프로그래밍언어론

2018학년도 2학기 기말시험

핵심체크 및 출제예상문제

범위 : 교재 전 범위

제1장 프로그래밍 언어의 소개

1. 프로그래밍 언어란?

(1) 프로그래밍 언어

① 프로그래머의 의사를 전달하는 방법이며, 동시에 프로그램을 작성하는 형식

② 컴퓨터가 읽을 수 있고 사람이 읽을 수 있는 형식으로 컴퓨터 계산(행동)을 서술하기 위한 표기 체계

③ 폰 노이만: 컴퓨터는 전선 연결방법으로 특정 작업이 수행되도록 지시되어서는 안 되며 중앙처리장치가 수행해야 할 작업을 일련의 명령코드로 작성해서 컴퓨터 내부의 자료로 저장해야 한다고 주장

(2) 프로그래밍 언어론의 필요성

① 현재 사용하고 있는 언어를 더욱 잘 이해함

② 유용한 프로그래밍 구사능력 증대

③ 프로그래밍 언어를 선택할 수 있는 능력 증대

④ 새로운 프로그래밍 언어를 배우기 쉬어짐

⑤ 새로운 프로그래밍 언어를 설계하기 쉬어짐

(3) 추상화

① 속성들의 특징적인 일부분만을 가지고 주어진 작업이나 객체들을 표현하고 그들의 공통점을 추출하여 표현

② 대상에 따라 분류: 자료추상화, 제어추상화

③ 정보양에 따라 분류: 기본추상화, 구조적 추상화, 단위추상화

(4) 계산 전형

① 명령형 언어(절차언어): 변수 값을 변경시키기 위한 명령문을 순서대로 나열한 것

② 함수 언어(적용형 언어)

- 알려진 값들을 함수들에 적용
- 매개변수에 함수를 적용함

③ 객체 지향 언어: 실세계에 존재하는 모든 유형 및 무형의 대상이 되는 객체를 클래스로 표현

④ 논리형 언어(선언적 언어): 반복이나 선택문 없이 계산의 내용만을 선언하듯 기술(기호논리학)

(5) 언어의 정의

① 구문론

- 언어 구성요소의 외부적인 형태에 관한 것
- 대부분 문맥 무관형 분법으로 정의되고 있다.
- C언어의 if문을 문맥 무관형 문법으로 정의하면 다음과 같음

  <if문>:: = if (<조건>) <문>

② 의미론: 언어의 표면적 구조만을 나타낸 것, 프로그램이 무엇을 어떻게 수행할지 나타내 줌

2. 프로그래밍 언어의 역사

(1) 1950년경: 기억장치를 고려한 최초의 프로그래밍 언어

① FORTRAN: 어셈블리 언어로 프로그래밍을 하였고 하드웨어가 고가였기 때문에 언어 설계에 있어서 무엇보

다도 효율성이 강조됨(수치 계산에 적합하도록 개발된 언어)
③ Cobol: 사무처리와 같은 사무처리의 목적으로 만들었으며 레코드 구조를 도입하고 자료구조 부분과 실행 부분을 분리
④ Algol 60: 연구용 및 실질적은 응용목적의 언어(후속 언어에 많은 영향을 미침)
⑤ LISP: 일반적인 리스트 구조와 함수 응용을 기본으로 함 (기호계산을 위한 함수 언어)
⑥ APL: 다양한 연산자로 배열조작을 간단히 한 함수(제어구조가 없음)

(2) 1960년대: 다양한 프로그래밍 언어 출현
① Algol 68: Algol 60에 중요한 언어 개념을 추구하고 신택스도 공식적으로 정의한 최초의 언어
② PL/I: FORTRAN, COBOL, Algol의 정점만을 언어에 넣음
③ SNOBOL: 최초의 문자열 처리 및 패턴 매칭 기능 언어
④ BASIC: 사용자와의 상호작용을 강조하여 편리성을 도모한 인터프리터 방식(간단한 언어 교육용, 사무처리 및 가정용으로 널리 보급)
⑤ Simula 67: 시뮬레이션 문제에 적합하게 설계된 언어로서 클래스라는 개념을 최초로 도입

(3) 1970년대; 간결성 추상화, 효율성
① Pascal: Algol의 후속언어, 작고 간결하고 효율적이며 교육적인 목적의 구조적 프로그래밍 언어(범용 언어로 개선)
② C: 컴퓨터 구조에 대한 접근을 제공해 주는 개념을 갖는 중급 프로그래밍 언어(UNIX 운영체제 구현 사용)

(4) 1980년대: 통합과 새로운 방향
① Ada: 기존의 명령형 언어의 개념을 종합한 것으로 볼 수 있으며 추상 자료형 구조, 병렬 프로그래밍 구조, 예외 처리 등의 흥미 있는 기능들이 많이 포함되어 있음
② Smalltalk: 본격적인 객체 지향 언어의 효시
③ 객체지향 언어로서 C++, Eiffel이 발표
④ CLOS는 LISP에 객체 지향 개념을 추가한 것
⑤ Modula-2: 추상화, 부분적 동시 처리 개념과 내장형 시스템 프로그래밍을 목적으로 함(모듈화 프로그램 설계 개념을 도입하여 Pascal의 후속 언어로서 발표)
⑥ Scheme: LISP를 교육용에 맞게 변형시킨 것(인공지능에 대한 관심으로 많이 이용)
⑦ Prolog: 논리 언어로 1972년에 제안되었으며 수학적 논리학을 적용한 프로그래밍 언어

(5) 1990년대: Web 프로그래밍
① 응용문제의 해결과 프로그래밍 간편성에 집중한 프로그래밍 언어들의 개발 집중
② Java: 간결하면서도 어느 기계로든 이동되어 실행될 수 있다는 점 때문에 네트워크 언어로서 각광을 받고 있음

3. 프로그래밍 언어의 설계기준

효율성(목적코드의 효율적, 번역의 효율성, 프로그래밍 효율성), 정확성(행위를 예측할 수 있는 정의에 의존). 표현력(언어가 복잡한 과정이나 구조 표현하는데 용이), 일반성, 획일성, 직교성, 컴퓨터 독립성, 안전성, 기존 표기나 규칙과의 일관성, 확장성, 부분성

<1장 출제예상문제>

1. 프로그래밍 언어론의 필요성으로 잘못된 것은?
① 유용한 프로그래밍 구사 능력이 증대된다.
② 현재 사용하고 있는 언어를 더욱 잘 이해한다.
③ 새로운 프로그래밍 언어를 배우기 쉬어진다.
④ 한 가지의 프로그래밍 언어의 능력이 증대 된다.

[해설] 프로그래밍 언어론의 필요성
- 현재 사용하고 있는 언어를 더욱 잘 이해함
- 유용한 프로그래밍 구사능력 증대
- 프로그래밍 언어를 선택할 수 있는 능력 증대
- 새로운 프로그래밍 언어를 배우기 쉬어짐
- 새로운 프로그래밍 언어를 설계하기 쉬어짐

[정답] 1

2. 프로그래밍 언어에서의 추상화를 정보의 양에 따라 분류한 것에 속하지 않는 것은?
① 구조적 추상화 ② 기본적 추상화
③ 자료 추상화 ④ 단위적 추상화

[해설] 정보의 양에 따라 분류
- 기본적 추상화: 가장 국지적인 기계 정보에 관한 추상화
- 구조적 추상화: 프로그램의 구조에 대한 보다 전역적인 정보에 관한 추상화
- 단위 추상화: 단위 프로그램 전체 정보에 관한 추상화

③ 대상에 따라 분류한 것으로 자료추상화(문자열, 변수, 탐색트리), 제어추상화(반복문, 조건문, 프로시저 호출)가 있음

[정답] 3

3. 반복이나 선택문 없이 계산의 내용만을 선언하듯 기술하는 언어는?
① 명령어 언어 ② 논리형 언어
③ 함수형 언어 ④ 객체지향 언어

[해설] 논리형 언어(선언적 언어)
- 반복이나 선택문 없이 계산의 내용만을 선언하듯 기술
- 기호 논리학에 근거를 두고 있음

[정답] 2

4. Smalltalk의 특징이 될 수 없는 것은?
① 캡슐화
② 계승
③ 함수언어
④ 객체 지향 언어

[해설] Smalltalk
객체지향 접근 방식으로 일관된 객체지향 언어의 순수한 모범 케이스

[정답] 3

5. 좋은 언어가 갖추어야 할 요건에 속하는 것은?
① 한 기종에서만 실행시킬 수 있도록 설계되어야 한다.
② 언어의 개념이 될 수 있으면 복잡해야 한다.
③ 구문이 명백해서는 안 된다.
④ 효율이 좋아야 한다.

[해설] 프로그래밍 언어의 설계 기준
효율성, 표현력, 정확성, 일반성, 직교성, 획일성

[정답] 4

6. 다음 중 논리 언어인 것은?
① Prolog
② Java
③ LISP
④ Smalltalk

[해설] Prolog
논리 언어로 1972년에 제안되었으며 수학적 논리학을 적용한 프로그래밍 언어로서 인공지능분야에 많이 쓰임

② Java: 중간 코드 형태로 제공되어 자바가상기계(JVM)위에서 수행됨으로써 이식성이 좋음
③ LISP: 일반적인 리스트 구조와 함수의 응용을 기본으로 하여 인공지능 분야에 폭넓게 사용

[정답] 1

7. 다음 중 컴파일러 방식과 인터프리터 방식의 혼합 형태를 취한 언어인 것은?
① C++
② Pascal
③ Java

④ Scheme

[해설]
① C++: C언어를 확장하여 객체지향 프로그래밍 개념 도입
② Pascal: 작고 간결하고 효율적이며 교육적인 목적의 구조적 프로그래밍 언어로 분리 컴파일 기능, 문자열 조작, 입출력 기능의 효율화 등의 고기능성을 갖춤
④ Scheme: LISP의 한 버전이며, 인공지능에 대한 관심으로 많이 이용

[정답] 3

8. 다음 중 프로그래밍 언어에 대한 설명으로 틀린 것은 ?
① 컴퓨터가 읽을 수 있고 사람이 읽을 수 있는 형식으로 계산을 서술하는 표기체계이다.
② 프로그램을 작성하는 형식이다.
③ 컴퓨터에 프로그래머의 의사를 전달하는 방법이다.
④ 컴퓨터의 구조를 서술하며 이를 통해 컴퓨터 간의 의사소통이 이루어진다.

[해설]
프로그래밍 언어란 프로그래머의 의사를 전달하는 방법이며, 동시에 프로그램을 작성하는 형식이고, 컴퓨터가 읽을 수 있고 사람이 읽을 수 있는 형식으로 컴퓨터 계산(행동)을 서술하기 위한 표기 체계이다. 컴퓨터의 구조를 서술하지는 않는다.

[정답] 4

9. 다음 중 실세계에 존재하는 모든 유형 및 무형의 대상을 객체라는 개념으로 정의하고 이에 특정값을 저장하는 기억장소와 그 기억장소의 값을 변경할 수 있는 연산으로 구성하는 방법을 사용하는 프로그래밍 언어를 무엇이라 하는가 ?
① 객체지향 언어
② 적용형 언어
③ 함수형 언어
④ 명령형 언어

[해설]
객체 지향 언어는 실세계에 존재하는 모든 유형 및 무형의 대상이 되는 객체를 클래스로 표현한다.

[정답] 1

10. Algol의 한계성을 극복한 하나의 사례로서 유닉스 운영체제의 구현에 사용되면서 많은 가능성을 보여주었던 프로그래밍 언어는 무엇인가 ?
① Fortran
② JAVA
③ Ada

④ C

[해설]
C는 컴퓨터 구조에 대한 접근을 제공해 주는 개념을 갖는 중급 프로그래밍 언어로서 UNIX 운영체제 구현에 사용 되었다.

