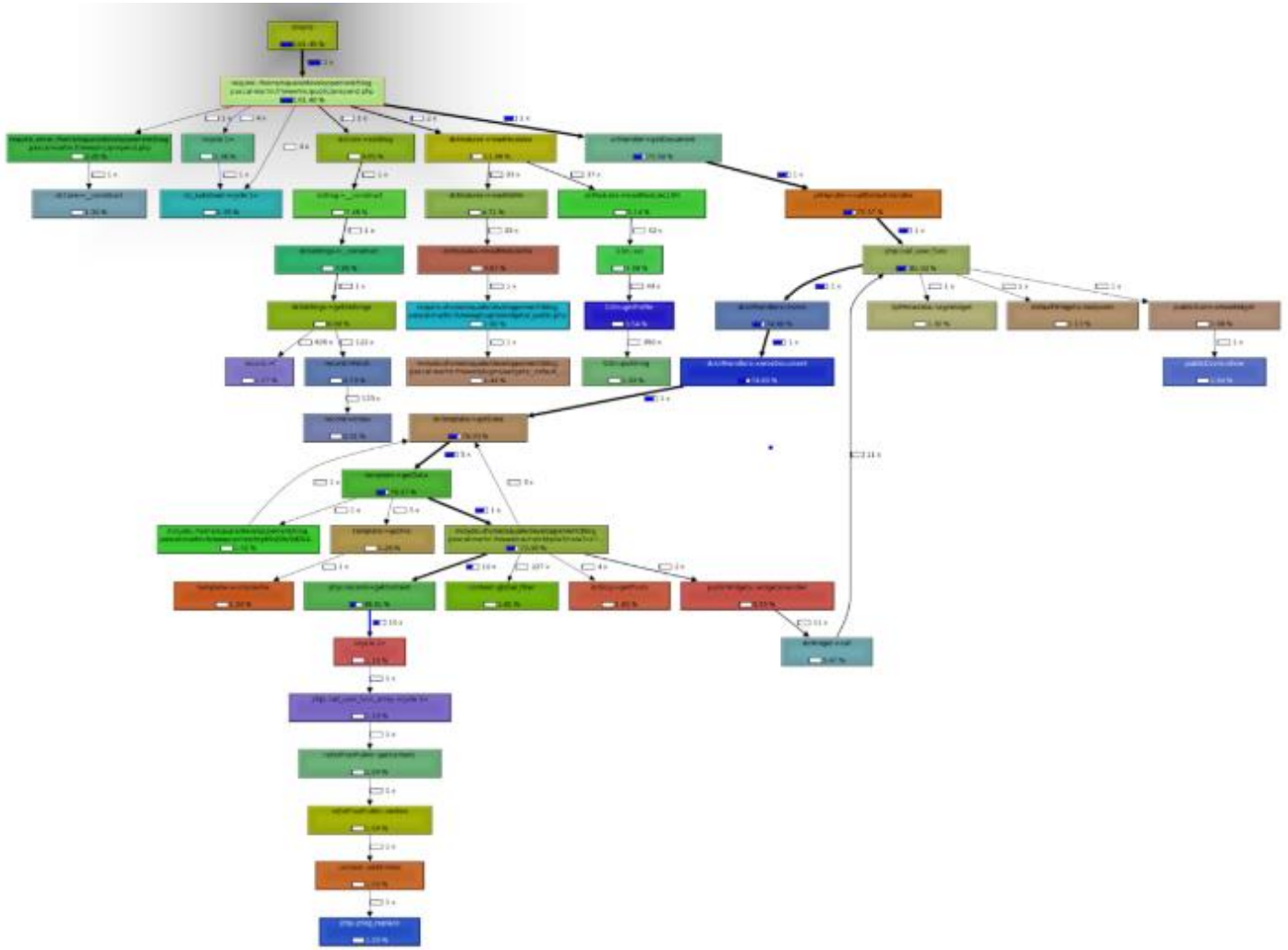
Call graph 예시



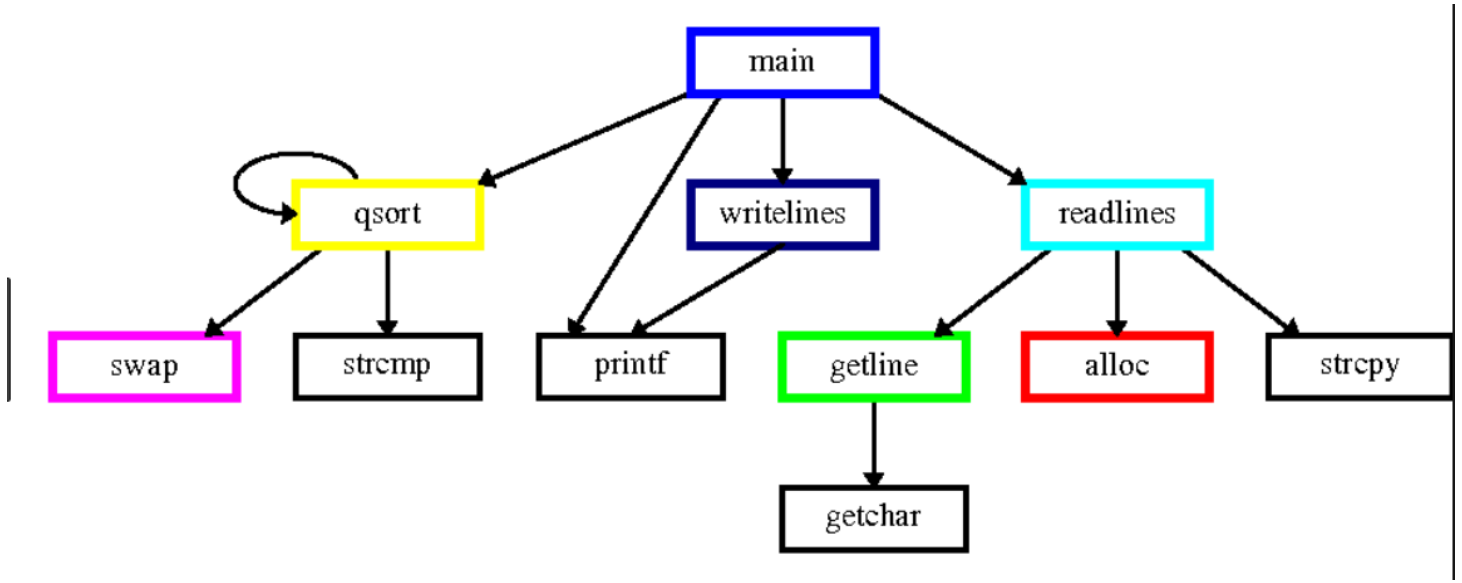
Call graph 예시



Call graph 예시



Call graph 예시



실습 설명

```c
int func(int x, int y)
{
}
int main()
{
        int c;
        int d=4;
        func();
        aa();
        aa();
        bb(3);
        cc(11,111,222);
}
```

func.c

실습 결과

```
[          @localhost practice]$ ls
ex  ex.l  ex.y  func.c  lex.yy.c  y.tab.c  y.tab.h
[          @localhost practice]$ ./ex < func.c
main -> func
main -> aa
main -> aa
main -> bb
main -> cc
```

분석 (ex.l)

```
%{
#include <stdio.h>
#include "y.tab.h"
extern char name[10];
%}

_
D       [0-9]
L       [a-zA-Z_]

%%
```

입출력을 위한 stdio.h

y.tab.h 를 참조 (yacc 명령으로 생성)

ex.y 에 있는 name

ex.l

yytext 는 파싱중 lexer가 읽고 있는 위치

```
%%
"int"    {return INT;}
{L}({L}|{D})*    { strcpy(name,yytext); return IDENTIFIER; }
{D}+     {return NUMBER;}
","      {return ',';}
"{"      {return '{';}
"}"      {return '}';}
"("      {return '(';}
")"      {return ')';}

[ \t\v\n\f]      {}
.        {return yytext[0];}
%%
```

Identifier 는 어디에??

y.tab.h

```
[       @localhost practice]$ cat y.tab.h
#define IDENTIFIER 257
#define NUMBER 258
#define INT 259
```

Apple 을 읽고 있다면 yytext[0] = A

ex.l

```
%%

int yywrap(void)
{
        return 1;
}
```

파싱 끝날때 호출(eof 만나면) 반드시 1리턴해야함

만약 파서가 " func" (IDENTIFIER) 라는 이름을 넘겨 받았다면? (이미 int는 넘겨 받은 상태) 어떻게 파싱될까??

