2014년

1번
책에는 object-based
데이터(representation)와 데이터와 관련된 procedure(operation)를 syntactic unit으로 묶어서 모듈화하여 object를 선언할 수 있기 때문이다. 그러나 object 개념만 있고 object를 찍어내는 class의 개념은 있지 않으며, 클래스 간의 관계인 상속의 개념이 없기 때문에 object-based 언어이다.
답지에는 class-based
ADT의 특성을 갖는 여러 개의 인스턴스를 만들 수는 있지만, (예를 들어 stack module/package의 특성을 갖는 여러 개의 인스턴스 stack i, j, k를 만들 수 있기 때문에 class의 기능은 있지만) 이런 module/package를 이용하여 새로운 module/package를 선언할 수 없기 때문이다. 즉, inheritance 기능을 제공하지 못하기 때문이다.

2, 3번 시험범위 해당 없음

4번
(a) (display_offset, local_offset)
display_offset: 가장 outer한 scope에서 봤을 때 해당 변수가 선언된 subprogram의 static depth
local_offset: ARI 시작 주소에서 떨어진 offset
(b) data 영역의 COMMON block에 nonlocal 변수들 저장. local 변수들도 data 영역 ARI에 저장됨. 실행 전에 이러한 것들이 할당됨.
답지: common 영역에 대한 주소
(c) ~~nonlocal 변수들에 대한 주소는 따로 생성하지 않음~~. dynamic link 따라가면서 해당 nonlocal 변수 나올 때 까지 찾아봄. 대신 ARI에 local 변수들의 이름을 모두 저장해야 함.
답지: non_local 변수 이름 자체
(d) ~~nonlocal 변수들에 대한 주소를 따로 생성하지 않음~~. compiler가 서로 다른 이름을 가진 변수들을 다 세서 각 변수명마다 seperate한 stack들을 만든 다음, subprogram이 호출될 때마다 해당 변수 stack에 호출된 subprogram의 이름을 갖는 cell을 top에 쌓음. 변수를 참조할 때는 해당 변수 stack의 위에 있는 cell을 참조하여 어디에 선언된 변수인지 확인함.
답지: non_local 변수에 대한 stack 주소

5번
```
float SUM(void ADDER, int INDEX, int LENGTH){
        float result;
        for (INDEX = 0; INDEX <= Length; INDEX++){
                result = result + ADDER;
        }
        return result;
}
```

6번
a) subprogram의 이름과 formal pamameter list(type과 개수)
b) macro는 코드를 expansion해서 overhead가 적지만 type checking 하지 않음.
subprogram은 type checking을 하지만 call과 return에서 일어나는 overhead가 존재함.

inline은 코드를 expansion하여 함수 호출 overhead가 없으면서 (macro의 장점) type checking (subprogram의 장점) 하지 않음.
c)
(가)
actual parameter를 넘겨줄 때: 함수 호출 수행 전 또는 함수 수행 후 값을 받을 때
formal parameter의 값을 받을 때: 함수 호출 수행 전 혹은 함수 수행 후 값을 받을 때
(나)
함수 호출 수행 전에 주소를 계산을 하여 주소를 넘겨준다.
(다)
수행 시 함수 body에서 formal parameter가 참조될 때마다 매번 계산
d) funtional side effect를 막아서 subprogram 바깥 쪽에 영향을 주지 않도록 하기 위해
답지: 함수 body에서 formal parameter가 변경되는 것을 방지하기 위해

7번
(a) // A::i = 19;
subclass에서 super class의 private access level로 선언한 instance variable를 접근 불가하다.
(b)
198 187 187

8번
(a) 6 10
(b) 6 unknown
(c) 6 11
(d) 6 12
(e) 8 10

2015년

1번
(a) inheritance는 기존에 정의한 data type을 가지고 새로운 data type을 정의할 때 공통적인 것을 선언하고 그것을 가져다 쓸 수 있도록 하기 위함인데, inheritance 기능이 없다면 다음과 같이 상속을 받고자 하는 subclass에는 super class의 특징을 모두 갖도록 선언해야 한다.
예) class A : super class , class B: sub class 일 때

```
class A (){
    int a;
    int aa();
}
class B(){
    int a;
    int aa();
    int b;
    int bb();
```

}
(b) parameterized abstract data type을 지원하는 template 기능이 있어서 class의 member function을 generic function처럼 만들어준다.
generic function을 진정으로 지원하려면 language가 dynamic type binding이어야 하지만, C언어는 기본적으로 static type binding이므로 컴파일러가 type에 따라 다른 object를 선언하면 type에 따른 각각 다른 코드를 expansion하여 generic function을 지원하는 것처럼 한다.

