중요한 내용

6장 (1)

스트링 길이 선택 사항

- 정적 길이 스트링
  - 스트링 생성 시 그 길이가 설정되고 고정
  - Python 의 String, Java 의 String
- 제한된 동적 길이 스트링
  - 스트링 선언 시 고정된 최대 길이까지의 가변적인 길이를 갖는 것을 허용
  - C
- 동적 길이 스트링
  - 최대 길이 제한 없이 가변 길이를 갖는 것을 허용
  - Perl, JavaScript

사용자 정의 순서 타입 종류

- 열거 타입
  - 모든 가능한 값들이 그 정의에서 제공되는, 즉 나열되는 타입
  - enum
- 부분 범위 타입
  - 갑 범위가 순서 타입의 연속된 부분적 순서 로 정의되는 타입
  - subtype

6장 (2)

배열의 종류

- **정적배열**
  - 첨자 범위와 기억 공간 할당이 정적
  - c의 static 배열
- **고정스택 동적배열**
  - 첨자 범위가 정적으로 바인딩
  - 기억 공간 할당은 동적
  - c 의 함수 내부에 선언된 배열
- **스택 동적배열**
  - 첨자 범위와 기억 공간 할당이 동적
  - 변수 존속기간 동안 고정
  - Ada 의 배열
- **고정 힙-동적배열**
  - 첨자 범위와 기억 공간 할당이 동적
  - 힙으로부터 할당
  - 바인딩 이후 고정
  - c 의 동적 배열
- **힙-동적배열**
  - 첨자 범위와 기억 공간 할당이 동적
  - 힙으로부터 할당
  - 바인딩 이후 변경 가능
  - JavaScript 배열

6장 (3)

공용체란

- 공용체 (Union) 은 그 변수가 프로그램 실행 중에 다른 시기에 다른 타입의 값이 저장할 수 있는 타입

공용체 유형

- 자유 공용체 (Free Union)
  - 타입 검사를 지원하지 않음
  - 안전하지 않음
- 판별 공용체 (Discriminated Union)
  - 타입 검사 지원

6장 (4)

포인터 연산

- 배정
  - 배정은 포인터 변수에 어떤 유용한 주소로 설정
- 역참조 (Dereferencing)
  - 포인터가 가리키는 메모리 위치에 저장되어 있는 값을 참조

포인터의 문제

- 힙-동적 변수와 다른 변수들도 참조 가능
- 유연성이 높지만 프로그래밍 오류 초래 가능성을 높임

포인터의 문제 상세

- 허상 포인터(Dangling Pointer) 또는 허상 참조(Dangling Reference)
  - 이미 회수된 힙-동적 변수를 여전히 카리키는 포인터
  - 허상 포인터의 원인은 힙 동적 변수의 명시적 회수
- 분실된 힙-동적 변수(Lost Heap-Dyanamic Variables)
  - 사용자 프로그램에서 더 이상 불가능한 할당된 힙-동적 변수
  - 힙-동적 변수 분실을 메모리 누수(Memory Leakage)라 합니다.

허상 포인터 (허상 참조)

이미 회수된 변수를 가르키는 포인터

- 있지도 않을 걸 가르켜서 문제가 발생 -> 명시적으로 회수하기 때문에 문제가 생기는 거임

메모리 누수

- 분실된 힙-동적 변수->더 이상 접근 불가능 할당된 힙-동적 변수(쓰레기) 아무도 사용 안하기 때문에 메모리 낭비가 생겨 메모리 누수 발생

6장 (5)

타입 검사

타입 검사는 연산자에 대한 피연산자들의 타입이 호환가능하지를 확인하는 행위

강 타입

프로그래밍 언어에서 강타입은 타입 오류가 항상 탐지되는 것을 의미한다.

타입 동등

두 변수의 타입이 동등하면 한 변수가 다른 변수에 할당될 수 있다.

