제8장 함수

8.1 함수 정의 및 호출
8.2 매개변수 전달
8.3 함수와 바인딩
8.4 함수의 타입 검사
8.5 함수 구현

8.1 함수 정의 및 호출

프로시저/함수 + 메서드(Method)

↳ 객체지향언어에서 사용하는 말

- 프로시저(Procedure)
  - 한 그룹의 계산과정을 추상화하는 메커니즘으로
  - 반환 값이 없으며
  - 매개변수나 비지역 변수를 변경한다.

- 함수(Function)
  - 반환 값이 있으므로
  - 식에 나타날 수 있고
  - 매개변수나 비지역 변수 값 변경은 선택사항

언어 S: 함수 확장

<command> → <decl> | <stmt> | <function>

<stmt> → ...

| return <expr>; // 리턴문

| id(<expr> {, <expr>}); // 함수 호출

↳ 매개변수 개수만큼의 arguments

<function> → fun <type> id(<params>) <stmt> //함수 정의

함수의 body

<params> → <type> id {,<type> id} → 매개변수선언

<type> → int | bool | string | void

<factor> → ...

| id(<expr> {, <expr>}) // 함수 호출

① 결과값이있는경우 하나의 factor로사용가능

② 호출만하면 statement의 역할(아무것도하지않고)

함수 정의

- [예제 1] in C

```
int max (int x, int y)
{
    return x > y ? x : y;
}
```

[예제 1] in S

```
fun int max (int x, int y)
    if x > y then return x;
    else return y;
```

- [예제 2]

```
>> int a = 10;
>> int b = 20;
>> fun int max(int x, int y)        // 함수 정의
       if (x > y) then return x;
       else return y;
>> print max(a,b);                  // 함수 호출
```

변수선언 1

2

3

Command!

프로시저 정의

- [예제 3] in S

```
fun void swap(int x, int y)
let int temp = x; in
    x = y;
    y = temp;
end;
```

- 호출 예

```
swap(a, b);
```

[예제 3] in C

```
void swap(int x, int y)
{  int temp = x;
   x = y;
   y = temp;
}
```

타입 없는 함수 정의

- Lisp/Scheme, JavaScript, Python
  - 변수의 타입을 선언하지 않고 바로 사용할 수 있다.
  - 함수를 정의할 때도 타입을 선언하지 않는다.

- Python 예

```
def 함수이름(매개변수):
    함수본체
```

- [예제 4]

```
>> def salePrice(price, percent):
        result = price * (1 – percent/100)
        return result
>> salePrice(48000, 30)
33600.0
```

[예제 5]

```
>> def salePrice(price, percent)
        result = price * (1 – percent/100)
        print("할인 가격:", result)
>> salePrice(48000, 30)
할인 가격: 33600.0
```

→ return 대신

사례 연구

- C, C++, Java, Python, S
  - 오직 함수만 있고
  - 프로시저는 반환 타입이 void 이다.
- Ada
  - procedure 와 function를 구분한다.
- Modula-2
  - 모두 프로시저이다.
  - 함수는 반환 값이 있는 프로시저이다.

8.2 매개변수 전달

매개변수 전달

- 함수 정의와 함수 호출
  - <function> → fun <type> id(<params>) <stmt> // 함수 정의
  - <params> → <type> id {,<type> id} // 형식 매개변수
  - id(<expr> {, <expr>}); // 함수 호출

- 형식 매개변수(formal parameter)
  - 선언에서 사용된 매개변수

- 실 매개변수(actual parameter)
  - 호출에서 사용된 매개변수로 인자(argument)라고도 한다.

- 매개변수 전달
  - 호출 시에 실 매개변수와 형식 매개변수를 매칭하는 것

매개변수 전달 방법

- 값 전달 pass by value
- 참조 전달 pass by reference
- 값-결과 전달 pass by value-result
- 이름 전달 pass by name

값 전달(pass by value)

- 값 전달 과정
  - 1. 식(expression)인 실 매개변수 값들을 계산한다.
  - 2. 계산된 값들을 대응되는 형식 매개변수에 전달 (복사 혹은 초기화)한다.
  - 3. 본체를 실행한다.