[정답] 4

제2장 프로그래밍 언어의 구조 및 해석

1. 언어구문

(1) 프로그래밍 언어의 어휘구조

① 프로그래밍 언어 알파벳 문자로 구성된 단어 = 어휘토큰

② 구문구조와 별개지이만 밀접한 관련 있음

③ 번역기

- 어휘분석: 입력프로그램의 일련 문자를 토큰으로 구분
- 구문분석: 구문구조 결정

④ 예약어: 미리 정의된 식별자(for, case 등)

- 장점: 오류 검색 시간 단축, 프로그램의 가독성 향상, 컴파일러의 탐색 시간 단축
- 단점: 프로그래밍 언어 확장시 예약어와 식별자의 충돌, 예악어 관리 어려움

(2) 문맥자유 문법과 BNF

1) 구문형식과 문맥자유 문법

① 문맥자유 문법: 모든 생성 규칙에서 정의될 대상이 하나의 비단말기호만으로 구성된 문법

② 문맥의존 문법: 특수한 문맥에 의존하여 대치되는 문법

2) BNF

① 생성규칙들의 집합

- 생성규칙을 적용하여 정의된 언어로 작성가능한 모든 정상적인 프로그램 산출
- 작성된 프로그램이 구문에 맞는 프로그램인지 아닌지를 결정하기 위한 구문 규율로 사용
- 왼쪽: 정의될 대상
- 오른쪽: 그 대상에 대한 정의

② 예: BNF 표기법에 의한 식별자 정의

<identifier> ::=<letter> | <identifier><letter> | <identifier><digit>

<letter> ::=A | B | C ... | X | Y | Z

<digit> ::= 0 | 1 | 2 ... |8 | 9

- 비단말기호: 각진 괄호 '<>'로 묶인 기호, 다시 정의될 대상
- 단말기호: 각진 괄호로 묶이지 않은 기호, 알파벳 문자 집합, 예약어
- 메타기호: '::=', '|', '<', '>'

3) EBNF

① 보다 읽기 쉽고 간결하게 표현할 수 있는 확장된 BNF

② 특수 의미의 메타 기호를 더 사용하여 반복되는 부분이나 선택적인 부분을 간결하게 표현함

③ 표현: <compound-statement> ::=begin <statement> {; <statement>} end

BNF표현: <compound-statement> ::=begin <statement-list> end

<statement-list> ::=<statement> | <statement-list> ; <statement>

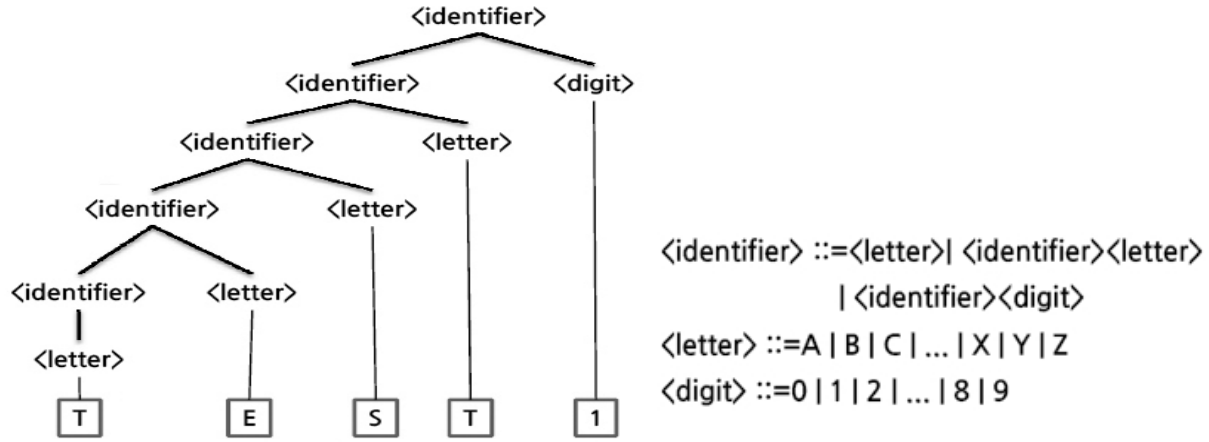
(3) 구문도표

① EBNF 방법 외에 구문에 대한 형식 정의하는 또 다른 방법
② 순서도와 비슷하며 EBNF 선언과 바로 대응시킬 수 있음
③ 비단말 기호: 사각형, 단말기호: 원 또는 타원, 정의관계: 지시선

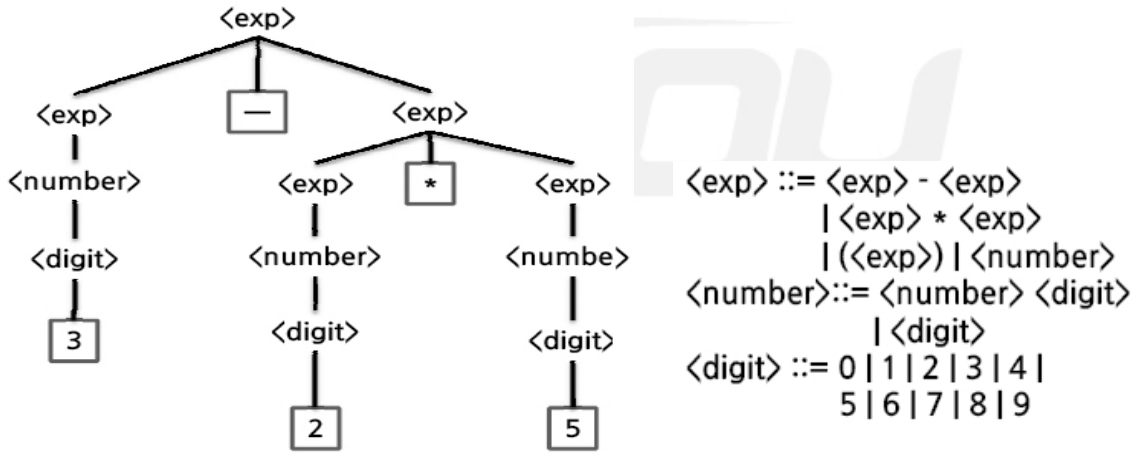
단말 X(::=x): 원 또는 타원 안에 표기하고 나가는 지시선을 그림 비단말 X(::=<x>): 사각형 안에 표기하고 지시선 그음

(3) 파스 트리와 프로그램 문법의 모호성
1) 파스트리
① 주어진 BNF에 의해 어떤 표현이 생성될 수 있는지 확인하기 위해 작성하는 트리구조 단말노드 나열

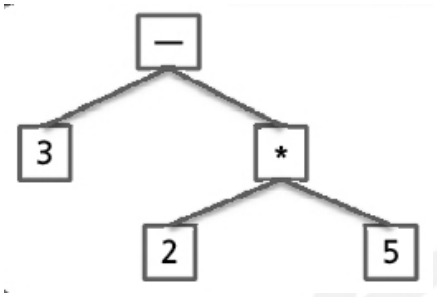
식별자 test 1에 대한 하스 트리



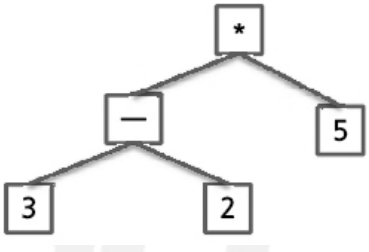
3-2*5에 대한 파스 트리



② 추상구문트리(구문트리): 파스트리의 본질적인 구조를 타내는 트리(불필요한 비단말 기호제거, 동일한 파스 트리, 서로 다른 파스 트리 등)
③ 추상구문 트리 비교



-> 결과값: -7



결과 값: 5

=> 위 경우와 같이 서로 다른 파스트리가 만들어지는 문법을 모호하다고 함(모호성)

④ 모호성 제거법칙
- 비단말 기호와 문법규칙 추가
- BNF 문법에 좌순환 규칙을 사용하여 좌결합 지원

2) 프로그램 문법의 모호성

① 프로그램 문법의 신뢰성: 프로그래밍 언어의 구문이 사람과 기계에 의해 쉽게 분석될 수 있어야 함
㉠ A=B=C; : B에 C의 값을 할당하고, A에는 B의 값을 할당하는 다중 할정
- A, B= C; : PL/1에서의 다중 할정문은 A, B = C;로 표현됨
- A=(B=C) : 사용한 의도의 다중 할당이 아닌, B와 C가 같은지 틀린지에 따라 A에 참 또는 거짓을 할당되는 것으로 해석

㉡ 변수의 선언을 명확하지 않은 언어의 경우 철자 오류가 묵시적 선언으로 간주되고 처리되어 올바른 프로그램을 어렵게 만듦

② 프로그램 문법의 모호성
㉠ IF절에서 else 문 처리
- else 문제 -> 현수 else(dangling else)
- BNF 문법과 관계없이 언어 자체가 포함하고 있는 모호성에 해당
- 현수 else 모호성: 두 가지 의미를 가질 수 있는 문제 발생, 이러한 else 문제를 현수 else이라 함. BNF 문법과 관계없이 언어 자체가 포함하고 있는 모허성에 해당

2. 프로그래밍 언어 구현기법

(1) 컴파일 기법

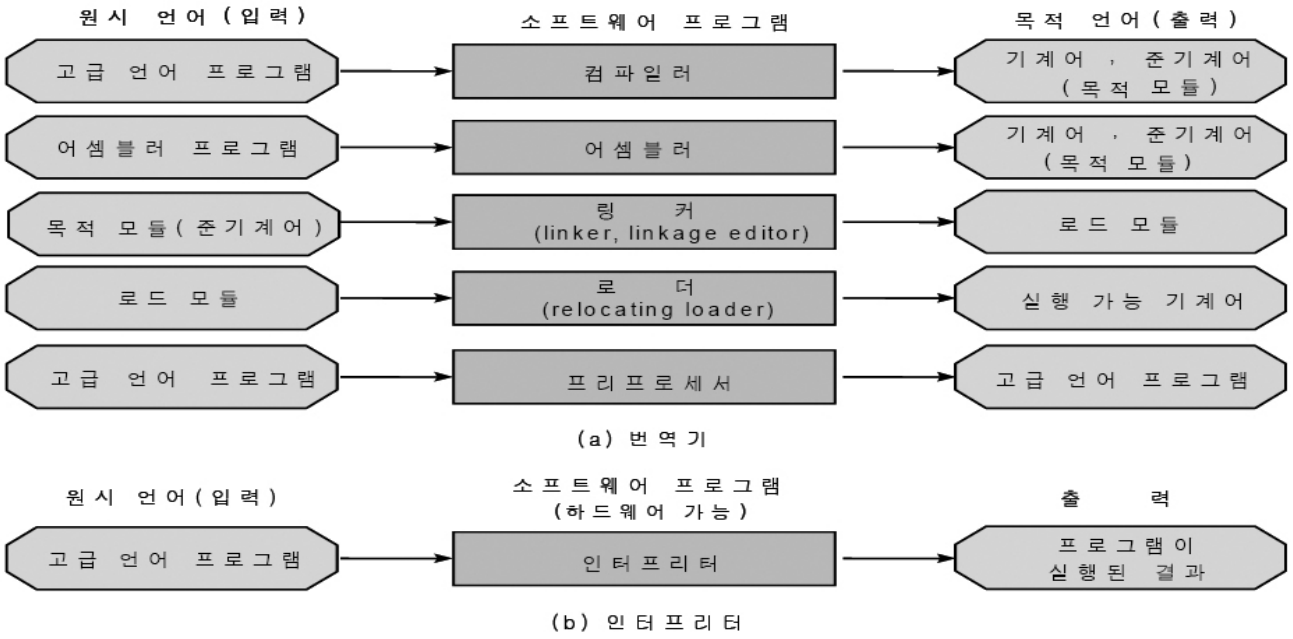
① 컴파일러: 고급언어를 저급언어인 목적언어로 만들어주는 번역기
② 어셈블러: 원시언어가 어셈블리 언어인 번역기
③ 링커: 재배치 형태의 기계어로 구성된 여러개의 프로그램을 묶어서 어느 정도 실행 가능한 하나의 기계어로 번역함
④ 로더: 로드 모듈의 기계어 프로그램을 실행 가능한 기계어로 번역하여 주기억 장치에 적재함
⑤ 프리프로세서: 원시언어와 목적언어가 모두 고급

(2) 인터프리트 기법

① 고급언어를 기계어로 취급하여 이를 실행할 수 있는 고급언어 기계를 소프트웨어로 시뮬레이션하여 구성하는 방법
② 인터프리터로 프로그램을 실행시키는 과정

입력자료
↓
고급언어로 작성된 원시프로그램 → 인터프리터 ← 실행결과

③ 번역기종류와 인터프리터



(3) 컴파일과 인터프리트 기법 비교

컴파일 기법	인터프리터 기법
입력프로그램과 동일한 목적 언어로 된 목적 코드 출력만 가능	직접 고수준의 프로그램을 실행 가능
각 문장을 입력된 순서대로 한 번씩 처리	논리적 순서에 따라 문장들 실행
입력(원시)언어가 어셈블리 언어처럼 기계어에 가까운 언어일 경우 주로 사용	운영체제의 작업 제어 언어나 APL과 같은 대화용 언어에 주로 사용
고급언어 번역 -> 목적 모듈 출력 -> 링크, 로드 -> 실행	고급언어 -> 중간코드까지만 번역 -> 실행
번역된 코드 -> 다시 번역할 필요 x	실행시마다 실행 시뮬레이션을 통해 실행
컴퓨터의 실행시간을 중시하는 경우 사용	사용자의 유연성을 중시하는 경우 사용
추가 기억장소 필요 없으며 유연성 높은 언어의 구현에 있어 편리함	많은 횟수로 반복처리되는 프로그램을 실행할 때 매우 효율적
번역된 프로그램이 매우 큰 기억장치를 요구할 수 있음	실생시간을 매우 많이 요구할 경우 발생
Fortran, Algol, PL/I, Pascal, Cobol, C 등	Lisp, Snobol4, APL, Prolog