타입 동등을 정의하는 2가지 접근 방법

- 이름 타입 동등
  - 두 변수가 동일한 선언문에 속함
  - 같은 타입 이름을 선언
  - 장점 : 구현하기 쉬움
  - 단점 : 매우 제약적
- 구조 타입 동등
  - 두 변수 타입이 동일한 구조를 가지면 구조 타입 동등 이라 한다.
  - 장점 : 매우 유연
  - 단점 : 구현하기 어려움

7장

부작용 (사이드 이펙트 / Side Effect)

- Side Effect 란 무엇인가
  - 함수가 양방향 매개변수나 전역변수 변경 시 발생

- 단락 평가 (Short Circuit)
    - 결과에 필요한 최소한의 코드를 평가하고 나머지는 평가하지 않는 것
    - 논리 연산을 할 때에 앞 연산자의 결과에 따라 뒤 연산자의 실행여부가 결정되는 계산 방식
    - 식에 포함된 모든 연산자나 피연산자가 평가되지 않고서 식의 값이 결정
    - 앞의 연산만 실행하고 뒤의 연산을 실행하지 않음으로써 부작용 발생 가능
- Side Effect 문제점
    - 식에서 참조된 함수가 식의 다른 피연산자를 변경할 때 발생
    - 피연산자 평가 순서에 영향을 미친다.
- Side Effect 해결책
    - 함수의 부작용을 허용하지 않는 것
    - 양방향 매개변수, 비 지역변수 참조를 불허
    - 유연성 저하
    - 피연산자 평가 순서를 정하는 것
    - Java의 경우 왼쪽부터 오른쪽으로의 순서를 가짐

타입 변환 종류

- 축소 변환
- 확장 변환

축소 변환

축소 변환은 원래 타입에 속한 모든 값들의 근사치마저도 저장할 수 없는 타입으로 변환

확장 변환

확장 변환은 적어도 원래 타입의 모든 값들의 근사치를 포함할 수 있는타입으로 변환

혼합형 식

혼합형 식은 한 연산자가 다른 타입을 갖는 피연산자들을 갖는 식이며, 이를 허용하는 언어는 묵시적 피연산자 타입 변환을 허용

8장

선택문

- 양방향
- 다중 선택

반복문

- 계수기 제어
- 논리 제어
- 사용자 지정 루프 제어
- 데이터 구조에 기반한 반복문

9장 (1)

부프로그램의 기본 정의

- 부프로그램 정의
    - 부프로그램의 인터페이스와 동작을 서술
- 부프로그램 호출
    - 부프로그램이 수행에 대한 명시적 요청
- 부프로그램 머리부
    - 부프로그램의 첫 줄에 나타나며, 부프로그램의 인터페이스를 명세한다.
- 매개변수 프로파일(프로필)
    - 형식 매개변수의 개수, 순서, 타입이다.
    - 서명(Signature)이라 불리기도 한다.
- 프로토콜
    - 매개변수 프로파일과 반환값 타입(함수인 경우)이며,부프로그램에 대한 인터페이스를 명세
- 형식 매개변수
    - 부프로그램 머리 부상에에 나열한 매개변수
- 실 매개변수
    - 부프로그램 호출문에 사용되는 매개변수
- 부프로그램 선언
    - 부프로그램의 프로토콜만 제공하며, 몸체는 포함하지 않음

매개 변수

부프로그램이 처리하는 데이터

9장 (2)