2.
(a) dynamic link, local variable, return address , parameter
(b) - (1) static link
(b) - (2) 같은 depth의 display pointer가 가리키는 주소(link)를, 변경되는 같은 depth의 ARI가 갖고 있어야 한다.
(c) dynamic link, parameter, return address
(d) - (1) local variable, local variable의 이름도 같이 저장해야 한다.
(d) - (2) ARI에 추가로 저장해야 할 정보는 없다.
(e) 데이터 영역에 각 subprogram마다 한 개의 ARI를 만드는데, ARI의 시작 주소로부터 일정 offset 떨어진 공간에 return address를 저장할 공간을 할당해놓고, 실행 시 해당 subprogram이 호출되면 callee의 ARI return address 공간에 caller 코드의 중단 지점 다음 instruction의 주소를 저장한다.

3.
(a) dynamic message binding에 의해 shape라는 super class object pointer가 가리키는 실제 object이며, shape의 subclass object에 선언된 member function들이다.
(b) C++는 파일 단위로 컴파일 하면서(independent compilation) 함수의 호출과 그의 header의 parameter type을 checking 하지 않으며, 또한 shape.c에 있는 area()함수를 수행할 때 dynamic message binding에 의해 실행 시간 때 super class object pointer가 가리키고 있는 sub class object를 확인하여 해당 member function을 찾아서 실행하므로 client.c를 컴파일할 필요가 없다.

4.
(a) 1, 3, 1
(b) 1, 3, 5
(c) 1, 3, 5
(d) 1, 3, 5
(e) 1, 5, 3

5.
(a) //i = Two::value() + 1
super class는 sub class의 outer scope이기 때문에 super class에서 scope resolution operator를 사용한다고 해서 sub class의 member function이 visible 하지 않아 호출이 불가하다.
// y = &c
subclass의 object pointer에 super class의 object를 할당할 수 없기 때문이다.
(b)
11
11
4

(c)
4
4
4

6.
(a)
```
 for (int i = 0; i < s; i++){
    p[i] = v.p[i];
}
return(*this);
```
(b)
```
for(int i = 0; i < s; i++){
    sum[i] = p[i] + v.p[i];
}
return(sum);
```
(c)
컴파일 에러가 아니다.
컴파일러가 programming language에서 제공하는 = operator의 양변의 operand type이 서로 같은지 check하고, 그에 맞는 type의 operation을 찾아서 해당 procedure 호출 코드로 바꾼다. 그런데 C++에서는 = operator의 양변의 operand data type이 다르면 오른쪽 operand의 data type을 확인하고, 해당 data type을 parameter로 갖는 constructor를 찾아서 이를 수행하여 임시 object를 생성하는 코드로 바꾼다. 즉, a = vect(10);으로 되기 때문에 컴파일 에러가 나지 않는다.
(d)는 잘 모르겠습니다...

7.
(a) :-a.가 수행되면 이 a라는 goal이 true인지 db에 저장된 fact와 rule들을 보고 backtracking하여 확인한다.
우선 a :- b(X, Y), c(X, Y)를 볼 텐데, 이 의미는 a가 true이기 위해서는 b(X, Y)가 true이고 c(X, Y)가 true이어야 한다는 것이다. 그래서 변수 X, Y의 임의의 값에 관해 b(X, Y)가 true인지 확인하기 위해 db 어딘가에 정의된 fact 또는 rule을 unify하여 찾고, b(X, Y)에 관해 true이면 해당 previous subgoal을 저장하여 다시 돌아왔을 때 이를 재개할 수 있도록 하고 마찬가지로 c(X, Y)도 이와 같이 수행할 것이다. c(X, Y)가 false이면 다른 X, Y에 관해 b(X, Y)가 true인지를 탐색하는 과정을 수행한다. 만일 b(X, Y)와 c(X, Y)가 X ,Y의 어떤 임의의 값에 관해 모두 true를 만족하지 않으면 다음 줄의 a:- d(A, B), e(A, B)를 참고하는데, 이는 첫째 줄과 둘째 줄이 or 관계이기 때문이다. 마찬가지로 둘째 줄에 대해서도 위에서와 같은 방법을 수행한다.
(b)
(가)
X = [], Y = [1, 2, 3]
X = [1], Y = [2, 3]
X = [1 ,2], Y = [3]
X = [1, 2, 3], Y = []
(나)
X = [i, [am, not], a, computer]