<2장 출제예상문제>

1. 인터프리터 기법에 대한 설명이 아닌 것은?
① 직접 고수준의 프로그램을 실행 가능
② 논리적 순서에 따라 문장들 실행
③ 컴퓨터의 실행시간을 중시하는 경우 사용
④ 운영체제의 작업 제어 언어나 APL과 같은 대화용 언어에 주로 사용

[해설]
③ 컴파일 기법에 대한 설명. 인터프리터 기법은 사용자의 유연성을 중시하는 경우 사용됨

[정답] 3

2. 예약어와 관련 없는 것은?
① 프로그램의 가독성 향상
② 예약어 관리의 어려움
③ 컴파일러 탐색 시간 단축
④ 오류 검색 시간 증가

[해설] 예약어의 장단점
- 장점: 프로그램 가독성 향상, 컴파일러의 탐색 시간 단축, 오류 검색 시간 단축
- 단점: 예약어 관리의 어려움, 프로그래밍 언어 확장시의 예약어와 식별자의 충돌

[정답] 4

3. 다음 중 좌순환적인 BNF 규칙은?
① <식> ::= <식> + <식>
<식> ::= <식> * <식>
② <식> ::= <식> + <식>|<항>
<식> ::= <항> * <항>|<수>
③ <식> ::= <식> * <식>
<식> ::= <식> + <식>
④ <식> ::= <식> * <식>|<항>
<식> ::= <항> + <항>|<수>

[정답] 2

4. 다음 중 비단말기호는?
① {}
② []
③ ()
④ <>

[해설] 비단말기호
각진 괄호 '<>'로 묶인 기호, 다시 정의될 대상

[정답] 4

5. 모호한 문법의 문제점은 무엇인가?
① 분석나무를 만들 수가 없다.
② 문맥 무관형 문법이 아니다.
③ 간결한 코드를 만들기가 어렵다.
④ 여러 분석나무 중에서 실제로 쓰이는 분석나무에 따라서 다른 의미의 코드가 만들어질 수 있다.

[정답] 4

6. 다음 중 형태가 순서도와 비슷하고 EBNF 선언과 대응시킬 수 있으며 사각형, 타원 등의 도형과 화살표로 구성된 것은 무엇인가 ?
① 단말도표
② 구문도표
③ 문맥자유 도표
④ 어휘분석 도표

[해설]
구문도표는 EBNF 방법 외에 구문에 대한 형식 정의하는 또 다른 방법이며, 순서도와 비슷하며 EBNF 선언과 바로 대응시킬 수 있다.

[정답] 2

제3장 변수, 바인딩, 식 및 제어문

1. 변수

선언문 또는 묵시적 선언으로 생성되며 식별자, 자료속성들의 집합, 하나 이상의 주소 그리고 자료 값들의 네 가지 요소로 구성되는데, 주소와 자료 값들의 관계는 변할 수 있음

2. 바인딩

(1) 개념

① 변수의 네 가지 요소에 값을 확정하는 것
② 프로그램 작성시 변수 속성은 변할 수 있으나 번역시간에 결정된 변수의 속성은 변할 수 없음
③ 일반적으로 변수 속성은 선언문을 통해 이루어짐
④ 변수의 주소란 변수값이 저장되는 기억장소 위치
⑤ 이름에 어떤 속성을 연결하는 과정

(2) 바인딩 시간

① 속성이 이름에 연결되고 계산되는 과정이 어느 시점에서 이루어지는 가에 따라 바인딩이 분류되는 시간
② 종류: 실행시간, 번역시간, 언어의 구현시간, 언어의 정의시간
③ 중요성: 효율성, 유연성

3. 선언

(1) 정의

실행 시 사용될 자료의 속성을 컴파일러 등에게 알려주는 프로그램 문장(바인딩을 제공하는 중요한 방법)

(2) 선언문의 목적

① 주기억장치 사용과 접근 방법의 효율성
② 주기억 장치 경영의 효율성
③ 정적 형 검사 가능

4. 할당문

(1) 정의

변수의 내용을 변경할 수 있는 연산

(2) 종류

단순 할당문, 다중목적 할당문, 조건 목적변수 할당문, 복합할당 연산자, 단항 할당 연산자, 식으로서의 할당문, 혼합형 할당형

(3) 식으로서의 할당문

① 식으로 혹은 다른 식에 포함된 피연산자로 사용가능
② 할당 연산자를 다른 이항 연산자와 유사하게 취급
③ 할당 연산자가 왼쪽 피연산자를 변경시키는 식 부작용을 가질 수 있음

④ 단점: 할당문을 식의 피연산자에 허용하는 단점으로 다른 종류의 식 부작용 유발, 프로그램 가독성 저하

(4) 혼합형 할당형
① 할당문의 양편 자료형이 서로 다를 경우
② 설계시 고려사항: 식의 타입이 할당된 변수의 형과의 동일성 검사, 두 형이 일치하지 않는 경우 묵시적 형변환의 사용여부

5. 상수 및 변수 초기화
① 상수: 변하지 않는 값을 갖는 변수의 사용을 지원하기 위한 개념
② 컴파일러가 쉽게 인식할 수 있으므로 프로그램의 신뢰성 증가
③ 식별자로 주어지며 프로그램 수행 중 변하지 않음
④ 주소를 가질 수 있지만 오직 한 개의 자료 값만을 갖기 때문에 내용이 변할 수 없음

6. 표현식, 조건문
(1) 표현식
1) 식의 개요
① 표현식: 하나 이상의 피연산자를 가지고 자료 값의 계산을 기술
② 식평가: 피연산자 값에 대하여 주어진 연산을 실행함으로써 이루어짐
③ 참조투명성: 프로그래밍 환경을 변화시키지 않고 오직 값만을 생성
④ 좌결합법칙: 동일 우선순위를 가지는 두 개의 연산자가 함께 이웃하였을 때 왼쪽 것을 먼저 수행

2) 논리조건
① 적용순서: 피연선자1 연산자 피연산자2와 같은 식의 평가에서 두 개의 피연산자를 평가한 후 결과를 얻기 위해 연산자를 적용하는 방법
② 단락 회로 평가기법: 컴파일러의 불일치를 제거하기 위하여 언어 설계자들은 언어 내부에 새로운 단락 회로 평가 기법 도입

(2) 조건문
① 정의: 조건에 따라 실행되는 부분이 달라질 때 사용하는 문장
② fortran 언어, 내포된 if 문, case 문, switch 문

7. 반복문
(1) 정의
① 한 개 이상의 문장을 0번 이상 반복하여 실행시키는 문장
② 필요성: 반복기능이 없으면 모든 동작을 순차적으로 기술, 복잡성 증가와 유연성의 저하와 가독성의 저하와 유지보수의 어려움이 있음

(2) 사용자 지정 반복
① 반복수행을 하려는 일련의 문장들을 괄호로 묶어서 단위화시켜 반복 수행시키는 것
② C/C++에서는 break 문, Continue 문

(3) 논리제어 반복문
① 초기 조건 검사 하고, 그 결과가 참이면 반복문 몸체를 한번 수행하고 다시 조건 검사를 행하는 과정 반복
② 조건문 결과가 거짓일 경우 반복문 영역을 벗어나 다음 문장으로 수행제어가 넘어감
③ while 문, until문, do-while 문, loop-repeat 와 exit

(4) 제어변수 반복문
① for 문: 반복제어(제어변수)를 사용하여 고정된 횟수의 반복 표시

<3장 출제예상문제>

1. 바인딩에 대한 개념으로 잘못된 것은?
① 변수의 네 가지 요소에 값을 확정하는 것
② 이름에 어떤 속성을 연결하는 과정
③ 일반적으로 변수 속성은 제어문을 통해 이루어짐
④ 변수의 주소란 변수 값이 저장되는 기억장소 위치

[해설]
③ 일반적으로 변수 속성은 선언문을 통해 이루어짐

[정답] 3

2. 바인딩 시간의 종류에 속하지 않는 것은?
① 번역 시간 ② 언어의 설계 시간
③ 실행시간 ④ 언어의 구현 시간

[해설] 바인딩 시간의 종류
- 실행시간: 변수의 값을 확정, 자료 구조에 기억 장소를 배당
- 번역시간: 구성은 컴파일시간, 링크시간, 로드시간, 종류로는 변수의 형, 자료구조의 형과 크기, 레코드의 각 항목들의 형 등을 확정
- 언어의 구현시간: 정수의 자릿수, 실수의 유효숫자 개수, 수의 기계 내에서의 표기법 등
- 언어 정의 시간: 혼합형 연산이 허용되는 연산에서 연산해야 될 두 피연선자의 형에 따라 어떤 형의 연산을 해야 되는지를 확정

[정답] 2

3. 다음 중 선언문의 목적으로 옳지 않은 것은?
① 주기억 장치 사용과 접근 방법의 효율성
② 주기억 장치 경영의 효율성
③ 정적 형 검사 가능
④ 번역기의 구현이 용이

[해설] 선언문의 목적
- 주기억 장치 사용과 접근 방법의 효율성: 프로그램 실행 동안 변하지 않는 자료 구조의 속성들을 한정해 주는 것, 컴파일러가 자료구조에 접근하기 위한 계산을 최적화함, 주기억 장치의 절약 및 프로그램 실행 시간 절약
- 주기억 장치 경영의 효율성: 자료 구조의 크기, 생성 시기, 소멸 시기 등을 번역시간에 알게됨으로써 보다 효율적인 기억장소 배당 기법 제공
- 정적 형 검사 가능: 잘못 사용한 자료형 등을 번역시간에 낮아낼 수 있어 프로그램의 신뢰성을 높일 수 있음

[정답] 4

4. 다음 중 동적 형 바인딩에 대해 바르게 설명한 것은?
① 프로그램 실행시에 프로그램의 외부 구조에 의해 결정된다.
② 프로그램 실행시에 실행 과정에 의해 결정된다.
③ 프로그램 번역시에 프로그램의 외부 구조에 의해 결정된다.
④ 프로그램 번역시에 실행 과정에 의해 결정된다.

[해설]
동적 형 바인딩은 실행 시간에 이루어짐

[정답] 2

5. 피연산자 값에 대해 주어진 연산을 실행함으로써 이루어지는 것은?
① 표현식 ② 참조 투명성
③ 좌결합 법칙 ④ 식 평가

[해설]
① 표현식: 하나 이상의 피연산자를 가지고 자료값의 계산을 기술하는 것
② 참조 투명성: 프로그래밍 환경을 변화시키지 않고 오직 값만을 생성하는 것
③ 좌결합 법칙: 동일 우선순위를 가지는 두 개의 연산자가 함께 이웃했을 때 왼쪽 것을 먼저 수행하는 법칙

[정답] 4

6. 상수에 대한 설명으로 틀린 것은?
① 상수는 변하지 않는 값을 갖는 변수의 사용을 지원하기 위한 개념이다.
② 식별자로 주어지며, 프로그램 수행 중에는 변하지 않는다.
③ 주소를 가질 수 있지만 오직 한 개의 자료 값만을 갖기 때문에 내용이 변할 수 없다.
④ 컴파일러가 쉽게 인식할 수 없으므로 프로그램의 신뢰성이 증가한다.

[해설]
④ 컴파일러가 쉽게 인식할 수 있으므로 프로그램의 신뢰성이 증가함

[정답] 4

7. 다음 중 실행 시 사용할 자료속성을 컴파일러 등에 알려주는 프로그램 문장으로 바인딩을 제공하는 중요한 방법은 ?

① 표현식
② 바인딩
③ 할당
④ 선언

[해설]
선언은 실행 시 사용될 자료의 속성을 컴파일러 등에게 알려주는 프로그램 문장이며 바인딩을 제공하는 중요한 방법이다.

