충남대학교 프로그래밍언어론 기말고사 2012년 6월 13일 오후 6:30~

1. 분반명 (또는 월요일반인지 목요일반인지)를 적고, 학번과 이름을 적으시오.

2. (30점) 다음 중 맞는 것에는 O 틀린 것에는 X 를 하시오.

(1) JIT 컴파일러가 프로그램을 중간언어로 변환하는 시점은 해당 부프로그램이 호출될 때이다. X

(2) 배열 index가 항상 최대범위를 넘지 않도록 검사하는 것은 readability를 높이는 데에 그 목적이 있다. X

(3) Parsing tree는 abstract syntax tree를 non-terminal symbol들을 제거하고 terminal과 의미 있는 node로만 이루어진 표현으로 간략하게 바꾼 것이다. X

(4) EBNF로 표현한 규칙 `S→A(a|b|c)B` 으로부터 derivation 되는 문자열은 BNF 규칙 집합 `S→AaB`, `S→AbB`, `S→AcB` 에서도 derivation 된다. O

(5) `void f() { int x; ... }`의 변수 `x`의 주소는 수행 시간 동안 늘 동일하다. X

(6) Dynamic scoping rule에서 subprogram의 중첩이 존재한다고 가정한다면 main() 속에 중첩되어 정의된 `f()`는 은폐된(hidden) 변수를 제외한 `main()`의 모든 지역변수를 접근할 수 있다. X

(7) C의 string 은 제한적 가변길이를 지원하므로 일반 가변길이 string보다 메모리 사용이 효율적이다. X

(8) 다음과 같은 C 변수 `a`와 `b` 가 있다고 가정하자.

```
struct  A {
    int f;
} a;
int b[1];
```

만일 `int i=0;`와 같은 선언이 추가되었을 때, `a.f` 보다 `b[i]`를 접근하는 속도가 더 빠르게 된다. X

(9) 다음과 같은 코드에서 마지막 줄의 strcpy 함수 호출 시 C는 컴파일 오류를 낸다. X

```
union number {
  int value;
  char data[4];
} x;
x.value = 100; ...
strcpy(x.data, "abc");
```

(10) 다음 코드는 분실된(lost) heap-dynamic 변수 문제를 안고 있다. X

```
char c;
char * p1= &c;
p1 = (char*) malloc(sizeof(char));
```

(11) Java의 generic은 actual parameter에 상관 없이 하나의 컴파일된 코드를 생성하는 데 비해, C++의 template은 실제로 사용되는 데이터와 타입에 따라 여러 개의 컴파일된 코드를 생성한다. O

(12) 다음 `int foo(int f[][]) {… }`과 같은 함수선언은 C, C++에서는 가능하나 Java에서는 가능하지 않다. X

(13) C에서 `n`개의 `int` 필드를 가지는 `struct A`를 리턴 타입으로 하는 함수를 호출하면, 리턴 값 전달 시간은 `n`이 커질수록 커지게 된다. O

(14) Python에서는 scalar 변수가 parameter로 전달 될때는 call-by-value가, 배열 등이 전달될 때는 call-by-reference를 사용한다. O

(15) 다음 C 함수가 호출되어 수행되는 동안 (ㄱ) 시점에서, 변수 x는 ARI내의 parameter 용 공간에 위치하고, 변수 y는 ARI 내에 local variable 용 공간에 위치한다. X

```
int inc(int x){
      static int y=0;
      y+= x;                    //(ㄱ)
      return y;
}
```

(16) static scoping rule을 따르는 경우 callee의 static parent는 caller자신이거나 caller의 static ancestor 중 하나이다. O

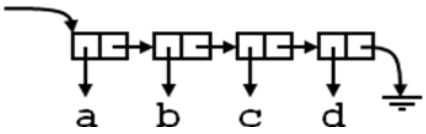
(17) 하위클래스 **Student** 타입의 변수 **x** 가 가리키는 것은 **Student** 클래스 객체일 수도 있고, 그의 상위 클래스**Person**의 객체일 수도 있다. **x**