인자(arguments)

- 호출 예
  - a =2; b = 3;
  - max(2 * a, a + b);

int max (int x, int y)

  - x = 4; y = 5; // 값 전달
  - if (x > y) then return x
    else return y // 본체 실행 및 반환

값 전달의 문제점 ?

```
>> int a = 10;
>> int b = 20;
>> swap(a,b);
```

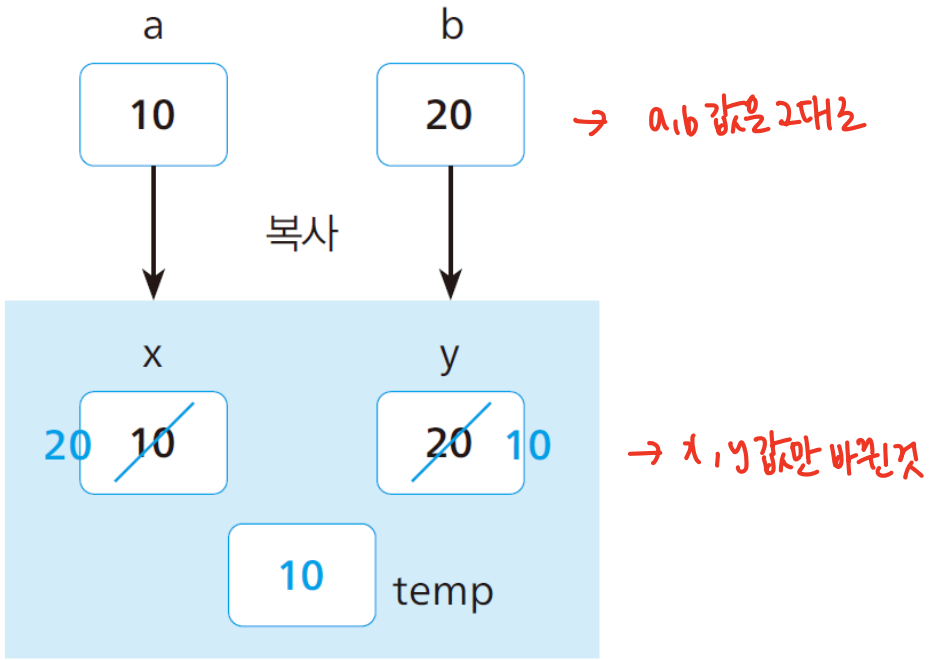
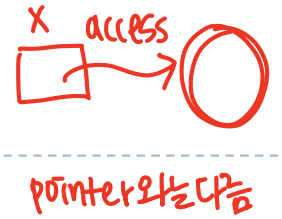


그림 8.1 값 전달

⇒ 값전달은 swap 불가능!

참조 전달



- 참조 전달 방법
  - 실 매개변수의 위치(주소)가 계산되어 형식 매개변수에 전달된다.
  - 따라서 실 매개변수는 할당된 위치가 있는 변수이어야 한다.

- 형식 매개변수 사용
  - 자동 주소 참조(dereferencing)가 이루어져 실 매개변수를 접근
  - 형식 매개변수를 변경하면 자동적으로 실 매개변수가 변경된다.

- 실 매개변수와 형식 매개변수의 이명(aliasing)

참조 전달 예(C++)

- [예제 6]

```
void swap(int& x, int& y)      // 헤더
{                              // 본체
    int t = x;
    x = y;
    y = t;
}
```