[정답] 4

제4장 자료형

1. 자료형과 형 선언

(1) 자료형

① 정의: 객체의 집합과 이 객체의 실체를 생성, 작성, 소멸, 분해, 수정하는 연산의 집합
② 내장된 자료형: 정수형, 문자형, 논리형
③ 강 자료형: 자료형에 과한 모든 특성들이 컴파일 시간에 확정되는 프로그래밍 언어
④ 자료형의 구성원: 객체, 요소, 값

(2) 형 시스템

① 자료형을 정의하고 변수를 어느 특정한 자료형으로 선언하는 도구
② 변수의 많은 특성이 선언문에서 확정
③ 선언된 형 이름과 변수 선언은 이러한 일반적인 특성 가짐 -> 유지보수성을 증가시킴
④ 선언문의 변화를 통해 선언된 형 이름과 변수선언이 가지는 특성을 변화시킬 수 있음
⑤ 신뢰성 증가뿐 아니라 프로그램을 일기 쉽게 해줌

2. 단순형, 열거형

(1) 단순형

① 수치형: 스칼라형으로 정수 또는 실수 값 표현
- 다형성: 한 연산자가 속성은 같은데 피연산자의 자료형에 의존되어 실제 기계에서 다른 것으로 간주되는 것을 의미

② 논리형: 논리형 값의 영역은 두 개의 객체 즉 참과 거짓으로 구성(and, or, not, implies(imp), equivalence(equiv))
③ 문자형: 문자열 연산 수행 후 문자열 길이는 번역시간에 의해 결정될 수 없음(프로그램 언어 구현 어려움, C, java)

(2) 열거형

① 열거형에 대한 객체의 영역은 리스트로 정해주며, 연산은 동등 및 순서관계와 할당연산 허용
② 제공되지 않을 경우: 리터럴값에 정수값을 대응시켜 표현 -> 기독성 저하
③ 열거형을 정의하는 쉬운 방법: 새로운 자료형 정의하고 선언할 수 있는 기법과 그 선언된 변수가 취할 수 있는 리터럴값을 선언하는 기법 제공

3. 배열, 연상배열

(1) 배열

① 동질형 자료의 집합체
② 이름, 차원, 원소형, 첨자 집형과 범위 등으로 특징지음
③ 특정원소: 두 가지 수준의 구문기법에 의해 참조(집합체 이름+참조(인덱스)인 선택자
④ 배열이름[첨자_리스트] -> 원소
⑤ 배열형 자료형: 원소형과 첨자형 포함
⑥ 첨자범위: 초기에는 묵시적으로 검사되어야 한다는 것을 명시하지 않아 오류가 잦았으나 첨자범위 검사 제

공(신뢰성에 중요한 요소)
⑦ 정적배열: 첨자범위가 정적으로 바인딩되고 기억장소 할당이 정적(실행시간 전에)으로 이루어지는 배열
⑧ 명세표: 기억장소 사상을 구현하기 위하여 배열에 관한 정보를 갖는 명세표 사용, 배열이름과 원소의 형, 원소의 길이, 배열의 시작 주소, 각 차원의 하안과 상한으로 구성됨
⑨ 배열의 저장: 행우선, 열우선(fortran)
⑩ 초기화: 배열의 기억장소가 할당되는 시간에 그 배열을 초기화하는 수단 제공

(2) 연상배열
① 키(key)라 불리는 값에 의해 접근되는 순서를 갖지 않는 데이터 원소의 집합체
② Perl에서 연상배열 구현과정에서 원소가 해시함수를 통해 저장되고 추출되기 때문에 종종 해시라고 불림 (모든 변수 앞에 % 붙임)
③ exists 연산자: 피연산자 키가 해시의 한 원소인지 원소가 아닌지에 따라 참이나 거짓 반환
④ keys 연산자: 해시에 적용될 때 그 해시에 포함된 키로 구성된 배열을 반환함
⑤ values 연산자: 해시의 값으로 구성된 배열을 반환
⑥ each연산자: 해시의 원소쌍에 대해 반복수행
⑦ 원소 탐색이 요구될 때 배열보다 해시가 훨씬 효율적

4. 레코드, 포인터 자료형
(1) 레코드: 이질형 자료의 집합체
- 레코드형: 이질형 자료로 구성된 조직적 자료형

(2) 포인터 자료형
① 포인터: 어떤 객체에 대한 기억장치 주소참조
② 포인터변수: 객체를 참조하는 기억장치 주소 값으로 취하는 식별자
③ 힙(heap)은 객체가 동적으로 할당되는 기억장소 영역이며, 이 힙에 할당되는 변수를 힙 변수라 함
④ 무명변수: 이름이 없는 변수
⑤ 자료 항목 간에 다중관계를 선언할 필요성이 존재한다는 것이며 포인터는 하나의 자료가 많은 리스트에 동시에 연결되는 것을 허용함
⑥ 적성력 향상시킴

1) c/c++의 포인터
① 주소가 어셈블리 언어에서 사용되는 것처럼 사용될 수 있음
② 포인터 산술 연산 가능 -> 유연함
③ 메모리에 있는 거의 모든 변수를 가리킬 수 있음 (‘*’ : 역참조 연산, ‘&’ : 변수의 주소 생성하는 연산자)

2) 참조형
① 주로 함수 정의에서 형식매개변수를 위해 사용 -> 호출함수와 피호출함수간에 양방향 통신 제공
② 변수 이름 앞에 ‘&’ 표시: 별명(하나의 자료 객체를 서로 다른 이름으로 접근할 수 있을 때 그 이름들을 별명이라고 함)
③ 예시
- 포인터 사용 함수: void incr(*int*p){(*p)++;}

- 참조형 사용: void incr(int & aa) {aa++;}
- 함수의 호출을 포인터 사용의 경우: incr(&x); , incr(&a);
- 참조형 사용: incr(x); , incr(a);

5. 자료형

(1) 정적형 검사와 동적형 검사

① 정적형 검사: 번역시간에 수행
- 강 자료형을 요구하며 컴파일러에게 일관성 있는 형 검사를 허용하고 효율적인 목적 코드를 생성할 수 있도록 해 줌

② 동적형 검사: 실행시간에 수행
- 행하는 프로그래밍 언어에서도 변수가 선언될 수 있으나 자료형은 언급되지 않음
- 실행시간의 유연성이 커지므로 프로그램 작성 시 단순성을 증가시킴
- LISP, APL

(2) 수식기술

① 프로그래머는 혼합형 연산의 사용을 원하나 컴파일러는 형고정 연상을 제공하므로 식에 사용된 자료형의 형 변환 요구

② 형 변환: 주어진 자료형의 값을 다른 자료형의 값으로 변환하는 것
- 묵시적 변환: 강제로 컴파일러에 요구되어 자동으로 수행되는 자동변환(강제변환)
- 명시적 형 변환: 프로그래머가 명령문으로 요구한 형 변환(캐스트)

<4장 출제예상문제>

1. 번역기가 프로그램의 형 정보에 일관성이 있는지 조사하는 것은?

① 구조적 자료형
② 기본 자료형
③ 형 양립성
④ 형 조사

[해설] 형 조사
- 번역기가 프로그램의 형 정보에 일관성이 있는지 조사하는 것
- 형 양립성 : 두 형이 같이 쓰일 수 있는 지를 결정하는 규칙에 사용됨

[정답] 4

2. 강 자료형에 대한 설명으로 잘못된 것은?

① 자료형에 관한 모든 특성들이 컴파일 시간에 확정되는 프로그래밍 언어이다.
② 사용하는 모든 변수의 선언과 변수의 모든 자료형 정보를 미리 결정해야 한다.
③ 프로그램의 신뢰성, 유지 보수성, 가독성을 향상한다.
④ 서로 다른 특성을 갖는 객체를 분명하게 구별하며, 컴파일러가 특성 제한을 수행한다.

[해설]
④ 형 시스템에 대한 설명

[정답] 4

3. 포인터가 지시하는 기억장소를 참조하는 것을 무엇이라 하는가?
① 할당
② 히프
③ 현수 참조
④ 참조 제거

[정답] 4

4. 동적 형 검사와 정적 형 검사에 대한 설명으로 잘못된 것은?
① 동적 형 검사는 실행시간에 수행된다.
② 정적 형에는 LISP와 APL이 있다.
③ 정적형은 강 자료형을 요구하며 컴파일러에게 일관성 있는 형 검사를 허용하고 효율적인 목적 코드를 생성할 수 있도록 해준다.
④ 동적형 검사는 행하는 프로그래밍 언어에서도 변수가 선언될 수 있으나 자료형은 언급되지 않는다.

[해설]
② 동적 형 검사에 해당

[정답] 2

5. C++의 참조형(reference type)에 대한 설명으로 올바른 것은?
① 참조제거(dereferencing)가 항상 명시적으로 이루어진다.
② 참조형 매개변수는 이름에 의해 매개변수가 전달된다.
③ 참조형 변수는 변수 선언시에 초기치가 함께 부여되며 그 후에는 바뀔 수 없다.
④ 참조형이나 포인터형이나 선언 형식은 마찬가지이다.

[해설] 참조형
- C++에 있는 특수한 포인터 형으로 변수 선언 시에 초기치가 함께 부여되어야 하며 그 후에는 값이 바뀔 수 없는 상수 포인터
- 참조형 변수를 함수의 형식 매개변수로 쓰면 포인터형에 의한 것보다 간단히 주소에 의한 매개변수 전달의 효과를 낼 수 있음
- 주로 함수 정의에서 형식매개변수를 위해 사용 -> 호출함수와 피호출함수간에 양방향 통신 제공

[정답] 2

6. 하나의 자료객체를 서로 다른 이름으로 접근할 수 있을 때 그 이름들을 무엇이라 하는가?
① 별명(alias)
② 부작용(side effect)
③ 양립적 명칭
④ 인스턴스

[해설] 별명
하나의 자료 객체를 서로 다른 이름으로 접근할 수 있을 때 그 이름들을 별명이라고 함

[정답] 1

7. 프로그래밍을 단순화시키고 자료구조 구축의 처리에 유연성이 허용되지만 프로그래밍 실행시간을 지연시키는 것은 무엇인가 ?
① 연산자 중복 정의
② 형고정 연산
③ 혼합형 연산
④ 동적형 검사

[해설]
동적형 검사는 실행시간에 수행하며, 행하는 프로그래밍 언어에서도 변수가 선언될 수 있으나 자료형은 언급되지 않으며, 실행시간의 유연성이 커지므로 프로그램 작성 시 단순성을 증가시킬 수 있다.

[정답] 4

8. 자료형을 정의하고 변수를 어느 특정된 자료형으로 선언하는 도구로서 변수의 특성을 선언문에서 결정하는 역할을 하는 것은 무엇인가 ?
① 연상배열 기법
② 형 시스템
③ 혼합형 연산
④ 자료형

[해설]
형 시스템은 자료형을 정의하고 변수를 어느 특정한 자료형으로 선언하는 도구이며, 변수의 많은 특성이 선언문에서 확정된다.