매개변수 전달의 구현 모델

- **값 전달** (Pass by value)
    - 형식 매개변수 (Callee) 는 대응 실 매개변수 (Caller)의 값으로 초기화
    - 데이터의 이동 방식
    - 물리적 값 이동 (값 복사) -> 일반적
    - 접근 패스 전달
    - 장점 :
    - 스칼라 변수의 경우 연결 비용과 접근 시간이 빠르다
    - 부프로그램의 외부 데이터 접근 제한
    - 단점 :
    - 값 복사가 사용될 경우:
        - 형식 매개변수에 대한 기억공간 할당, 값복사에 따른 비용 부담
    - 접근 패스 전달의 경우:
        - 형식 매개변수에 대한 쓰기-보호 요구되고, 간접 주소 지정에 따른 접근 비용 부담
- **결과 전달** (Pass by result)
    - 출력 모드의 구현
    - 실 매개변수는 형식 매개변수에 값을 전달하지 않는다.
    - 피호출자 종료 전에 형식 매개변수의 값을 대응 실 매개변수로 전달
- **값-결과 전달** (Pass by value result)
    - 값이 복사되는 입출력 모드의 구현
    - 값 전달과 결과 전달의 혼합
    - 처음 : 형식 매개변수를 대응 실 매개변수의 값으로 초기화
    - 피호출자 종료 전 : 형식 매개변수의 값을 대응 실 매개변수로 전달
- **참조 전달** (Pass by reference)
    - 접근 패스 (주로 주소) 를 전달
    - 장점
    - 전달 과정 자체가 시간,기억장소 관점에서 효율적
    - 단점
    - 별칭
        - 실 매개변수간의 충돌
        - 배열 원소간의 충돌
        - 형식 매개변수와 가시적 비 지역변수 간의 충돌
        - 신뢰성 저하
- 이름-전달 (Pass by name)

9장 (3)

다형 부프로그램

- 다형 부프로그램 유형
    - 중복 부프로그램
    - 매개변수 부프로그램
    - 부타입 부프로그램
- overloaded subprogram
    - 중복 부프로그램 (overloaded subprogram) 은 같은 참조 환경에서 다른 부프로그램과 이름이 같은 부프로그램
    - int -> double , double -> int 허용
- parametric subprogram (매개변수 부프로그램)
    - 매개변수 부프로그램 (Parametric Subprogram)은 부 프로그램의 형식 매개변수에, 포괄형 매개변수를 갖는 부프로그램 제공
    - C++에서 매개 변수 다형성은 템플릿을 통해 구현할 수 있다.
- subtype subprogram (부타입 부프로그램)
    - 부타입 부프로그램은 타입 T의 변수가 T의 객체나 T로부터 파생된 임의 타입의 객체를 접근할 수 있음을 의미
    - 객체지향 프로그래밍 언어에서 지원

10장

단순 부프로그램 구현

부프로그램 활성화

부프로그램의 ARI를 생성하고 스택에 배치하는 것을 의미한다.

단순 부프로그램의 구성

- 변하는 부분 => 활성화 레코드
  - 지역변수
  - 매개변수
  - 복귀주소
- 변하지 않는 부분
  - 코드

행동

- 피호출자의 프롤로그 행동
  - 현재의 EP를 동적 링크로서 스팩에 저장하고, EP를 새로운 ARI의 기준 주소로 설정
- 피호출자의 에필로그 행동
  - 스택 TOP포인터를 현재의 EP-1로 설정하고, EP를 피 호출자 ARI의 동적 링크의 주소로 설정

용어

- **활성화 레코드 (AR / Activation Record)**
  - 컨텍스트 스위칭을 하기 전에 함수 상태를 기록하고 복원하기 위한 것
- **Dynamic Link**
  - 호출자의 ARI 기준(첫 주소)에 대한 포인터
  - 정적 영역 언어
  - 실행-시간 오류가 발생할 때 **추적 정보를 제공**하기 위해 사용된다.
  - 동적 영역 언어
  - 비지역 변수 **(전역변수)를 접근**하기 위해 사용
- **환경 포인터 (EP / Environment Pointer)**
  - 현재 실행중인 프로그램 단위의 ARI의 기준 주소를 가리킨다.
  - 부프로그램 호출시0 0현재의 EP는 새로운 ARI의 동적링크로서 저장된다.
  - EP는 새로운 ARI의 기준 주소를 가리킨다.
  - EP는 ARI에 포함된 데이터 내용을 접근하는 오프셋 주소의 기준으로 사용
- **Dynamic Chain**
  - 주어진 시점에서 스택에 포함된 동적 링크들의 집합
  - 실행의 현 시점까지 도달한 동적 History를 반영
- **local offset**
  - 활성화 레코드의 시작 시점으로부터 지역변수의 오프셋을 의미함
  - 지역변수의 오프셋은 ARI 시작 지점 (EP) 으로부터 (매개변수 개수 + 2) 의 위치부터 할당