자동주소참조

- 사용

```
int a = 10;
int b = 20;
swap(a,b);
```

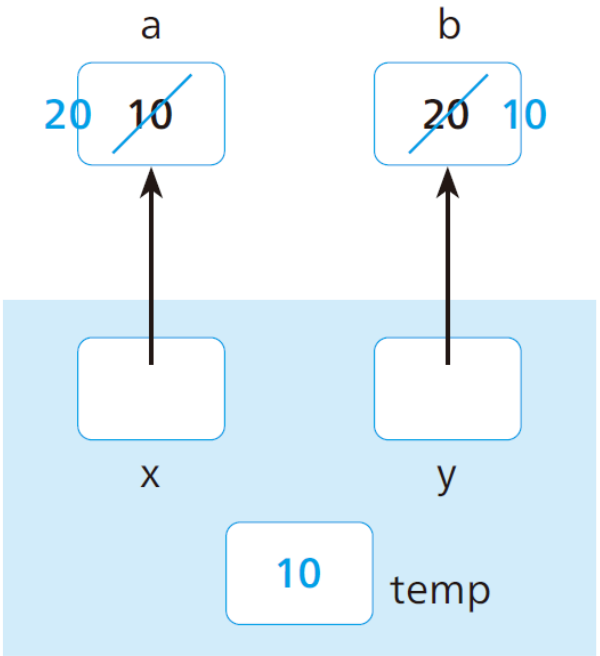


그림 8.2 참조 전달

C 언어에서 참조 전달 효과 내기

- 기본 아이디어
  - C는 원칙적으로 값 전달만 제공하나 포인터 타입이 있다.
  - 참조전달 효과
    - 포인터 값(위치/주소)을 명시적으로 전달하여 낼 수 있다.
  - 주소 참조(dereferencing)
    - 함수 내에서 주소참조는 프로그래머가 명시적으로 해야 한다.

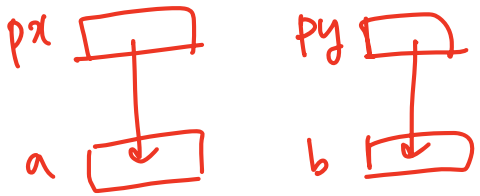
- 사실상 프로그래머가 참조전달을 구현한다.

예제(C 언어)

- [예제 7] intswap의 C 버전

```
void swap(int *px; int *py)
{
    int t;
    t = *px;
    *px = *py;
    *py = t;
}
```

주소를 따라가라고 표시해줌



- 사용

```
int a = 10, b = 20;
swap(&a, &b);
```

예제(C++)

- [예제 8]

```
int a = 1;
void ack(int& x, int& y)
{
    x = 2; ✓
    a = 3;
    y = 4; ✓
}
  . . .
ack(a, a); ⇒ a는 2→3→4로 변화, 결과는 a는 4
```

- 참조 전달의 문제점 ?

값-결과 전달

- 기본 아이디어
  - 호출할 때
    - 실 매개변수 값은 형식 매개변수에 전달(복사) 한다.
  - 반환할 때
    - 형식 매개변수 값을 실 매개변수에 역으로 전달(복사) 한다.
  - copy-in, copy-out이라고도 한다.

- 참조 전달과 뭐가 다른가 ?

값-결과 전달: 예제

- [예제 9]

```
void swap(int x, int y)          // 헤더
{                                // 본체
    int t = x;
    x = y;
    y = t;
}
```

명시하지 않으면 값전달이랑 어떻게 구분?