[정답] 2

제5장 영역과 수명

1. 블록과 영역

(1) 개념

① 영역: 프로그램에서 사용되는 식별자가 의미를 가질 수 있는 범위

② 블록: 복합문 안에 변수, 부 프로그램, 레이블과 같은 지역 식별자를 선언하며 그 묶인 부분만이 복합문의 영역이며 블록이라 함

(2) Algol 60

① 복합문: begin-end를 사용하여 일련의 문장집합을 하나의 단위문장으로 표시하는 형태

② 블록: 복합문의 변수 영역으로 내부에서 식별자를 선언하여 새로운 프로그램 환경을 설정할 수 있는 특별한 언어구조 제공

(3) 변수

블록이 실행되는 동안 의미 있는 값을 지니며 프로그램의 수행제어가 블록을 벗어나면 할당되었던 기억장소가 회수됨

(4) C 프로그램

① 단일 함수에 단일 블록으로 구성

② 각 블록은 여러 개 블록을 내포할 수 있음

③ 스택 기반의 기억장소할당기법에서 지원

④ 정적 영역 규칙: 식별자의 영역이 번역시 결정(일반적인 컴파일러 언어)

⑤ 동적 영역 규칙: 실행시간에 영역이 결정됨

2. 정적 영역과 동적 영역

(1) 지역변수

정적 내포관계를 유지하는 블록 구조 언어에서 블록에서 선언된 변수와 형식 매개변수

(2) 전역변수 = 비지역 변수

블록을 내포하고 있는 외부 블록에서 선언된 변수

(3) 정적 영역

① 자유변수: 현 블록을 내포하고 있는 가장 안쪽의 바깥쪽 블록을 조사하고 그 블록에도 해당 이름이 선언되어 있지 않으면 또 다음 바깥쪽의 블록을 조사하는 작업을 반복하여 찾아서 영역을 결정함

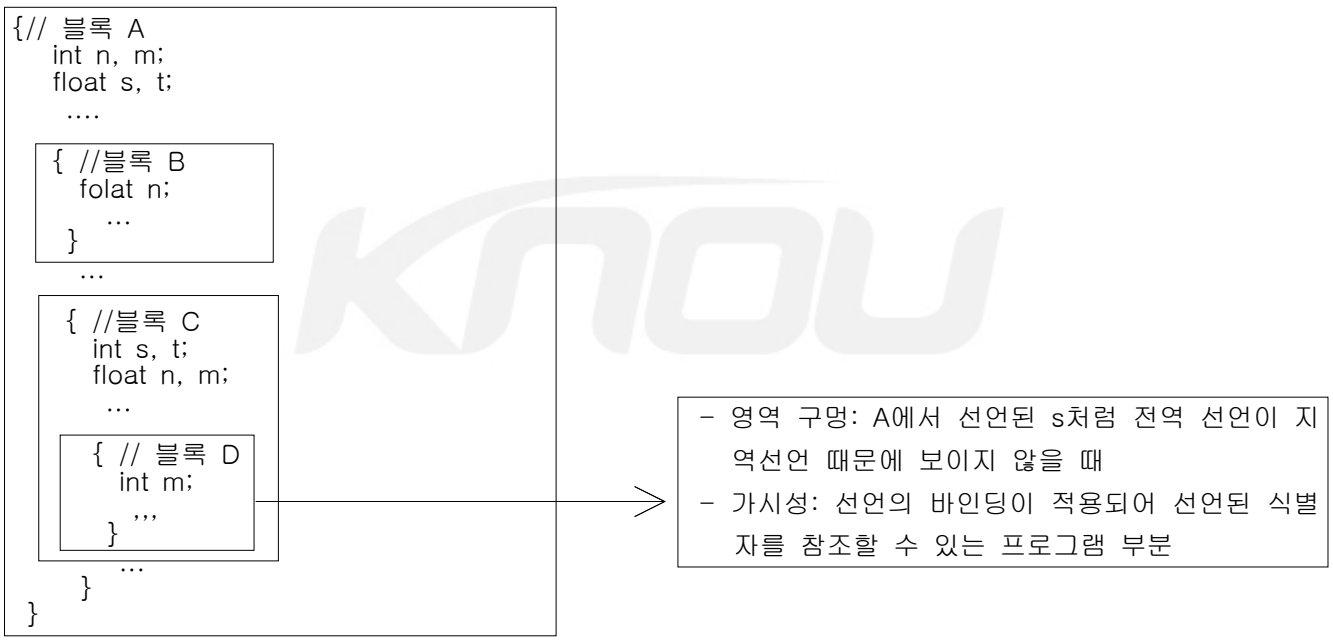
② 정적: 변역시 프로그램 문장만 조사하여 변수의 정의된 상태를 결정할 수 있는 것

(4) 변칙현상

① 정적영역 규칙을 따름으로써 생기는 현상

② 블록 C에서 n, m, s, t가 재선언 되어 블록 C에서 벗어날 때까지 이 변수는 블록 A에서 선언된 속성은 무효 (영역 구멍)

③ 블록 C에서는 s가 정수형으로 사용되고 블록 C 이외에서는 블록 A에서 정의한 실수형으로 사용



3. 언어에서의 영역

(1) Fortran

① 모든 변수는 명시적 혹은 묵시적으로 선언되고 선언된 프로그램 내에서 지역변수가 됨

② 지역변수 선언: COMMON문 사용

(2) 영역관계

① PL/1

- Algol 계열의 begin-end 블록 개념인 BEGIN-END 그룹 도입
- 모든 변수를 반드시 명시적으로 선언하지 않아도 되기 때문에 문제 발생

② Pascal

- 복합문을 위해 시작과 마침을 표시하는 기호로 begin-end를 사용하지만 영역 한정자로서의 begin-end 의미는 제거됨
- 오직 프로시저 함수만이 영역을 구분함

(3) 외부영역

① 어느 블록에서도 속하지 않고 모든 함수 전체를 영역으로 하는 부분

② C언어의 주프로그램 역시 프로그램의 다른 부분과 분리되는 자신의 지역영역을 갖는 하나의 함수이기 때문에 이러한 외부영역만을 가짐

```
int n, m;
float s, t;                  /* n, m, s, t 전역변수 */
void main(void)
{
      int n, w;              /* n, w는 main 함수의 지역적임 */
      float a,b,c;           /* a, b, c는 main 함수의 지역적임 */
      .....
}
```

(3) C++, Java

① 변수 정의가 함수의 어느 부분에서도 나타날 수 있음

② 변수 정의가 함수의 시작부 이외에 나타나면 그 정의문부터 함수의 끝까지 그 변수의 영역이 됨

③ for문의 초기화식에 제어변수가 새로이 정의되는 것 허용

(4) 블록 구조를 통한 영역의 개념

① 변수를 사용할 프로그램 문장 근처에서 선언하도록 하기 때문에 프로그램의 지역성 높여줌

② 프로그램 문장과 변수의 지역성은 프로그램의 효율적 수행에 도움이 됨

③ 프로그램의 구성을 단계적으로 세분화하는데 큰 도움이 됨

④ 루틴으로 구성된 표준 패키지를 프로그래머 프로그램에 결합시켜 하나의 프로그램을 만들기 쉬움

4. 변수의 수명

① 변수가 값을 저장하기 위해 기억장소를 할당받고 있는 시간을 의미

② Fortran: 모든 변수의 수명은 프로그램 수명과 같음

③ Algol 60: 지역변수들의 수명은 선언된 블록의 활성화에 의존하여 결정됨

(1) 속성

① C

- 자동할당방식: 주로 사용하는 변수를 선언하는 방법으로 변수 수명은 그 변수가 포함된 블록의 범위와 동일함
- 정적할당방식: 프로그램 시작 시 기억장호가 할당되며 프로그램 종료시 회수됨
- 프로그램 지정 할당방식: malloc()함수를 이용하여 기억장소 배정, free()함수를 호출하여 기억장소 회수

(2) 환경

① 지역단위로 묶여진 장소와 관련된 모든 식별자를 정의한 용어

② 지역변수, 진입점과 비지역변수에 접근하기 위한 정보 및 그 블록에서 선언된 프로시저와 레이블을 포함

③ 실행시간 동안 각 블록은 하나의 새로운 환경을 가짐

④ 선언문은 새로운 환경을 만들고 명령어는 새로운 저장을 의미

(3) 변수의 수명

① Algol: 영역(프로그램의 외형적인 구문과 관계), 수명(프로그램 실행과 관련)

② Fortran, Alogl, Pascal: 정적영역 규칙과 밀접한 관련성 가짐

③ Fortran: 변수 수명은 프로그램이 실행되는 전체 시간

④ Algol 60: 지역적 수명으로 새로운 블록이나 프로시저의 기억장소가 할당되고, 변수가 정의된 영역의 단위에서 벗어날 때까지 유효함

(4) 동적수명
① 프로그래머에게 기억장소의 수명 제어권을 부여하는 것
② Java, C++, Pascal: new 연산에 의해 동적으로 생성

(5) 기억장소 회수
① 기억장소를 실행사긴 내내 할당하다가 주프로그램 종료 시 회수 (예: Fortran)
② 기억장소가 다시 사용될 수 있도록 할당된 기억장소를 해제하는 명시적 명령어 제공(보편적 사용) (예: Pascal: dispose, PL/1:free)
- 문제점: 두 개 이상의 포인터가 동일한 기억장소를 가리킬 때 프로그래머가 이 사실을 잊어버리고 실수로 dispose를 사용하여 그 기억장소를 해제하면 허상참조 발생
③ 쓰레기 수집: 새 기억장소가 요구되는 경우 이들 기억장소를 회수하여 재생
- 방법: 기억장소를 계속 할당하다가 사용가능한 기억장소 풀ㄹ이 작아지면 자동적으로 사용하지 않는 기억장소를 모아 사용가능하도록 재생
- 문제점: 재생시간이 많이 필요하고, 그 재생이 예측할 수 없이 발생

5. C의 영역
(1) 영역단위
① 파일범위: C 소스 전체
② 함수범위: 함수가 선언되고 함수가 정의되는 부분
③ 블록범위: 일반적으로 중괄호 {}로 묶여진 모든 영역의 범위
④ C전역변수와 지역변수로 나누어 변수 범위 정함
⑤ 전역변수로 선언된 변수를 함수영역안에서 다시 선언하여 사용하려면 문제 발생
⑤ 전역변수로 선언된 변수를 함수 안에서 명시적으로 선언하여 사용할 수 있는 방법 제공

<5장 출제예상문제>

1. 정적 내포 관계를 유지하는 블록 구조 언어에서 블록에서 선언된 변수와 형식 매개변수는?
① 지역변수 ② 전연변수
③ 자유변수 ④ 비지역변수

[해설]
②④ 전역변수(비지역변수)는 블록을 내포하고 있는 외부 블록에서 선언된 변수
③ 자유변수는 현 블록을 내포하고 있는 가장 안쪽의 바깥쪽 블록을 조사하고 그 블록에도 해당 이름이 선언되어 있지 않으면 또 다음 바깥쪽의 블록을 조사하는 직업을 반복하여 찾아서 영역을 결정

[정답] 1

2. 영영관계에서 PL/1에 대한 설명으로 틀린 것은?
① 오직 프로시저나 함수만이 영역을 구분할 수 있다.
② 모든 변수를 반드시 명시적으로 선언하지 않아도 되기 때문에 문제점이 나타났다.
③ Algol 계열의 begin-end 블록 개념인 BEGIN-END 그룹을 도입하였다.
④ 블록에서 사용되는 대부분의 벼누가 지역변수로 사용되기를 원했지만 기대하지 않았던 결과가 초래되었다.

[해설]
① Pascal에 대한 설명

[정답] 1

3. 블록구조를 통한 영역의 개념으로 옳지 않은 것은?
① 변수를 사용할 프로그램 문장 근처에서 선언하도록 하기 때문에 프로그램의 지역성을 높여준다.
② 프로그램의 구성을 단계적으로 세분화하는데 큰 도움이 된다.
③ 루틴으로 구성된 표준 패키지를 프로그래머 프로그램에 결합시켜 하나의 프로그램을 만들기가 어렵다.
④ 프로그램 문장과 변수의 지역성은 프로그램의 효율적 수행에 도움이 된다.

[해설]
③ 루틴으로 구성된 표준 패키지를 프로그래머 프로그램에 결합시켜 하나의 프로그램을 만들기 쉬움

[정답] 3

4. 다음 중 C언어 변수 수명의 속성에서 프로그램 시작 시에 기억장소가 할당되는 방식은?
① 자동 할당방식 ② 정적 할당 방식
③ 프로그래머 지정 할당 방식 ④ 동적 할당 방식

[해설] 정적 할당 방식
- 프로그램 시작 시 기억장소가 할당됨
- 블록이 끝나더라도 기억장소 값은 그대로 유지됨
- 프로그램 종료시 회수됨

① 자동할당방식: C언어에서 주로 사용하는 변수를 선언하는 방법으로 변수수명은 그 변수가 포함된 블록의 범위와 동일함
③ 프로그래머 지정 할당방식: malloc()함수를 이용하여 기억장소를 배정하고, free() 함수를 호출하여 기억장소를 회수함

[정답] 2

5. 기억장소 회수 방법에 대한 설명으로 옳지 않은 것은?
① 기억장소를 실행사긴 내내 할당하다가 주프로그램 종료 시 회수한다.
② 기억장소가 다시 사용될 수 있도록 할당된 기억장소를 해제하는 명시적 명령어를 제공한다.

③ 쓰레기 수집방법의 문제점으로는 두 개 이상의 포인터가 동일한 기억장소를 가리킬 때 프로그래머가 이 사실을 잊어버리고 실수로 dispose를 사용하여 그 기억장소를 해제하면 허상참조가 발생한다.
④ 기억장소를 계속할당하다가 사용가능한 기억장소 풀이 작아지면 자동적으로 사용하지 않는 기억장소를 모아 사용가능하도록 재상한다.