(18) 상위클래스 **Person**과 하위 클래스 **Student** 모두 **void hello()** 메소드의 구현을 가지고 있는 경우, **Person** 타입 변수 **y**에 대해 다음과 같은 호출을 할 때

```
y.hello();
```

C++ 에서는 별도의 조치가 없으면 **y**의 값에 상관없이 **Person**에 정의된 **hello()**가 불린다. O

(19) Scheme에서 아래 리스트는 **(cons 'a '(b c d))**의 내부구조를 나타낸다. O



(20) Prolog의 **P(X) :- Q(X)**는 "**P(X)**가 만족하면 **Q(X)** 가 만족한다"는 의미를 가진다. X

3. (5점) 다음 grammar가 모호한 grammar인지 여부를 적으시오. 그 근거도 함께 적으시오.

```
<stmt> ::= if <exp> <stmt> else <stmt>
         | if <exp> <stmt>
```

답)

모호한 grammar이다 만 쓰면 1점

parse tree가 두 개 생긴다 만 적거나, 중첩에 대한 언급 없이 if/else 문 얘기만 있으면 3점

3점의 두가지 경우를 모두 적었으나 구체적인 예가 없으면 4점

4점의 답에 구체적인 예 (if E if S else S) 까지 있으면 5점

4. (5점) C 언어에서, 덧셈 (즉, +) expression이 교환 법칙이 성립되지 않는 예를 들고 이유를 적으시오.

답) "++a + a" 과 같은 구체적인 식을 적으면 5점만점

(++a 대신, a++이나 a 를 변화시키는 sideeffect. 가 있는 함수 호출 사용 가능)

side effect 를 언급하거나 ++a, a++ 만 들어간 식을 적으면 1점~3점

5. (5점) 허상참조 (dangling reference) 를 해결하는 방법을 한가지 들고 간단히 설명하시

오.

답)
"비석"에 대한 언급과 뭔지 설명하면 5점
"비석" 이름 언급 2점

6. (5점) C 언어에서 아래와 같이 선언된 3차원 배열(array)가 행우선(row-major) 방식으로 메모리가 할당될 때, `a[i][j][k]`의 주소를 계산하는 식을 쓰시오. `a[0][0][0]`의 주소는 A라고 하고, int 형의 크기는 4-byte라고 가정한다. (즉, 숫자와 A, i, j, k 만으로 구성된 식을 적으시오.)

```
int a[5][6][7];
```

답) `A+(6*7*i+ 7*j + k) *4`
6,7 대신 7,8로 적거나 i대신 i-1 등으로 적는 경우 2~3점 범위 부분점수

7. (10점) 다음은 리스트 `ls`가 중첩되지 않은 리스트를 나타낸다고 할 때, 특정 `item`이 `ls`에 들어있는지를 알아보는 함수이다. 밑줄을 완성하시오.

```
(lambda (item ls)
        (cond  ((null ls) (1)__________)
               ( (2)____________________) ))
```

답) 문제의 오타 때문에 10점 중 기본점수 5점과 아래 사항의 5점 만점으로 배정
(anonymous 함수의 recursive call이 되어 (2)에서 표현할 수 있는 코드가 없거나 복잡해짐. 최대한 후하게 채점)
(1) 3점 : #f, false, 거짓 등..
(2) - 3점 : equality 비교 (eq item car(ls)) #t
car 빠지면 0, eq 표시는 빠졌는데 의미상 확인되면 부분점수
- 4점 : else (lambda item cdr ls)
밑줄은 member, 또는 생략도 허용, 모두 점수 부여
item 없이 lambda cdr ls 로 적거나 cdr ls 만 있는 경우도 1~2점 부분점수