- 사용

```
int a = 10, b = 20;
swap(a, b);
```

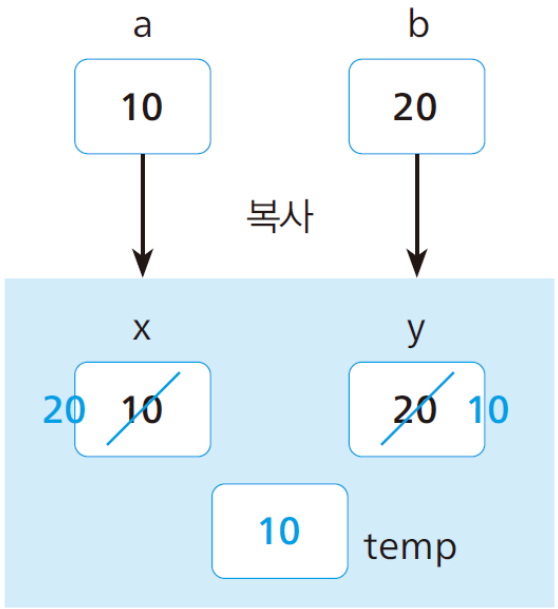


그림 8.3 값 전달

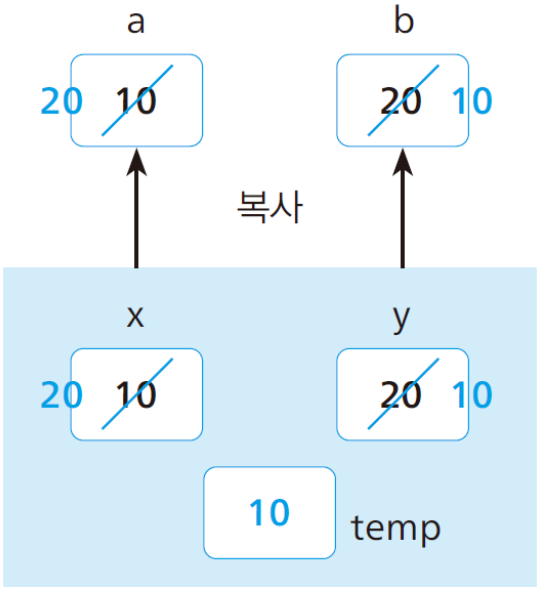


그림 8.4 결과 전달

[연습문제 1]

- 함수 p를 호출할 때 호출 후의 a의 값은 무엇인가?
  - 값 전달, 참조 전달, 값-결과 전달

```
void p(int x, int y)
{
        x++;
        y++;
}

main()
{
        int a = 1;
        p(a,a);
        ...
}
```

(1) 값전달 : 1 (그대로)
(2) 참조전달 : 3
(3) 값-결과전달 : 2

이름 전달

- 기본 아이디어
  - 게으른 계산 !
  - 사용될 때까지 버틴다.
- 방법 1
  - 실 매개변수는 대응 형식 매개변수가 사용될 때까지 계산되지 않고
  - 형식 매개변수가 사용될 때 비로소 계산된다.
- 방법 2
  - 형식 매개변수를 실 매개변수 이름으로 대치하고 실행한다고 생각할 수 있다.
- 함수형 언어의 지연 계산(delayed evaluation)에서 사용된다.

이름 전달: 예제

- [예제 10]

```
int  i;
int a[10];

void p(int x)
{  i = i + 1;
   x = x + 1;
}

main()
{  i = 1;
   a[1] = 10;
   a[2] = 20;
   p(a[i]);
}
```

a[i] = a[i] + 1;
↳ 액세스하면 i가 2가 됨

이름 전달: 예제

- 선언

```
void swap(int x, int y)          // 헤더
{                                // 본체
    int t = x;
    x = y;
    y = t;
}
```

- 사용

```
swap(i, a[i]);
```

사례 연구

- Ada
  - *in* 매개변수: 값 전달
  - *in out* 매개변수: 값-결과 전달
- C
  - 값 전달
  - 포인터를 이용한 참조 전달 효과
- C++, Pascal, Modula-2
  - 값 전달
  - 참조 전달
- Java
  - 기초 타입(primitive type)은 값 전달
  - 객체는 참조 전달 (자동으로)
- FORTRAN, Python
  - 참조 전달, Python은 객체 참조 전달

8.3 함수와 바인딩

함수와 유효범위 규칙

- 선언된 이름(식별자)의 유효범위(scope)
  - 선언된 변수 혹은 함수 이름이 유효한(사용 가능한) 프로그램의 범위 혹은 영역
- 정적 유효범위(static scope) 규칙
  - 선언된 이름은 선언된 블록 내에서만 유효함
  - 대부분 언어에서 표준 규칙으로 사용됨
- 동적 유효범위(dynamic scope)
  - 선언된 이름은 선언된 블록의 실행이 끝날 때까지 유효함
  - 실행 경로에 따라 유효범위가 달라질 수 있다.
  - Old Lisp와 SNOBOL에서 사용됨

예제

[예제 12]

```
1 int x = 0;
2 fun void g(int y)
3     x = x + y * y;
4 let int x = 1; in
5     g(5);
6 end;
```

- 정적 유효범위 규칙
- 동적 유효범위 규칙

[예제 13]

```
1 int x = 0;
2 fun void g(int y)
3       x = x + y * y;
4 fun void f(int z)
5 let int x = 1; in
6       g(z);
7 end;
8 f(5);
```

함수와 바인딩 → 이름이뭔지결정하는것!

- 수식 혹은 문장의 의미 파악
  - 유효한 변수/함수에 대한 속성 또는 바인딩 정보가 필요하다.