[해설]
③ 기억장소가 다시 사용될 수 있도록 할당된 기억장소를 해제하는 명시적 명령어를 제공하는 방법의 문제점임. 쓰레기 수집방법의 문제점으로는 재생시간이 많이 필요하고, 그 재생이 예측할 수 없이 발생함

[정답] 3

6. 다음 코드를 보고 밑줄 친 부분의 출력 값은?

```
#include<stadio.h>

int n=10;
int m=20;
....
int main(void) {
     int n=11;
     extern int m;
     print('%d, %d₩n', n, m);
     function();
}

void function(void)
      exern int n;
      int m=22;
      print('%d, %d₩n', n, m);
}
```

① 11, 20　　② 10, 22
③ 10, 20　　④ 11, 22

[해설]
extern 예시문으로 전역변수로 선언된 변수를 함수안에 사용하는 방법을 나타내는 코드임. print('%d, %d₩n', n, m);의 출력 값은 11, 20이며, 아래 쪽의 print('%d, %d₩n', n, m); 출력 값은 10, 22임

[정답] 1

제6장 기억장소 할당

1. 정적 및 동적 기억장소 할당

(1) 정적 기억장소 할당

① 배열에 할당된 기억장소의 크기나 위치 등이 정적(번역시간에 확정)으로 결정되는 경우

② 기본조건

- 언어 설계에서 사용된 모든 배열은 확정된 고정 크기로 선언되어야 함
- 부프로그램은 재귀호출이 허용되지 않아야 함

(2) 동적 기억장소 할당

① 인터프리터 언어의 경우 동적 기억장소 할당이 필요

② 대부분 인터프리터 언어로 프로그래밍할 경우 매우 간결하게 처리되지만 컴파일러 언어보다 많은 실행시간을 요구함

③ 예: Snobol4, APL, LISP 등

(4) PL/1

① 정적 기억장소 할당과 동적 기억장소 할당의 가장 좋은 특징을 합친 기억장소 할당 기법 제공

- 정적기억장소할당: 정적 기억장소 할당이 가능한 변수에 대해서는 정적 기억장소를 할당
- 동적기억장소할당: 필요하다고 판단된 경우에 동적 기억장소 할당기법 사용

② 선언문

- AUTOMATIC: Algol 언어에서의 지역변수와 동일한 개념을 갖기 때문에 부프로그램의 진입에서 생성되었다가 반환될 때 회수
- CONTROLLED: 프로그램 실행시간에 ALLOCATE문으로 자료를 생성했다가 FREE문으로 회수하는 변수 선언 가능

2. 단위 프로그램의 실행

① 지역변수: 단위 프로그램이나 블록에서 선언하여 사용하는 변수

② 활성화 상태: 한 단위 프로그램의 실행 시작부터 종료까지의 시간

③ 단위 활성화: 실행시간에 한 단위 프로그램이 표현된 상태

- 코드부: 코정 크기로 내용이 프로그램의 실행동안 변하지 않음
- 활성화 레코드: 프로그램 실행에 필요한 모든 정보를 가지고 있으며 내용이 프로그램의 실행에 따라 변함

④ 참조환경

- 지역변수: 자신의 활성화 레코드에 할당
- 비지역변수: 사용가능한 비지역변수(다른 단위 프로그램의 활성화 레코드에 저장되어 있음)

⑤ 재귀호출: 한 프로그램에 대한 새 활성화가 그 프로그램의 이전 활성화가 끝나기 전에 새로 발생함

3. 정적 기억장소 할당

(1) ForTran 77

① 구성: 하나의 주프로그램과 몇 개의 부프로그램

② 각 단위 프로그램에서 지역변수들에 필요한 기억장소의 총합은 컴파일 시간에 결정됨

③ 변수영역: 변수가 선언된 단위 프로그램에 국한
④ COMMON: 전역변수 선언
※ 정적변수: 실행시간 전에 생성되어서 프로그램 실행 시간 전체를 변수의 수명으로 하는 변수

(2) 차감거리
단위프로그램을 컴파일 할 때 지역변수는 프로그램에서 발생한 순서대로 단위 활성화 레코드의 연속된 장소 차지

(3) 정적기억장소 할당
① 변역하는 동안 활성화 레코드에 대한 기억장소 위치가 결정되어 있지 않음
- 변역시간 마지막 단계인 프로그램 실행 전에 링크되어 적재할 때 확정되는 정적 기억장소 할당 기법 허용
② 각 단위 프로그램에서 요구하는 기억장소의 크기: 실행시간 전에 확정되며, 프로그램이 실행하는 동안 크기는 고정됨
③ 단순하여 매우 쉽게 구현 가능
④ 프로그래밍 언어 대한 유연성이 적음

4. 스택 기반 동적 기억장소 할당
(1) 블록 기반 언어 범주
① 블록: 전체 프로그램 실행 과정에서 특정 블록의 실행 차례까 되었을 때 활성화됨
② 부프로그램: 호출문에 의해 호출되었을 때 활성화됨

(2) 활성화 레코드의 크기가 정적으로 결정되는 경우
① 단위 프로그램이 활성화되는 시점에서 지역변수들이 생성되고, 지역변수들이 필요로 하는 기억장소 번역시간에 결정되는 경우
② 대표적 언어: 포인터 개념을 제외환 Pascal, 도적 기억장소 할당(calloc()과 malloc()함수)을 제외한 C언어
③ 준정적 변수: 활성화 레코드의 크기는 정적 바인딩하며 기억장소할당은 동적바인딩 하는 변수
- 효과: 재귀호출을 가능하게 하며, 스택 기반의 기억장소 할당을 수행할 수 있어 효율적인 주기억장소의 활용

(3) 단위프로그램의 전형적인 활성화 레코드 구조
① 동적체인: 현재 활성화된 활성화 레코드로부터 동적 링크를 추적한 연결로 단위 활성화의 동적 내포관계를 표현함. 즉 현 활성화까지 계속 호출된 활성화 레코드 체인을 의미
② 실행 중 스택에서 활성화 레코드들의 동적 링크 관계
- 한 활성화의 실행이 끝나면 해당 레코드 회수
- 새로운 활성화가 발생되면 주 기억장소에 새로 할당
- 호출 생성순서는 활성화 레코드가 발생되었다가 회수되는 개념
③ 활성화 되는 시점에서 활성화 레코드 결정 - 준동적 변수
- 단위 프로그램이 활성화되는 시점에서 지역변수들이 모두 생성되고, 지역변수가 요구하는 기억장소의 크기가 결정됨
- 번역시간에 단위 활성화 레코드에서 지역변수들의 차감거리가 상수 값으로 확정되지 못하여 주소에 대한 최정 확정을 실행시간까지 늦추어야 함

- 동적 변수와의 차이점: 한 번 바인딩되면 활성화된 프로그램이 종결될 때까지 크기가 변할 수 없음

④ 준동적 변수 할당순서

- 준정적 변수에 필요한 기억장소와 준동적 변수의 명세표에 대한 기억장소 할당
- 준동적 변수에 대한 기억장소의 실제 크기가 계산되면 그때 계산된 기억장소만큼 활성화 레코드를 확장시켜 할당함
- 그 주소를 먼저 할당한 명세표의 포인터에 바인딩함

(4) 활성화 레코드가 동적으로 변하는 경우

① 동적변수: 프로그램 실행 중에 새로운 자료 값이 생성되고 회수되어 활성화 레코드의 크기가 동적으로 변하는 경우로 프로그래머가 실행 중에 기억장소의 크기를 변화시키는 자료 값을 다룰 수 있음

- C, C++, Java, PL/1, Pascal, Algol68, Ada: 예로 힙 동적 배열 또는 유연성 배열이 있음. 이것은 프로그램 실행 중에 할당되는 자료들에 맞추어서 크기를 조절하는 배열

② 동적변수의 예

- 프로그램에서 변수 생성 시점을 정해주는 변수: PL/1, Pascal
- 보통 포인트로 연결되며, 연결리스트나 트리 구조를 구성
- 프로그램 실행 중에 노드들이 할당됨
- 프로그램 작성시 또는 번역시에는 실행 중에 발생되는 노드들의 개수를 알 수 없기 때문에 노드들에 이름을 부여한다는 것은 불가능

  예: new(P): [C의 경우 P=(t*) malloc(sizeof(t));]

  P를 t라는 레코드 자료형에 대한 포인터 선언

  t형의 레코드를 생성하여 그 주소를 P에 할당하는 명령문

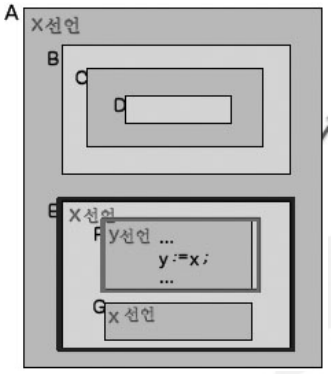
  수명: 생성 명령문이 선언된 블록의 실행이 끝난 후는 물론, 언어에 따라서는 포인터 P의 생존주기가 끝난 후에도 계속 지속됨

③ 힙과 스택 변수 - 힙 기억장소에 할당하는 이유

- 자료를 회수하는 문장을 제공하는 언어(C++, PL/1)
- 자료에 대한 포인터가 존재하는 동안만 자료가 존재하도록 하는 언어
- 힙 동적 배열을 사용하는 단위 프로그램과 이 배열을 선언한 단위 프로그램이 서로 다른 경우도 위와 같은 문제점 발생
- 동적 변수 자료들에게 기억장소를 스택방법으로 할당할 수 없기 때문
- 힙에 할당한 후 포인터 변수가 이를 가리키도록 바인딩함
- 스택변수: 준동적 변수/ 준정적 변수, 힙변수: 동적변수

(4) 비지역변수의 참조방법

① 변수 선언의 예



- y의 l-value: CURRENT+y의 차감거리로 번역 시간에 확정되어 사용
- x의 r-value의 값이 들어있는 위치는 x 이름으로 가장 최근에 할당된 스택 위치가 아님

② 정적 체인 사용기법: 모든 활성화 레코드에 정적 링크라고 부르는 포인터를 할당하고, 그 포인터로 하여금 작성된 프로그램의 정적 내포 관계에 있는 활성화 레코드를 가리키게 하는 방법임

※ 정적 체인: Current 활성화 레코드로부터 아래로 연결된 정적 링크의 순서, 비지역 변수에 대한 참조시 정적 체인을 따라 검색해서 먼저 발견된 변수 참조

(5) 디스플레이 사용기법

① 비지역 변수들의 자료값에 대한 참조 시간으르 줄이기 위한 구현기법

② 단위 프로그램의 호출과 반환 횟수에 비하여 비지역변수들의 사용이 상대적으로 증가할 경우 매력적인 방법

③ 정적 링크 대신에 실행 시간 어느 시점에서나 정적 체인관계를 디스플레이라고 부르는 1차원 가변 배열을 사용하여 유지

④ 어떠한 비지역 변수에 대한 참조시간이 동일하지만 이 경우에는 활성화 레코드가 할당되거나 회수될 때마다 디스플레이 내용을 변경시키는 일을 수행해야 함

⑤ 활성화 레코드에 변화된 일부 디스플레이 값을 저장하기 위한 기억장소를 요구함

<6장 출제예상문제>

1. 정적 기억장소 할당의 기본 조건에 대한 설명으로 틀린 것은?

① 언어 설계에서 사용된 모든 배열은 확정된 고정 크기로 선언되어야 한다.

② 부프로그램은 재귀호출이 허용되어야 한다.

③ 프로그램을 실행하는 동안 어떤 부 프로그램에 대해서도 부 프로그램이 요구하는 크기의 기억장소만 필요하다.

④ 실행시간에 하나의 주프로그램과 관련된 부프로그램이 요구하는 기억장소의 크기는 각 단위프로그램이 필요로 하는 기억장소의 총합을 넘지 않는다.

[해설]

② 부프로그램은 재귀호출이 허용되지 않아야 함

[정답] 2

2. 동적 기억장소 할당에 대한 설명으로 잘못된 것은?
① 인터프리터 언어의 경우 동적 기억장소 할당이 필요하다.
② 대부분 인터프리터 언어로 프로그래밍 할 경우 매우 간결하게 처리되지만 컴파일러보다 많은 실행 시간을 요구한다.
③ 배열에 대한 접근 코드를 더욱 효율적으로 작성이 가능하다.
④ 대표적이 예로는 Snobol4, APL, LISP 등이 있다.

[해설]
③ 정적 기억장소 할당에 대한 설명임. 정적 기억장소 할당은 배열에 할당된 기억장소의 크기나 위치 등이 정적(번역시간에 확정)으로 결정되는 경우를 말함

[정답] 3

3. 다음의 기술보고 고정된 크기의 활성화 레코드의 동적 할당기법의 중요한 효과로만 묶인 것은?

