HDC Language Reference

팀명 : hdcon

구성원 : 이한결,이동기

2016.11.26

1 Synopsis

1.1 Language

HDC 는 C 언어를 기반으로 하여 만들어졌다.

기존 C 문법 중 HDC 에서 사용할 수 없는 문법은 다음과 같다.

- dowhile, switch 는 지원하지 않는다. (while, for loop 만 지원)
- typedef
- 구조체는 다음과 같이 사용해야 한다. struct NAME {…};, 즉 typedef struct {}, struct {…}NAME; 형태를 지원하지 않는다.
- type 은 char, int, byte, void 만 지원 가능하다. short, long, float, double type 은 지원하지 않는다.
- 묵시적, 명시적 type casting 은 불가능하다. type casting 을 위해서는 따로 함수를 작성해서 사용자가 구현하여야 한다.

1.2 Special Features

A : 연산자

- Python 에서 배열 접근하는 방식인 : 연산자를 C 에 추가한다.

  따라서 A[:-1] 과 같은 연산이 가능하다.

B For in loop

- C 에서 제공하는 for loop 이 외에 for in loop 를 지원한다.

C 배열에서 연속된 assign

- 배열에서 한번에 하나의 원소에만 assign 이 가능한 기존 C 에서 연속된 여러 원소에 한번에 assign 이 가능하도록 지원한다.

D Byte 형의 한 bit 씩 조작할 수 있다.

- 기존 C 언어에서는 하나의 데이터 형에서 bit 하나하나를 조작하는 것이 매우 불편하다. 이를 byte 형의 한 bit 씩을 조절할 수 있게 해주는 기능이 추가되었다.

E Byte 배열의 중간부분을 하나의 변수처럼 사용할 수 있다.

- 기존 C 언어에서는 배열의 형태에서 일부만을 따로 변수로 잡을 수 없었다. 이 언어에서는 예를 들어 8byte 의 byte 형이 있다면 2 번째부터 6 번째까지 4byte 를 따로 잡아서 쓰는 것이 가능하다.

2 Lexical Issue

2.1 Data Type

A Scalar type

- integer, character, byte

B Vector type

- vector, Scalar type token 뒤에 [] 형태(Array)

2.2 Method

A 메소드는 하나의 Data type 을 반환 한다.

B 모드 메소드는 파라미터로 모든 Data Type 을 가질 수 있다.

2.3 Branch

A IF – ELSE 구문

2.4 Iteration

A For, While 구문

2.5 Operator

A Infix 연산자 : +,-,*,/,%

B 배정연산자 : =,+=,-=,*=,/=,%=

C 논리연산자 : !, &&, ||

D 관계연산자 : ==,!=,<,>,<=,>=

E 증감연산자 : ++,--

F 배열접근연산자 : :

G 비트연산자 : <<,>>,&,~,^,|

2.6 Comments

A //

- // 뒤부터 개행문자를 만날 때까지 컴파일러는 해당 문자 무시

B /* .. */

- /* 에서부터 */ 를 만날 때까지 컴파일러는 해당 문자 무시

3 Grammar

3.1 Symbol

- [x] : x 가 0 번 혹은 1 번 나올 수 있음
- x^* or { x }: x 가 0 번 이상 나올 수 있음
- x^+ : x 가 1 번 이상 나올 수 있음
- x | y : x 혹은 y 가 1 번 나올 수 있음