8. (10점) 객체지향 프로그램에서 클래스 A와 B가 주어졌을 때, `A` 객체를 인자로 받을 수도 있고, `B` 객체를 인자로 받을 수도 있는 어떤 함수 (메소드, 멤버함수, static 메소드, static 멤버함수 ... 등) `f`를 "다형성(polymorphism)" 극대화 하여 설계하고자 한다.
(1)~(4)경우 각각에 대해 가장 적절하다고 생각되는 형태를 보기(ㄱ)~(ㅁ) 에서 하나만 골라 적으시오.
답) ㄴ, ㄹ, ㄱ, ㅁ 각 2.5점씩

(1) `f`가 `A`를 인자로 받았을 때와 `B`를 인자로 받았을 때 하는 일이 완전히 다르고, `A`가 `B`의 상위클래스일 때
(2) `f`가 `A`를 인자로 받았을 때와 `B`를 인자로 받았을 때 하는 일이 완전히 다르고, `A`와 `B`가 상하위 클래스 관계가 아닐때.
(3) `f`가 `A`를 인자로 받았을 때의 프로그램 로직이, 인자 타입에 대한 부분을 제외하면 `B`를 인자로 받았을 때와 동일한데, `A`가 `B`의 상위 클래스일 때.

(4) f가 A를 인자로 받았을 때의 프로그램 로직이, 인자 타입에 대한 부분을 제외하면 B를 인자로 받았을 때와 동일한데, A와 B가 상하위 클래스 관계가 아닐때.

<보기>

(ㄱ) 함수 f 를 클래스 A의 멤버함수/메소드로 둔다.

(ㄴ) 클래스 A와 B 모두 f를 멤버 함수/메소드로 정의하고 dynamic binding을 한다.

(ㄷ) 클래스 A와 B 모두 f를 멤버함수/메소드로 정의하고 static binding 을 한다.

(ㄹ) A 타입 인자를 받는 함수 f와 B 타입 인자를 받는 함수 f를 둘다 static 메소드/멤버함수로 정의한다 (즉, overloaded subprograms).

(ㅁ) 타입 parameter를 도입하여 임의의 타입 T를 인자로 받도록, static 메소드/멤버함수 f를 정의한다.

9. (10점) 다음 Java 프로그램에서 main의 수행 결과가 4, 4가 되도록 CountedM의 addAll()을 수정하시오.

```
import java.util.*;
public class M {
      List s;
      public M() { s = new ArrayList();}
      public Object get (int i) {return s.get(i);}
      public void add(Object e) { s.add(e);}
      public void addAll(M c) {
            for (int i=0; i < c.s.size(); i++){ add(c.get(i));}
      }
}
public class CountedM extends M{
      private int addCount= 0;
      public CountedM() { }
      public void add(Object e) {
            addCount++;
            super.add(e);
      }
      public void addAll(M c) {
            addCount += c.s.size();
            super.addAll(c);
      }
      public int size() { return addCount;}
      public static void main(String[] args){
            CountedM m1 = new CountedM();
            m1.add("a"); m1.add("b"); m1.add("c"); m1.add("d");
            CountedM m2 = new CountedM();
            m2.addAll(m1);
            System.out.println(m1.size() + ",," + m2.size());
```

```
        }
    }
```

답) 아래 답들이 가능

답1)

```
addCount += c.s.size();
for (int i=0; i<c.s.size(); i++)
super.add(c.get(i));
```