```
int func(int x, int y)
{
}
int main()
{
        int c;
        int d=4;
        func();
        aa();
        aa();
        bb(3);
        cc(11,111,222);
}
```

```
        ;
declarator
        : IDENTIFIER
        ;
initializer
        : IDENTIFIER
```

```
        ;
primary_expression
        : IDENTIFIER {}
        | NUMBER
        ;
```

```
parameter
        : INT IDENTIFIER
        ;
```

```
func_name
        : IDENTIFIER {strcpy(func, name);}
```

결국 여기로 파싱

```
function
        : INT func_name declarator compound_expression
        ;
```

```
func_name
        : IDENTIFIER {strcpy(func, name);}
```

다음의 코드를 분석해볼 것

```
statement
        : expression ';'
        {
                if(checkfunc == 1)
                {
                        strcpy(call[i], name);
                        i++;
                }
                checkfunc=0;
        }
```

```
postfix_expression
        : primary_expression
        | postfix_expression '(' ')' {checkfunc=1;}
        | postfix_expression '(' argument_expression_list ')' {checkfunc=1;}
```

Makefile

- hw5 : y.tab.c lex.yy.c
- 	cc y.tab.c lex.yy.c -o hw5
- y.tab.c : hw5.y
- 	yacc -d hw5.y
- lex.yy.c : hw5.l
- 	lex hw5.l

사용법은 “make” 입력하면 끝

dot명령어를 이용한 jpg 출력

- https://graphviz.org/pdf/dot.1.pdf
- http://www.graphviz.org/pdf/dotguide.pdf

도큐 참고. .gv파일을 만들고 정해진 형식에 맞게 작성.

dot -Tjpg ~~.gv -o ~~.jpg

Q & A