㉠ 재귀호출을 가능하게 한다. ㉡ 번역시간에 단위 활성화 레코드에서 지역변수의 차감거리가 상수 값으로 확정되지 못하여 주소에 대한 최종확정을 실행시간까지 늦추어 준다. ㉢ 활성화 레코드의 크기가 동적으로 바인딩되지만 한 번 바인딩되면 활성화된 프로그램이 종결될 때까지 크기가 변하지 않는다. ㉣ 스택 기반의 기억장소 할당을 수행할 수 있어서 주기억장소를 효율적으로 사용할 수 있다.

① ㉠㉡ ② ㉡㉢
③ ㉠㉣ ④ ㉡㉣

[해설] 고정된 크기의 활성화 레코드의 동적할당기법(준정적 변수)의 효과
- 재귀호출을 가능하게 함
- 스택 기반의 기억장소 할당을 수행할 수 있어 효율적으로 주기억장소를 사용할 수 있음

[정답] 3

4. 준동적 변수에 대한 설명으로 틀린 것은?
① 활성화 레코드의 크기가 동적으로 바인딩 된다.
② 단위 프로그램이 활성화되는 시점에서 지역변수들이 모두 생성되고, 지역변수가 요구하는 기억장소의 크기가 결정된다.
③ 번역시간에 단위 활성화 레코드에서 지역변수들의 차감거리가 상수 값으로 확정되지 못하여 주소에 대한 최정 확정을 실행시간까지 늦추어야 한다.
④ 한번 바인딩되면 활성화된 프로그램이 종결될 때까지 크기의 변화가 생긴다.

[해설]
④ 한번 바인딩되면 활성화된 프로그램이 종결될 때까지 크기가 변할 수 없음

[정답] 4

5. 힙 기억장소에 할당하는 이유로 틀린 것은?
① 자료를 회수하는 문장을 제공하는 언어이다.
② 활성화된 레코드와 함께 회수되지 않고 계속 더 유지할 수 있기 때문에 스택에 할당되는 활성화 레코드에 할당이 가능하다.
③ 동적 변수자료들에게 기억장소를 스택방법로 할당할 수 없기 때문이다.
④ 힙에 할당한 후 포인터 변수가 이를 가리키도록 바인딩한다.

[해설]
② 활성화된 레코드와 함께 회수되지 않고 계속 더 유지할 수 있기 때문에 스택에 할당되는 활성화 레코드에 할당이 불가능

[정답] 2

6. 디스플레이 사용기법에 대한 설명으로 잘못된 것은?
① 비지역 변수들의 자료 값에 대한 참조 시간을 늘리기 위한 구현기법d이다.
② 단위 프로그램의 호출과 반환 횟수에 비하여 비지역변수들의 사용이 상대적으로 증가할 경우 매력적인 방법이다.
③ 활성화 레코드에 변화된 일부 디스플레이 값을 저장하기 위한 기억장소를 요구한다.
④ 정적 링크 대신에 실행 시간 어느 시점에서나 정적 체인관계를 디스플레이라고 부르는 1차원 가변 배열을 사용하여 유지한다.

[해설]
① 비지역 변수들의 자료 값에 대한 참조 시간을 줄이기 위한 구현기법d이다.

[정답] 1

7. 다음 중 실행시간 전에 생성되어 프로그램 실행시간 전체가 변수의 수명이 되는 변수는 무엇이라 하는가 ?
① 동적 변수
② 정적 변수
③ 비지역 변수
④ 지역 변수

[해설]
정적변수는 실행시간 전에 생성되어서 프로그램 실행 시간 전체를 변수의 수명으로 하는 변수이다.

[정답] 2

제7장 부프로그램

1. 부프로그램의 개요

(1) 특성

① 각 프로시저는 단일 진입점을 가짐

② 호출프로그램은 호출된 프로그램이 실행되는 동안 호출프로그램의 실행은 중단됨

③ 부프로그램의 실행이 끝나면 제어는 호출자에게 돌아감

(2) 프로지서

① 명령형 프로그래밍 언어에 국한된 용어로 함수와 서브루틴을 구별하지 않는 부프로그램을 의미함

② 구성요소: 프로시저 이름, 매개변수 리스트, 몸체, 실행환경

③ 서부루틴: 결과값을 하나 이상의 매개변수에 할당하거나 자신의 환경을 변환하거나 또는 이 두 가지 일을 다 수행함으로써 주어진 목적을 완성하는 프로시저(예약어: procedure)

④ 함수: 하나의 결과값 만을 반환하는 프로시저로서 식에서 변수처럼 하나의 원소로 사용(예약어: function)

⑤ procedure 프로시저 이름(매개변수 리스트)
선언부(declarations)
end 프로시저 이름

(3) 함수값을 결정해 주는 방법

① 지역변수로 간주되는 함수 이름에 결과값을 배정

```
function gcd(m,n: interger) : interger;
  begin
  if n=0 then gcd :=m
         else gcd :=(n,m mod n);
  end
```

② C에서처럼 return 문장으로 결과값을 반환

```
int gcd(int m, int n)
{
   if (n==0)
         return(m);
   else
         return(gcd(n,m%n);
}
```

2. 매개변수 전달 기법

(1) 형식매개변수와 실 매개변수

① 형식매개변수: 부프로그램이 실행되는 동안에 호출자가 보내준 식 또는 다른 이름을 대신하여 그 프로그램에서 사용되는 이름을 의미

② 실 매개변수: 원래의 식 또는 이름으로 상수나 식의 사용여부는 프로그래밍 언어에 따라 다름

(2) 참조호출
① 실 매개변수들의 주소를 대응되는 형식 매개변수들에게 보내는 방법
② 대표적인 언어: Fortran, PL/1
③ 호출방법
- 호출프로그램은 사용된 실 매개변수가 식인지를 판가름하여 식이면 그 값을 계산해서 임시 기억장소에 저장함
- 필요한 경우 실 매개변수에 해당되는 주소를 계산함
- 실 매개변수를 참조할 수 있도록 그 주소들을 호출된 프로시저에 전달

(3) 값 호출(call by value) = 값 전달 기법
① 호출된 프로시저 자신이 형식 매개변수에 해당되는 기억장소를 별도로 유지하는 방법
② 실 매개변수값이 어떠한 경우라도 변하지 않으며 C, Pascal에서 기본적으로 사용
③ 호출방법
- 호출 프로그램이 실 매개변수에 해당되는 주소를 보냄
- 호출된 프로시저는 r-value를 구하여 자신이 보유하고 있는 형식 매개변수의 기억장소에 복사해서 다른 지역변수와 동일하게 취급하여 사용
④ 참조호출과 값 호출

	참조호출	값 호출
기억장소	-	형식 매매변수에 대한 기억장소가 추가로 요구됨
주소참조	값을 참조할 때마다 주소를 참조함	값을 참조할 때마다 주소를 참조하지 않음
값의 변화	원하지 않을 경우도 실 매개변수 값이 변화될 수 있음	실 매개변수 값이 전혀 변화되지 않음
독립성	부프로그램을 블랙박스로 간주해서 모듈화 하는데 방해됨	모듈간의 독립성이 높음

(4) 결과호출(call by result) - 결과전달기법
① 호출된 프로시저가 형식 매개변수 값을 저장할 기억장소를 보유함
② 호출방법
- 호출된 프로시저가 형식 매개변수를 지역변수로 취급하여 모든 연산 수행
- 호출 프로그램에서 반환하기 직전에 이 형식 매개변수의 값을 그에 대응되는 실 매개변수에 복사
- 지역변수인 형식매개변수의 최종 값을 반환하기 직전에 호출된 프로시저에게 실 매개변수에 복하는 끝맺음부를 첨부함으로써 가능

(5) 이름 호출(call by name)
① 형식 매개변수가 사용된 모든 자리에 실 매개변수를 그대로 복사한 것처럼 간주하여 사용하는 방법
② 다른 매개변수 전달기법에 비하여 구현이 난해하며, 이름 호출 기법을 사용한 프로그램을 읽기가 어려움

3. 부작용, 별명, 연산자 다형성

(1) 유해한 작용

① goto문, 포인터: 프로그래밍 언어에서 가독성을 저해하거나 프로그래머의 의도와는 다른 유해한 작용을 유발시키는 특징을 가지고 이씀

② 부작용, 별명: 프로그램에서 변수가 어떤 값을 갖게 될지 모르는 경우를 만들어 프로그램의 가독성을 저해

③ 연산자 다형성: 중복개념의 사용으로 유해한 작용 유발

(2) 부작용

① 국제표준: 실행에서 야기되는 간접적인 결과들을 모두 의미함

② 일반적으로 지역변수 이외의 변수값을 변화시키는 것으로 국한함

③ 비지역 변수의 접근에서 또는 매개변수의 전달기법들 중에서 참조전달기법과 이름전달기법에서 대표적으로 발생

④ 블록 중심언어: 현재 실행 중인 블록을 내포하고 있는 블록의 비지역변수를 사용할 때 나타남

(3) 부작용

① 프로그래밍 언어의 변수에 대해 그 변수명을 대체할 수 있는 다른 식별자

② 별명효과: 한 변수의 값을 변화시키면 자동적으로 동일 장소를 함께 사용하는 모든 변수의 값이 변한 상태

③ C언어의 경우 두 포인터 함수가 동일한 객체를 가리키는 경우 서로 다른 별명이라고 함

(4) 연산자의 다형성

① 중복정의

- 한 개체가 두 가지 이상의 개념으로 사용되었을 때
- 일반적인 표기에 대한 다양한 사용을 지원하기 위해 제공
- 예: + 연산자 -> 정수형과 실수형 등의 더하기 연산으로 사용자가 정의한 클래스의 객체에 대한 더하기 연산임

② 연산자의 다형성의 예

- Matrix operator*(const Matrix & m) X; C++에서 * 연산자를 행렬곱셈으로 중복 정의하고자 할 때

<7장 출제예상문제>

1. 매개 변수 전달 기법 중 호출된 프로시저 자신이 형식 매개변수에 해당되는 기억장소를 별도로 유지하는 방법은?

① 참조 호출 ② 값 호출

③ 이름 호출 ④ 결과 호출

[해설]

① 참조호출: 실 매개변수들의 주소를 대응되는 형식 매개변수들에게 보내는 방법

③ 이름호출: 식 매개변수가 사용된 모든 자리에 실 매개변수를 그대로 복사한 것처럼 간주하여 사용하는 방법

④ 결과호출: 호출된 프로시저가 형식 매개변수 값을 저장할 기억장소를 보유함

[정답] 2

2. 참조전달기법과 이름전달기법에서 대표적으로 발생하는 것은?
① 별명 ② 강압
③ 부작용 ④ 과부하

[해설] 부작용
- 지역변수 이외의 변수값을 변화시키는 것으로 국한됨
- 비지역 변수의 접근에서 또는 매개변수의 전달기법들 중에서 참조전달기법과 이름전달기법에서 대표적으로 발생
- 두 단위프로그램 사이에서 필요한 계산결과가 단지 매개변수를 통해서만 전달되기 때문에 판독성이 증가하며, 이 경우 값호출 기법을 사용해서 실 매개변수의 값을 받기만 하고 실 매개변수의 값을 변화시키지 않음으로써 프로그램 오류를 줄일 수 있음

[정답] 3

3. 다음의 설명으로 옳은 것은?
① 매개변수는 각 모듈 간에 자료 객체나 자료값을 넘겨주기 위한 언어 구성자이다.
② 실 매개변수는 변수가 아니라 프로시저가 호출되었을 때 실제 매개변수 역할을 대신하는 식별자 이름이다.
③ 형식 매개변수는 프로시저가 호출되었을 때 대응되는 형식 매개변수를 치환할 수 있도록 프로시저에 실제로 제공되는 변수와 식이다.
④ 서브루틴은 하나의 결과값만을 반환하는 프로시저로서 식에서 변수처럼 하나의 원소로 사용된다.

[해설]
② 실 매개변수는 프로시저가 호출되었을 때 대응되는 형식 매개변수를 치환할 수 있도록 프로시저에 실제로 제공되는 변수와 식
③ 형식 매개변수는 변수가 아니라 프로시저가 호출되었을 때 실제 매개변수 역할을 대신하는 식별자 이름
④ 서브루틴은 결과 값을 하나 이상의 매개변수에 할당하거나 자신의 환경을 변환하거나 또는 이 두 가지 일을 다 수행함으로써 주어진 목적을 완성하는 프로시저

[정답] 1

4. 참조 호출과 값 호출에 대한 설명으로 잘못된 것은?
① 참조호출은 값을 참조할 때마다가 주소를 참조한다.
② 참조호출은 부 프로그램을 블랙박스로 간주해서 모듈화하는데 방해된다.
③ 값 호출은 모듈간의 독립성이 높다.
④ 값 호출은 원하지 않을 경우도 실 매개변수 값이 변화될 수 있다.