- 변수의 속성
  - 변수의 선언된 타입
  - 변수 이름의 유효범위
  - 변수의 값 혹은 위치

- 함수의 속성
  - 함수의 선언된 타입
  - 함수 이름의 유효범위
  - 함수의 코드 위치

[예제 11]

```
>> int y;
>> fun int square(int x) return x * x;
>> y = square(5);
```

바인딩

- 일반적으로 바인딩(binding)
  - 이름을 어떤 속성과 연관(association) 짓는 것을 말하며,
  - 보통 변수, 상수, 함수 등의 이름(식별자)을 속성과 연관 짓는 것을 말한다.

- 바인딩 사례
  - 이름 상수의 실제 상수 값을 결정하는 것
  - 변수 또는 함수의 유효범위 또는 타입을 결정하는 것
  - 변수가 메모리에 적재될 때 기억장소의 주소를 정하는 것
  - 연산 기호(예를 들어 '*')가 나타내는 실제 연산('곱셈')을 정하는 것
  - 함수 호출과 호출된 함수를 연관 짓는 것

바인딩

- 정적 바인딩
  - 컴파일 시에 한번 이루어지고 실행 동안 변하지 않고 유지된다.
  - 정적 바인딩되는 속성은 정적 속성이라고 한다.

- 동적 바인딩
  - 실행 중에 이루어지는 바인딩으로 실행 중에 속성이 변경될 수 있음
  - 동적 바인딩되는 속성은 동적 속성이라고 한다.

바인딩 정보 관리

- 유효한 이름(식별자)
  - 프로그램 각 지점에서 유효한 변수, 함수 이름들은 다르다.
  - 프로그램 각 지점에서 유효한 이름들에 대한 바인딩 정보를 유지 관리해야 한다.

- 컴파일러/인터프리터
  - 프로그램을 한 줄씩 읽으면서 번역/해석한다.
  - 프로그램의 각 지점에서 유효한 이름들에 대한 바인딩 정보를 이용하여 번역/해석한다.

바인딩과 심볼 테이블

- 심볼 테이블(Symbol Table)
  - 유효한 바인딩 정보를 유지/관리하기 위한 자료 구조
  - 환경(environment)이라고도 한다.

- 컴파일러
  - 심볼 테이블에 컴파일 하기 위해 필요한
  - 변수 또는 함수의 바인딩 정보 유지 관리
  - 타입 검사기 TypeChecker.java
    - 심볼 테이블에 변수 또는 함수의 타입 정보 유지 관리

- 인터프리터
  - 심볼 테이블에 해석해서 실행하기 위해 필요한
  - 변수의 값, 함수의 코드 위치 등 유지 관리

심볼 테이블

- **[예제 14]**

```
>> int y;
>> fun int square(int x) return x * x;
>> y = square(5);
```

식별자	타입	유효범위	값
y	int	전역	25
square	함수 int->int	전역	square의 AST
x	int	지역(square)	5

8.4 함수의 타입 검사

타입 규칙: 함수 정의

- [예제 15]

```
fun bool f(int x)
    if (x > 0) then return 1;
    else return 0;
```

- [예제 16]

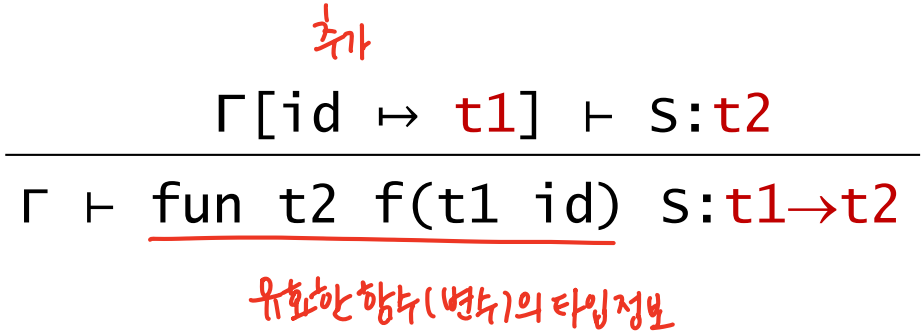
```
>> fun int square(int x) return x * x;
>> int y;
>> y = square(5);
```

- 함수의 타입

  square: int → int

타입 규칙: 함수 정의

- 함수 정의의 타입 규칙
  - 함수 헤더에 선언된 것처럼 함수가 정의됨을 확인하는 규칙
  - 즉 매개변수 id가 t1 타입일 때 함수 본체 s가 t2 타입이어야 한다.
  - 이를 만족하면 이 함수의 타입은 t1 → t2가 된다.