3.2 Grammar

- Abstract Syntax

```
program -> {type ID '(' funParam ')' statements } 'int' 'main' '('')' statements
statements -> '{'{ varDeclaration}{(expressionStmt|loopStmt|ifStmt|returnStmt|ioStmt)
';' } '}'
varDeclaration -> type ID';' | type ID'['NUM']' ';'

expressionStmt -> var assignOp expression | var,var = '{'NUM','NUM'}' | var
        | expression relOp expression | expression binOp expression | uOp expression
        | '('expression')' | funCall |
binOp -> '+' | '-' | '*' | '/' | '%' | '&' | '|' | '^'
uOp    -> '-' | '++' | '--' | '~' | '<<' | '>>'
funCall -> ID '(' arg ')'
arg -> (var|NUM) { ',' (var|NUM) }
relOp -> '<' | '>' | '<=' | '>=' | '=' | '!='
assignOp -> '+=' | '-=' | '*=' '/=' | '%=' | '='
var ->   ID | ID'['NUM']' | ID'['']'?NUM? ':' ''?NUM?']' | ID'.'dotOp | ID '['selector']'
selector -> '[' num [,num] ']' [ .num ]
loopStmt -> 'for' '(' expression ';' expression ';' expression ')' statements
     | 'for' '(' expression 'in' var ')'
       | 'for' '(' expression ',' expression 'in' var')'
  | 'while' '(' expression ')' statements
ifStmt -> 'if' '(' expression ')' statements { 'else if' '(' expression ')' statements }   ['else'
statements]
ioStmt -> 'hin' '>>' expression {'>>' expression} | 'hout' '<<' expression {'<<'
expression} {'<<' 'endl'}
funParam -> param {',' param}
param -> type ID | type ID'['']' | empty
dotOp -> 'first()' | 'second()' | 'reverse()' | 'length()' | 'push_back('NUM')' |
'delete('NUM')'
returnStmt -> 'return' | 'return' expression
type   -> (int|void|char) | 'pair''<'(int|void|char),(int|void|char)'>'
| 'vector''<'(int|void|char)'>' | 'string'
```

4 Semantics

4.1 타입

현재 HC 제공하는 타입은 다음과 같다.

ScalarType : int, char, byte, void

VectorType : vector (vector<int or char or byte>)

4.2 연산자

대입연산자는 +=,-=,*=,/=,%=,= 만 지원한다.

5 Sample Program

```
void createHeader(int flag, byte simpleHeader[]) {
  int counter;
   simpleHeader[0,3]++;    //ACK Number
   simpleHeader[4,7] = flag;  //flag
   simpleHeader[8,14]++;   //Data Number
   /*
    각각의 bit가 의미를 가지는 경우
    */
   simpleHeader[15].1 = 1;
   simpleHeader[15].2 = 0;
   simpleHeader[15].3 = 1;
   simpleHeader[16,17] = 5; //other flag


   for(counter; counter < 9; counter++) {
      simpleHeader[counter:counter+1] += 1;
      simpleHeader[18,19] = simpleHeader[18,19] +
simpleHeader[counter,counter+1];
   }//일부 바이트를 정수형처럼 쓸 수 있다. checksum계산을 표현했다.
   simpleHeader[18,19] = ~simpleHeader[18,19];
}

int calculateChecksum(byte header[]) {
   int output;
   int counter;
   byte[2] checksum;

   for(counter; counter<9;,  counter++) {
     //:연산자를 통한 배열에서 연속된 assign이 가능한 동작 표현
       header[counter:counter+1] += 1;
checksum[0,1] = checksum[0,1] + header[counter,counter+1];
   }
   checksum[0,1] = ~checksum[0,1]

   if(checksum[0,1] == header[18,19]) {
      output = 1;
   }
   else {
      output = 0;
   }
```

```
    return output;
}

int main() {
    int flag;
    byte it;
   byte minValue = 255;
    byte[50] header;
    header = createHeader(flag);
    if(calculateChecksum(header)) {
        return 0;//체크섬 확인
    }
    else {
        return 1;//체크섬 실패
    }
for (it in header[20:-1]) { //header의 20번째 원소부터 마지막 원소중에서
가장 작은 값을 출력한다.
        if (it < minValue) minValue = it;
    }
    hout << minValue << endl;
}
```

6 Remarks

A. 입출력은 기존 C 의 scanf/printf 를 사용할 수 도 있다. 하지만 HC 에서는 string type 의 입출력을 편리하게 하기 위해 C++의 입출력 방식인 cin/cout 를 참고하여 hin/hout 으로 지원한다.

B. main 함수는 함수들 중 가장 마지막에 선언되어야 한다. 이는 Grammar 를 편리하게 하기 위해 이렇게 두었다.

7 Reference

[1] Louden, Kenneth C , 컴파일러 제작: 원리와 실제, 2009

[2] https://docs.python.org/2.7/reference/index.html - python 2.7 reference

[3] http://en.cppreference.com/w/c - c reference