프로그래밍 언어론

Assignment #4

전기컴퓨터공학부 정보컴퓨터공학전공

201524582 정희석

1. For an elementary data type in a language with which you are familiar, do the following:

=> Language: C => integer(int), real number(float), pointer

a. Describe the set of values that data objects of that type may contain.

1) integer(int type): -2,147,483,648(int MIN) ~ 2,147,483,647(int MAX)

2) real number(float type): -3.4×10^{38}~3.4×10^{38}

3) pointer: (32bit) 0x00000000 ~ 0xFFFFFFFF,

(64bit) 0x0000000000000000 ~ 0xFFFFFFFFFFFFFFFF

b. Determine the storage representation for values of that type (used in your local implementation of the language).

1) int type => ex) 452 => 0x000001C4

-> 00000000 00000000 00000001 11000100

2) float type => ex) 3.28 => 0x4051EB85 = 01000000 01010001 11101011 10000101

-> 0 1000000 10100011110101110000101

3) pointer => ex 32bit) *p = 0x40052338

-> 01000000 00000101 00100011 00111000

=> 0x40052338 주소를 가리킴

c. Define the syntactic representation used for constants of that type.

Int => literal: integer number ex) 24, 31, 458886

=> Named constant: TRUE, FALSE, #DEFINE CONSTANT ex) TRUE = 1, FALSE = 0

Float(double) => literal: real number ex)2.34, 4.8576411, 125.33574895

=> Named constant: PI, #DEFINED CONSTANT ex) M_PI = 3.14159265358979323846

d. Determine the set of operations defined for data objects of that type; for each operation, give its signature and syntactic representation on the language.

=>int, float, pointer

1) Arithmetic operations: +, -, *, /, %

2) relational operations: <, <=, >, >=, ==, !=

3) Assignment operations: = -> +=, -=, *=, /=, %=

4) Unary operations: ++, --

5) Bitwise operations: &, |, ^, <<, >>, ~

e. For each operation, determine whether it is implemented through software simulation or directly as a single-hardware instruction.

1) Arithmetic operations => single-hardware instruction ->ex) + : LOAD & ADD & STORE

2) relation operations => software simulation -> ex) == : LOAD & SUB -> result = 0

3) Assignment operations: software simulation => ex) STORE

4) Unary => single-hardware instruction -> ex) ADD, SUB

5) Bitwise => single-hardware instruction -> ex) AND, OR, NOT, LSL, LSR ...

f. Describe any attributes that a data object of that type may have other than its data type.

1) [int a] data type: signed int, size: 4byte, name: a

2) [float b] data type: signed float, size: 4byte, name: b

3) [pointer *p] data type: (hexadecimal) pointer, size: 32bit OS => 4byte, 64bit OS => 8byte, name: p

g. Determine if any of the operation symbols or names used to represent operations of the type are overloaded. For each overloaded operation name, determine when (compile time or run time) the specific meaning of each use of the overloaded name in a statement is determined.

=> in C language, operations

=> in C++ language, operations are overloaded and defined at compile time

=> ex) + -> I + I, F + F, P + P and P + I =P + (I * 32 or 64bit)

h. Determine whether static or dynamic type checking is used to determine the validity of each use of each operation of the type.

=> in c operation, declaration of operation and data object are static type checking. Because we include implemented operations.

2. For an elementary data type in a language with which you are familiar, do the following:

=> Language: C => integer(int), real number(float), pointer

a. Explain the difference among the type, variables of that type, and constants of that type.

-> int 와 float 의 type 에서의 차이는 정수와 실수를 다룬다는 점에서 다르다.

그리고 type 에서 variable 은 data object, constant 는 data value 이다.

b. Show a situation during execution where a data object of that type exists that is neither a variable nor a constant.

-> C 언어에서는 변수와 상수 모두 아닌 상태는 선언하고 초기화가 안된 상태를 말한다.

c. Explain the difference between data objects of that type and the values that those data objects may contain.

-> data object 와 data value 의 차이에서 data object 는 L-value(변수)로 값을 포함할 수 있고 data value 는 R-value(상수)로 변하지 않는 값이다.

3. For a language with which you are familiar and uses static type checking, give **two examples of constructs** that cannot be checked statically. For each construct, determine by running a test program whether the language implementation provides dynamic type checking or leave the construct unchecked during execution.

You use C as a language.

=> example) union & void *이 있다.

```
#include <stdio.h>

Typedef union var{

    int i_var;

    char c_var[4];

}Variable;

Int main(){

    void* a; //type isn't determined

    Variable b; //union type isn't determined ->int vs char array

    b = "asdf"; //select to char array

    a = &b; //a has union's address

    printf("%s is character array in union, %d is integer in union₩n",a.c_var, b.i_var);

    return 0;

}
```

4. For a language that provides a pointer type for programmer-constructed data objects and operations such as new and dispose, which allocate and free storage for data objects, **write a program segment** that generates garbage (in the storage management sense). Write a program that generates a dangling reference. If one or the other program segment cannot be written, explain why.

You can find an example in C programming language

```
#include <stdlib.h>

Void Make_Garbage(){

        int* val1 = malloc(sizeof(int));

        int* val2 = malloc(sizeof(int));

        *val1 = 42;

        *val2 = 24;

        val2 = val1;

        free(val1);

}
```

=> 위의 상황은 int 형 포인터 val1 과 val2 를 각각 선언 및 공간 할당을 진행한 후 각각 값을 assign 하고 val2 포인터가 가리키는 주소를 val1 의 주소로 함으로써 val2 가 기존에 가리키던 주소의 공간이 garbage 가 되며 val1 을 free 로 공간할당을 해제함으로써 val1 과 val2 가 가리키는 주소에는 아무것도 남지 않게 되는 허상 포인터로 만드는 프로그램이다.