[해설]
④ 참조호출에 대한 설명

[정답] 4

5. 다음은 연산자 다형성의 예시이다. 다음을 보고 설명한 것으로 잘못된 것은?

```
Matrix operator*(const Matrix & m) X;
```

① *가 두 개의 행렬을 매개변수로 취하여 그 결과로서 행렬을 반환하는 함수를 의미한다.
② 프로그매러가 A, B, C를 Matrix형으로 선언하여 “A=B*C"라는 문장을 사용하면 두 행렬 B와 C를 행렬 곱셈하여 행렬 A에 할당한다.
③ 명령문이 정수곱인지, 실수곱인지, Matrix 곱인지는 B와 X의 자료형에 따라 C++ 시스템에서 프로그래머가 직접 결정하여 수행한다.
④ 연산자를 주어진 연산 개념 이외에 새로운 의미를 사용자가 추가로 부여하는 것이 가능하면 연산자 중복정의를 허용한다고 한다.

[해설]
③ 명령문이 정수곱인지, 실수곱인지, Matrix 곱인지는 B와 X의 자료형에 따라 C++시스템이 자동으로 결정하여 수행함

[정답] 3

6. 다음 설명 중 틀린 것은 무엇인가 ?
① 함수 : 하나의 결과값을 하나 이상의 매개변수에 할당하거나 자신의 환경을 변환한다.
② 서브루틴 : 결과값을 하나 이상의 매개변수에 할당하거나 자신의 환경을 변환한다.
③ 이름 호출 : 실 매개변수의 주소를 대응되는 형식 매개변수에 보내는 방법이다.
④ 실 매개변수 : 프로시저가 실제로 호출되었을 때 대응되는 형식 매개변수를 치환할 수 있도록 프로시저에 실제로 제공되는 변수와 식을 말한다.

[해설]
함수는 하나의 결과값 만을 반환하는 프로시저로서 식에서 변수처럼 하나의 원소로 사용된다.

[정답] 1

제8장 추상자료형

1. 추상자료형의 개요

① 자료를 그 자료의 처리 연산과 함께 선택할 수 있어야 함
② 정보은닉 개념을 도입하여 프로그램을 쉽게 읽을 수 있고, 유지보수 용이하게 함
③ 도입목표: 수정용이성, 재사용성 향상, 보안성 향상

2. 추상자료형의 소개

(1) 추상화

① 개념: 속성들의 일부분만 가지고 주어진 작업이나 객체들을 필요한 정도로 묘사할 수 있는 방법을 지원하고 유사성을 표현하고 차이점을 삭제함으로 관련된 사항들을 하나로 묶어 표현하는 방법임
② 프로시저 추상화: 프로시저는 어떻게 수행되는가를 기술하지 않고 무엇이 수행되는가를 묘사함으로써 추상화시켜주는 프로세스 추상화 기법

(2) 자료추상화

① 자료형: 객체들의 집합과 이들 객체들에 적용되는 연산자들의 집합
② 자료추상화: 자료형의 표현과 그에 관련된 연산들을 함께 묶어 캡슐화하는 기법
③ 캡슐화: 사용자가 잘 정의된 방법으로 이를 호출하여 사용하는 것을 허락하는 방법 제공(공용부(public part), 전용부(private part), 공개한다(export), 도입한다(import))
④ 자료추상화의 예
 ㉠ queue 구조
  - structure: 자료 추상화 개념에 대한 새로운 이름
  - 시작: 예약어 structure
  - 영역의 한계: 예약어 end
  - structure 내부에 자료형의 이름, 각종 선언문들, 그리고 예약어 structure 이름으로 주어진 자료형의 표현과 일련의 프로시저문 정의

```
structure queue =                    // queue 정의의 시작
    operations ADDQ, DELETEQ, ISEMPTYQ
    representation... end rep        // 자료형 queue의 표현 정의
        procedure ADDQ(..)
          ...
        end ADDQ
        procedure DELETEQ(...)
          ...
        end DELETEQ
        procedure ISEMPTYQ(...)
         ...
        end ISEMPTYQ(...)
        procedure NEXT(...)          // 공개되지 않기 때문에 지역함수로만 사용
         ...
        end NEXT
        begin initialization code
        end
    end queue                        // queue 정의의 끝
```

(3) 큐를 추상화환 자료형

① 폐쇄 영역: 구조에서 정의된 이름들을 명시적으로 공개하지 않는 한 외부에서 사용할 수 없는 영역

② 추상자료형 구성요소: 자료형에 관련된 일련의 연산이름, 자료형의 표현, 연산들의 구현, 초기화를 위한 코드

3. C++의 추상자료형

(1) 개념

① C언어에 여러 특징을 추가하여 만들어진 언어

② 중요한 특징: 객체지향 프로그래밍을 지원하는 것(추상자료형 지원: 클래스 제공)

(2) 클래스

① 데이터 멤버: 클래스에서 정의된 자료

② 멤버함수: 클래스에서 정의된 함수

③ 한 클래스의 모든 인스턴스는 멤버 함수 집합 하나만 공유하지만 각 인스턴스는 각자 자신의 클래스 데이터 멤버 집합을 가짐

④ 인스턴스: 항상 객체선언 결과로 생성되며, 수명은 인스턴스화한 선언문의 영역에서 벗어날 때 끝남

(3) C++ 생성자 함수: constructor

① 객체를 생성할 때 필요한 매개변수의 제공과 초기화를 위하여 사용

② new 연산자를 사용하여 객체의 일부를 힙에 할당 가능

③ 클래스의 일부로서 클래스와 같은 이름을 가짐

④ 한 클래스는 한 개 이상의 생성자를 가질 수 있음

⑤ 클래스형 인스턴스가 생성될 때 묵시적으로 호출됨

(4) C++ 소멸자 함수: destructor
① 클래스 인스턴스의 구명이 끝났을 때 묵시적으로 호출됨
② 힙 변수의 객체 수명: 힙 기억장소를 할당받은 후부터 delete 연산자를 가지고 명시적으로 회수될 때까지의 기간
③ 스택 변수 클래스의 인스턴스는 생성자에서 new 연산자로 생성시킨 힙 변수 데이터 멤버 포함 가능함
④ 클래스 인스턴스에 대한 소멸자는 힙 변수 멤버의 기억장소를 회수하기 위하여 delete 연산자 포함할 수 있음
⑤ 소멸자 이름은 클래스 이름 앞에 (~)을 붙여서 사용

(5) C++ 매개변수 추상자료형
① 매개변수 추상자료형을 사용하여 좀 더 편리하게 추상자료형 이용
② 예: 어떤 스칼라형 원소만 저장할 수 있는 스택에 대한 자료추상화보다 여러 가지 다른 자료형을 위한 분리된 스택에 대한 자료추상화가 요구될 수 있음

(6) 틀 매개변수
① 클래스 틀로 만들 때 필요한 매개변수로 스택의 원소형을 사용할 수 있는 자료형
② 예: stack <int> stk(150);
- 클래스 틀 이름 다음에 나온 각괄호(<>) 속에 stack의 틀 매개변수인 원소의 자료형 선언
- int형은 stack이고 생성자에 스택 크기를 150으로 주었기 때문에 s사라는 스택은 150개의 정수형 원소를 갖도록 생성함

4. Java의 추상자료형

(1) 개념
① 클래스: 모든 사용자 정의 자료형
② 모든 객체는 힙에 할당되고 참조형 변수를 통하여 접근됨
③ 부프로그램(메소드)은 오직 클래스에서만 정의
④ private, public: 메소드와 변수 정의에 첨가될 수 있는 수정자로 간주

(2) 패키지
① 한 개 이상의 클래스 정의를 가지며 패키지 내에서 각 클래스는 다른 클래스의 부분 프렌드임
② 부분 프렌드: public이나 protected 선언 되거나 또는 접근 명시자가 생략되었으면 패키지 내의 클래스에서 정의된 개체는 패키지 내의 모든 클래스에서 가시적이라는 것. 즉 private 접근 수정자가 선언되지 않는 한 가시적임
③ 패키지 영역
- 접근 수정자가 없는 개체들은 패키지 내에서 가시적이므로 패키지 영역을 갖는다고 함
- 명백한 프렌드 선언을 가질 필요성이 적으므로 C++의 프렌드 함수 또는 프렌드 클래스를 포함하지 않음
- Java의 표준 클래스 라이브러리는 패키지의 계층구조에서 정의됨

<8장 출제예상문제>

1. 추상 자료형에 대한 설명으로 올바른 것은?
① 세부적인 구현 내용을 수혜자가 자유롭게 바꿀 수 있어야 한다.
② 자료구조뿐만 아니라 그것에 대한 연산도 함께 정의된다.
③ 자료구조를 수혜자가 융통성있게 바꿀 수 있도록 선언되어야 한다.
④ 자료형에 관한 구체적인 내용이 정의되지 않은 것을 말한다.

[해설] 추상 자료형
- 자료를 그 자료의 처리연산과 함께 선택할 수 있어야 함
- 정보은닉개념을 토입하여 프로그램을 쉽게 읽고, 유지보수를 용이하게 함
- 도입목표: 수정용이성, 재사용성 향상, 보안성 향상

[정답] 2

2. Java의 추상 자료형은 어떤 구조로 정의될 수 있는가?
① package ② procedure
③ task ④ cluster

[해설] 패키지
- C++는 그들의 유일한 캡슐화 구조인 클래스에만 의존하지만 Java는 클래스 상위 수준에 두 번째 캡슐화 구조인 패키지를 가짐
- 패키지는 한 개 이상의 클래스 정의를 가지며, 패키지 내에서 각 클래스는 다른 클래스의 부분 프렌드임

[정답] 1

3. 캡슐화에 대한 설명으로 틀린 것은?
① 사용자가 잘 정의된 방법으로 이를 호출하여 사용하는 것으로 허락하는 방법(윈도우)을 제공한다.
② 공용부는 윈도우를 통해서 외부에 알려주는 부분이다.
③ 전영부는 윈도우 캡슐화를 통해 보호하는 부분이다.
④ 도입한다는 다른 모듈 프로그램에서 비공개된 객체들을 사용할 수 있도록 연결하는 행위이다.

[해설]
④ 도입한다는 다른 모듈 프로그램에서 공개된 객체들을 사용할 수 있도록 연결하는 행위

[정답] 4

4. 다음 중 추상자료형의 구성요소로 옳지 않은 것은?
① 자료형의 표현 ② 연산들의 연결
③ 자료형에 관련된 일련의 연산 이름 ④ 초기화를 위한 코드

[해설] 추상자료형의 구성요소
자료형에 관련한 일련의 연산이름, 자료형의 표현, 연산들의 구현, 초기화를 위한 코드

[정답] 2

5. 다음은 queue 자료형 변수 선언의 예시이다. 다음을 보고 설명한 문장으로 잘못된 것은?

```
var x: queue
```

① 실행 시간에 이 선언문을 만나면 representation .. end rep에서 정의된 x의 기억장소가 할당된다.
② 이 구조 끝부분에 있는 초기화 코드가 수행된다.
③ 비지역 변수를 포함하지 않기 때문에 외부 환경에 영향을 준다.
④ 사용자는 operation 문으로 공개된 연산만을 사용해 queue 자료형의 구조에 접근한다.

[해설]
③ 비지역 변수를 포함하지 않기 때문에 외부 환경에 영향을 주거나 받지 않음

[정답] 3

6. c++의 소멸자 함수인 destructor에 대한 설명으로 잘못된 것은?
① 클래스 인스턴스에 대한 소멸자는 힙 변수 멤버의 기억장소를 회수하기 위해 delete 연산자를 포함할 수 있다.
② 스택 변수 클래스의 인스턴스는 생성자에게 new 연산자로 생성시킨 힙 변수 데이터 멤버 포함 가능하다.
③ 소멸자 이름은 클래스 이름 앞에 물결표(~)를 붙여서 사용한다.
④ 클래스의 일부로서 클래스와 같은 이름을 가진다.

[해설]
④ C++ 생성자 함수인 constructor 에 대한 설명이다.

[정답] 4

7. 다음 중 c++에서 객체의 일부를 힙에 할당할 때 사용되는 것은 ?
① 포인터
② 소멸자
③ 파괴자
④ 생성자

[해설]
C++ 생성자 함수는 객체를 생성할 때 필요한 매개변수의 제공과 초기화를 위하여 사용되며, 새로운 연산자를 사용하여 객체의 일부를 힙에 할당 가능하다.

[정답] 4