타입 검사: 함수 정의

- 함수 square의 타입은?

  fun int square(int x) return x * x;

- 타입 검사

$$\{\ \} \vdash \text{fun int square(int x) return x * x} : \text{int} \rightarrow \text{int}$$

$$\Big|$$

$$\{x \mapsto \text{int}\} \vdash \text{return x * x} : \text{int}$$

$$\Big|$$

$$\{x \mapsto \text{int}\} \vdash \text{x * x} : \text{int}$$

타입 검사: 함수 정의 오류

- [예제 15]

```
fun bool f(int x)
    if (x > 0) then return 1;
    else return 0;
```

- 타입 검사 과정

$$\{\ \} \vdash \text{fun bool f(int x) if (x>0) ... : error}$$

$$\Big|$$

$$\{x \mapsto \text{int}\} \vdash \text{if (x>0) then return 1; else return 0 : int}$$

$$/ \qquad \Big| \qquad \backslash$$

$$\{x \mapsto \text{int}\} \vdash (x>0): \text{bool} \quad \{x \mapsto \text{int}\} \vdash \text{return 1:int} \quad \{x \mapsto \text{int}\} \vdash \text{return 0: int}$$

타입 규칙: 재귀 함수 정의

- 재귀 함수 정의의 타입 규칙

$$\frac{\Gamma[f \mapsto t1 \to t2,\ id \mapsto t1] \vdash s : t2}{\Gamma \vdash \texttt{fun}\ t2\ f(t1\ id)\ s : t1 \to t2}$$

- [예제 17]

```
fun int fact(int n)
    if (n==1) then return 1;
    else return n * fact(n-1);
```

Γ = {fact ↦ int → int, n ↦ int}

$$\{\} \vdash \text{fun int fact(int n) if (n==1) ... : int} \to \text{int}$$

$$\Gamma \vdash \text{if (n==1) then return 1; else return n * fact(n-1) : int}$$

$$\Gamma \vdash \text{(n==1): bool} \qquad \Gamma \vdash \text{return 1: int} \qquad \Gamma \vdash \text{return n * fact(n-1) : int}$$

$$\Gamma \vdash \text{n:int} \qquad \Gamma \vdash \text{fact(n-1) : int}$$

타입 규칙: 함수 호출

- 함수 호출의 타입 규칙
  - f가 유효한 함수 이름이어야 한다(함수 f의 타입 확인).
  - 인자 타입은 매개변수 타입과 같아야 한다.
  - 호출 결과 타입 = 함수의 f의 반환 타입

$$\frac{\Gamma(f) = t1 \rightarrow t2 \quad \Gamma \vdash E : t1}{\Gamma \vdash f(E) : t2}$$

타입 검사: 함수 호출

- 함수 호출 square(5)의 타입은?

```
>> fun int square(int x) return x * x;
>> int y;
>> y = square(5);
```

- 타입 검사 square(5)

$\Gamma = \{y \mapsto int\ ,\ square \mapsto int \rightarrow int\}$

$$\frac{\Gamma(square) = int \rightarrow int \qquad \Gamma \vdash 5 : int}{\Gamma \vdash square(5) : int}$$

fact(n-1)

$\Gamma = \{fact \mapsto int \rightarrow int,\ n \mapsto int\}$

$$\frac{\Gamma(fact) = int \rightarrow int \qquad \Gamma \vdash n\text{-}1 : int}{\Gamma \vdash fact(n\text{-}1) : int}$$

8.5 함수 구현

함수 정의 파싱

- 함수 정의 구문법

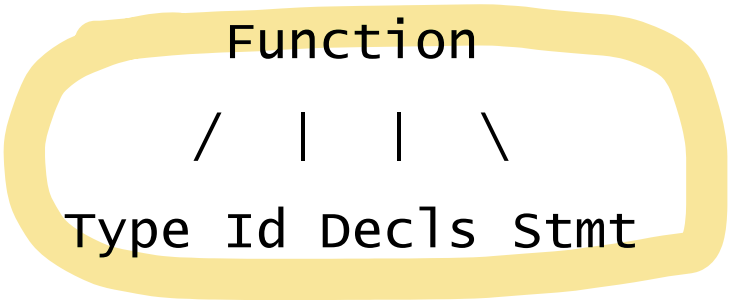
<function> → fun <type> id(<params>) <stmt>
<params> → <type> id {,<type> id}

```
private Function function () {
    match(Token.FUN);
    Type t = type( );
    String str = match(Token.ID);
    funId = str;
    Function f = new Function(str, t);
    match(Token.LPAREN);
    f.params = params( );
    match(Token.RPAREN);
    Stmt s = stmt( );
    f.stmt = s;
    return f;
}
```