X = [i, are, a, computer]
X= [you, [am, not], a, computer]
X = [you, are, a, computer]
(c)
(가)
자매(X, Y) :- female(X), female(Y), parent(X, Z), parent(Y, Z), not(X=Y).
(나)
이모(X, Y):- parent(X, Z), 자매 (Z, Y).
(다)
조부모(X, Y) :- parent(X, Z), parent(Z, Y).

2017년

1.
(a)
```
HighOrderSort(OP, int Arr[], size){
    int i, j, temp;
    for (i = 0; i < size; i++){
        for (j = size-1; j > i; j--){
            if (Arr[j] OP Arr[j -1]){
                temp = Arr[j];
                Arr[j] = Arr[j-1];
                Arr[j-1] = temp;
            }
        }
    }
}
HighOrderSort(>, a, 100); // descending sort
HighOrderSort(<, a, 100); // ascending sort
```
(b)
generic sorting subprogram은 여러 type의 element를 갖는 array(예: int a[100], float b[100] 등)를 sorting 하여야 한다. 이를 위해서는 HighOrderSort 함수의 parameter로 넘어온 array element tpye과 비교 operator('<', '>')를 참조하여 if(Arr[j] OP Arr[j-1])에서 어떤 type에 관하여 비교 operation을 수행할지 dynamic하게 결정하여 수행해야 한다. 이런 작업을 하는 코드를 compiler가 생성하기 어렵기 때문에 interpreter가 이런 일을 수행하는 것이 바람직하다.

2. 2014년 8번과 중복

3.
(a) 호출할 때 parameter의 주소를 미리 계산하는데, 변수에 대한 연산이 parameter로 올 경우 연산을 수행한 값을 임의의 temporary 메모리에 저장하여 해당 메모리의 주소를 parameter로 넘겨준다.
(b) CALL P((X+0), (X+Y), 2)
CALL P((X), (X+Y), 2) 처럼 변수 X의 값을 갖는 임의의 temp 변수에 따로 저장하여 이 메모리의 주소를 넘겨주도록 변수 X에 괄호를 넣는다.

4. 2014년 6번과 중복

5.
(a)
swap(i , ch);
swap(str1, str2);
(b)
template를 사용하는 subprogram에서 parameter를 받을 때 같은 type을 generic하게 받는데, swap(i, ch) statement는 parameter의 type이 서로 다르고, swap(str1, str2)는 변수가 배열의 주소 그 자체인데 함수 선언부에서는 이를 주소로 또 받으려고 하며,

6.
(c)
static chianing
XXX 변수 주소: (chain_offset, local_offset)이므로 (2, ARI format에 따라 local offset은 다를 것임)
수행할 때: subprogram이 호출되면 callee 선언한 subprogram과 caller의 static depth 차이 구해서 그만큼 caller부터 static link 따라간 곳을 callee의 static link를 연결해줌. nonlocal 변수 참조할 때 compiler에 의해서 생성된 변수 주소만큼 현재 ARI에서 static link 따라가고 그 ARI 시작으로부터 offset만큼 떨어진 곳을 참조함.
display
XXX 변수 주소: (display_offset, local_offset)이므로 (0, ARI format에 따라 local offset은 다를 것임)
수행할 때: 현재 수행되고 있는 subprogram들의 static depth마다 display pointer로 링크해줌. 만약 같은 depth의 display link가 변하면 변하기 전의 depth의 display link가 가리키고 있는 주소 정보를 새로 변하는 같은 depth의 ARI에 저장해야 함. nonlocal 변수 참조할 때 compiler에 의해서 생성된 변수 주소의 display_offset에 해당하는 display pointer 찾아서 그것이 가리키는 곳의 ARI의 시작으로부터 떨어진 local offset을 통해 참조함.