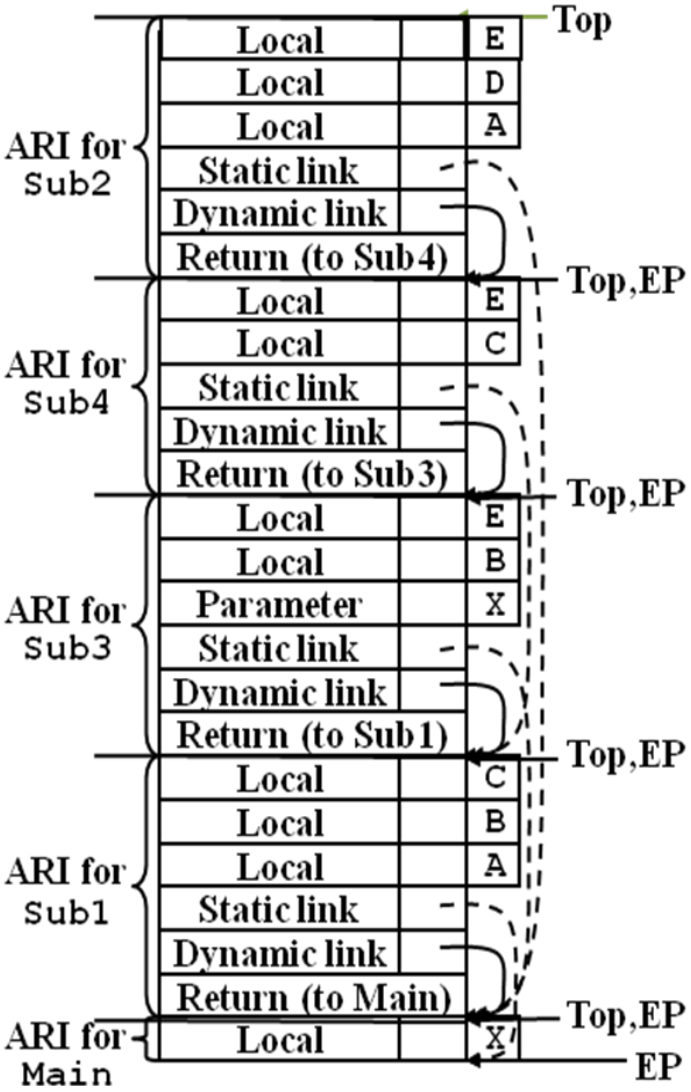
답2)

```
super.addAll(c);
```

답3)

```
for (int i=0; i<c.s.size(); i++)
add(c.get(i)); //or this.add(..)
```

10. (1) (10점) 오른쪽 그림은 **Sub1, Sub2, Sub3, Sub4** 의 subprogram들과 **Main**으로 이루어진 프로그램을 수행했을 때 구성된 runtime stack의 이다. 본 그림으로부터 유추할 수 있는 subprogram들의 nesting 구조와 각 변수가 선언된 위치 (어느 subprogram의 내부 인지)를 파악하여 원래 프로그램을 복원해보시오. 단 실행 코드 부분은 복원하지 않는다.



답)

```
main
  x
   sub1
      a,b,c
      sub3
         b,e,x
         sub4
              c,e
      sub2
         a,d,e
```

- 구조 6점 : 문제가 뭔지 알면 1점, 234를 같은 level로 적거나, 1,2,3을 같은 level로 적은 경우는 부분점수

- 변수 4점 : 대충 구조 알고 있으면 1점, parameter (X)를 빼먹거나 local로 적거나, 혹은 변수 하나 정도 못 쓰면 부분점수

(2)(5점) 이 프로그램이 static scoping rule을 따른 다면, display를 써서 수행하는 것과 static chain 을 써서 수행하는 것 중 어느 쪽이 더 나을지 논하시오. (필요하다면 적절한 가정을 하시오.)

답) static chain의 경우 depth에 따라 접근 속도가 커질 수 있는데 display로 해결할 수도

있지만, 예제 처럼 depth 가 2로 별로 안크고 display는 register에 구동되지 않으면 소용 없는 부분도 있으므로 static chain이 낫다.

- 아예 논하지 않거나 관련 없는 것을 적으면 0점
- 본 예와 무관한 일반론만 적은 경우 2점~3점 부분점수

(3) (5점) 이 프로그램이 dynamic scoping rule을 따른 다면, shallow access 방식을 써서 수행되었을 때, 위 그림과 동일한 시점의 symbol table의 모습을 그려보시오.

				2	
2	3	4		4	3
1	1	1	2	3	M
A	B	C	D	E	X

틀이 맞고 틀린 개수가 적을 때 3점내외 부분점수

한학기 동안 수고하셨습니다.