함수 정의의 AST

- Function의 AST

  Function = Type type; Identifier id; Declarations params; Statement stmt



- class Function extends Command {
  Identifier id;
  Decls params;
  Stmt stmt;
  Function (String s, Type t) {
  id = new Identifier(s); type = t; params = null; stmt = null;
  }
  }

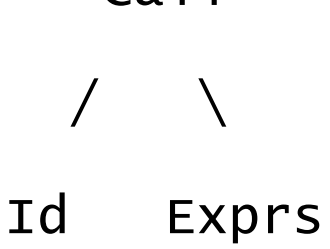
함수 호출 파싱

- 함수 호출의 구문법

  id(<expr> {, <expr>});

- 함수 호출의 AST

  Call = Identifier id; Exprs args;



- 함수 호출 파싱

```
private Call call(Identifier id) {
    match(Token.LPAREN);
    Call c = new Call(id, arguments());
    match(Token.RPAREN);
    match(Token.SEMICOLON);
    return c;
}
```

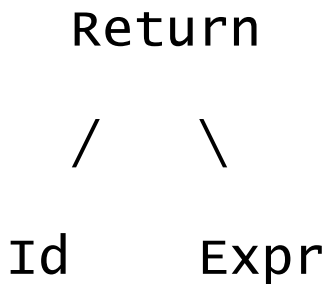
리턴문 파싱

- 반환문의 구문법

  return <expr>;

- 반환문의 AST

  Return = Identifier id; Expr expr;



- private Return returnStatement() {

  match(Token.RETURN);

  Expr e = expr();

  match(Token.SEMICOLON);

  return new Return(funId, e);

  }

타입 검사 구현 : 함수 정의

```
public static Type Check(Function f, TypeEnv te) {
    te.push(f.id, new ProtoType(f.type, f.params));
    for (Decl d : f.params)
        te.push (d.id, d.type);
    Type t = Check(f.stmt, te);          // 함수 본체 타입 검사
    if (t != f.type)                     // 리턴 타입과 비교
        error(f, "incorrect return type");
    for (Decl d : f.params)
        te.pop();
    te.pop();                            // 함수 타입 제거
    // 타입 검사된 함수 타입 추가
    te.push(f.id, new ProtoType(f.type, f.params));
}
```

fun <type> id (<params>)
<stmt>

교수님자료랑 다름...

타입 검사 과정 : 함수정의

```
public static Type Check(Function f, TypeEnv te) {
    te.push(f.id, new Prototype(f.type, f.params));
    for (Decl d : f.params)
        te.push(d.id, d.type);
    Type t = Check(f.stmt, te); → 함수본체타입검사
    for(Decl d : f.params)
        te.pop();
    te.pop(); → 함수타입제거
    if(t == f.type) { → 리턴타입과 비교
        te.push(f.id, new ProtoType(f.type, f.params)); → 타입검사된 함수타입 추가
        return f.type;
    } else { → 타입오류
        error(f, "incorrect return type");
        return Type.ERROR;
    }
```

타입 검사 구현: 함수 호출

```
static Type Check(Call c, TypeEnv te) {
    if (te.contains(c.fid)) {
        error(c, "undefined function: " + c.fid);
        return c.type;
    }
    Exprs args = c.args;
    ProtoType p = (ProtoType)te.get(c.fid);
    c.type = p.result;
    if (args.size() == p.params.size()) {
        for (int i=0; i<args.size(); i++) { // 인수와 매개변수 비교
        Expr e = (Expr) args.get(i);
        Type t1 = Check(e, te);
        Type t2 = ((Decl) p.params.get(i)).type;
        if (t1 != t2)
            error(c, "argument type does not match parameter");
    }
    ...
    return c.type;
}
```