8.
(b) check([A, A]).
check([A, _|L]) :- check([A|L]).
(c) O(L)

9.
(a) dynamic
db에 서 대응되는 fact 또는 rule을 찾아서 constant term을 parameter passing하여 unify할때?
(b) 제공한다.
첫 번째 list의 원소를 두 번째 list 앞에 넣어서 새로운 list를 만들 때, 원소의 data type이 integer이든 character이든 두 가지 이상의 data type에 관해 모두 지원하기 때문에?
(c) scoping rule이 없음. nonlocal, static 변수가 존재하지 않기 때문에?
(d) backtracking하며 rule 또는 fact에 선언된 parameter tpye 직접 대조해서??

2018년

1.
(a) COMMON으로 선언한 변수들은 모두 data 영역의 COMMON Block에 저장함. nonlocal 변수 참조할 때 COMMON Block 영역에 대한 주소를 가지고 참조함.
(b) 자기가 선언하지 않은 nonlocal 변수들을 참조하기 위한 방법이며, C언어는 nested된 subprogram을 지원하지 않아서 자기가 선언하지 않은 변수들은 모두 global 변수이기 때문에 이런 구현이 필요 없다.
(c) 기본적으로 C언어는 함수를 호출하는 부분과 함수 선언 header 부분 사이의 parameter type checking을 하지 않기 때문에 함수의 prototype을 선언하여 parameter type checking과 coercion을 구현하기 위해서.
(d) 거시적으로 봤을 때, 메인 메모리에 올라간 두 개 이상의 procedure가 동시에 수행되는 것처럼 보이는 것이 병행처리. 이는 실제 CPU가 하나임에도 불구하고 두 개 이상의 procedure가 짧은 interval를 두고 번갈아가면서 시행하여 동시에 수행되는 것처럼 보이는 것과, 두 개 이상의 CPU가 각각의 procedure를 실행함으로써 병렬적으로 동시에 수행하는 것으로 나눌 수 있다. 후자가 병렬처리이다.
(e) 같은 특성을 갖는 object를 찍어내는 틀이 없기 때문에 object마다 일일이 class에 해당하는 특성을 모두 갖도록 직접 선언해줘야 한다.
예)

```
object a{
    int a;
    int aa();
}
object b{
    int a;
    int aa();
}
...
```

2.
(a)

```
procedure integrate (function fun(x: real) :real; lowerbd, upperbd: real, var result: real);
    var funval : real;
    begin
        ... (interval마다 함수값 구해서 interval끼리 곱한 것 더하는?)
        funval := fun(lowerbd);
        ...
    end;
```

(b) 함수의 header 선언 부분에 parameter type이 선언되지 않아야 하고(syntax에서), 넘어오는 parameter type이 dynamic type binding이어야 한다(mechanism에서)?

3.
(a) c = a;
subclass에서 superclass에서 private access lever로 선언된 instance variable 참조할 수 없다.
(b)
Creating A
Creating B!

a = 3
b = 3, c = 4
Destroying B!
Destroying A!

4.
(a) B::i = 0
(b) A::d2.i=3
(c) A::i=3
(d) B::d2.i=5
(e) A::d2.i=3

5.
(a) abstract base class를 쓰는 이유는 virtual로 선언한 member function을 subclass에서 overriding하도록 enforce하기 위한 목적인데, 만약 virtual로 선언하면서 body가 null인 member function을 사용하지 않으면 subclass들이 이 member function을 상속할 수 없으며, abstract base class를 사용하는 다음과 같은 경우에 compile error가 발생한다.

```
shape *ptr_shape;
...
ptr_shape->area();
```

(b) dynamic message binding을 지원해서 runtime 때 해당 object를 찾아가 직접 object의 class type을 봐서message를 sending하기 때문에 필요 없음.

6.
중복

7.
(a)
```
birthday(홍길동, 20110312).
...
```

(b) :-writeln('Enter Name'), read(X), birthday(X, Y), write(Y).
(c) :-birthday(X,Y), write(X), write('/'), write(Y), nl, fail.