컴파일러 언어 설계 과제 보고서

Oh 조

권성철(200924413)

조용래(201224540)

1. Synopsis

Tiny Set Language(TSL)은 수학에서의 집합 개념을 언어의 기본 기능으로 넣어 이후의 추가적인 개념을 프로그래머가 자유롭게 만들 수 있게하는 것을 목표로 한다. 수학적으로 엄밀하게 만들어지는 것은 아니지만 이것을 이상으로 삼고 그 외형을 흉내내는 데 일차적인 언어 설계의 목적을 둔다.

1.1 입출력

표준 입출력만 제공하며, 표준 입력은 "cin<type> >> 입력대상", 표준 출력은 "cout << 출력대상"으로 정의한다.

1.2 타입

C언어에서 기본적으로 지원하는 정수형(int)와 문자형(char) 타입을 포함하며 논리형(bool)이 있다. 추가적으로 집합과 튜플 개념을 언어 차원에서 지원하고 싶어서 set과 tuple도 추가 시켰다. intset, charset, inttuple, chartuple, bitset을 지원한다. 배열 대신에 set과 tuple이라는 개념을 도입해서 데이터를 다루게 한다. 이름이 암시하듯이 기본적인 수학적 연산이 프로그래밍 언어 차원에서 지원된다. bitset 타입은 비트 조작을 원활하게 지원한다.

Element Type	Set	Tuple
int	intset	inttup
char	charset	chartup
bool	bitset	

<표 1> Type

1.3 제어문

C언어 기반의 제어 구문을 포함한다(If~else, while, for, do while, switch). 순서쌍을 가지는 tuple에 대해서는 전통적인 C스타일의 순회 방식을 따르지만 set 타입은 순서가 없기 때문에 루프를 돌릴 수 없다. 수학에서는 집합 안의 특정한 원소 집합을 가리킬 때 조건제시법을 사용한다. 부분 집합 역시 순서가 없지만 순회할 수 있기 위해선 순서가 필요하다. 새롭게 고안된 순회 방법은 집합에 대해 적용된다. 구문은 다음과 같다.

touch(set; condition; sort-criteria) body

집합 set에서 부분집합을 추려내기 위해 condition에 뽑아 내는 함수를 쓴다. Condition에 들어가는 함수를 추출함수, sort-criteria에 들어가는 함수를 정렬 함수라고 이름 붙인다. 추출함수와 정렬 함수는 조건에 맞는 원소를 뽑아내므로 리턴 타입이 bool로 고정되어 있다. sort-criteria는 C에서 제공하는 기본 연산자를 일부 조합하여 순서 기준을 만들어 낸다. 예를 들어 집합 A ={2,3,4,7,8} 이고 추출 함수가 bool func1(int a) {if(a % 2 == 0) return true;}, 정렬 함수가 bool func2(int a, int b) {if(a < b) return true;} 일때, "touch(A, func1, sort1) body" 구문은 집합 A의 부분 집합 A' = {2,4,8}

을 오름차순으로 정렬했을때 A″ = (2,4,8)에 대해 2,4,8 순으로 순회를 한다. 순회 하면서 하는 일은 body 구문에 적어준다. 이때 A″는 임시적으로 생성되고 이 구문을 벗어나면 사라지게 되므로 나중에 사용하기 위해선 복사해두어야 한다. body안에서 A″에 대한 접근은 tempobj라는 객체로 접근 가능하다. 자세한 사항은 문법 파트에서 다시 설명한다.

2. Lexical Issues

TSL로 짜여진 프로그램은 다음과 같은 언어 토큰들로 구성 된다.

2.1 키워드(Keyword)

키워드는 언어 상에서 이미 용도가 정해져 있기 때문에 변수나 함수 이름으로 사용할 수 없다. 키워드 목록은 <표 2>와 같다.

Keyword	Category	Etc
Int	Type	Basic type
Char	Type	Basic type
Bool	Type	Basic type
Intset	Type	Derived Type
Charset	Type	Derived Type
Inttup	Type	Derived Type
chartup	Type	Derived Type
bitset	Type	Basic Type
If	Control	Selection
Else	Control	Selection
While	Control	Iteration
For	Control	Iteration
touch	Selection + Iteration	

<표2> Keyword List

2.2 명칭(Identifier)

프로그래머가 직접 만들어서 사용한다. 변수나 함수는 고유한 이름을 가져야 하고 일정한 형식을 갖춰야 한다. 명칭은 글자와 숫자로 이루어진다.

Letter -> [A-Za-z]

digit -> [0-9]

id -> {letter}({letter} | {digit})*

2.3 상수(Constant)

변수와 반대되는 개념으로 고정된 값을 가지는 식이다. 상수식은 Basic type에만 적용된다. 3이나

'c' 또는 ture/false 같은 식은 상수로 취급된다.

정수상수 -> {digit}+
문자상수 -> [^₩' [:alnum:]₩'$]
문자열상수 -> [^₩"[:alnum:]+₩"$]
논리형상수 -> true | false

2.4 연산자(Operator)

정수는 C에서 제공하는 기본적인 사칙연산의 대상이 될 수 있다. 비트셋이나 튜플은 새로운 의미가 추가 된다. 집합에서의 이항연산자 '+'와 '-'는 합집합과 차집합을 의미한다. '+'의 한쪽에 집합이 있을 경우 의미가 달라지는 셈이다. 만약 집합 A, 정수 a에 대해 A+a라는 식이 있다면 이 식은 A에 원소 a를 추가시킨 새로운 집합을 리턴한다. 하지만 a - A같은 식은 비논리적이므로 틀린 문법으로 정의 된다. 적법한 문장식은 다음과 같다.(목록만 나열하였음)

expression ->

| int + int | int – int | int * int | int / int | int % int
| intset + int
| intset – int
| intset + intset
| intset - intset
| int + intset
| charset + char
| char + charset
| charset + charset
| bitset and bitset
| bitset & bitset
| bitset or bitset
| bitset | bitset
| bitset xor bitset
| bitset bic bitset
| ~ bitset
| id = expression

Op -> '+' | '-' | '*' | '%' | '/' | '&' | '|' | AND | XOR | OR | '~' |

(각 연산자가 적용될 수 있는 타입과 그렇지 않은 타입을 주의할 것)

2.5 구두점(Punctuator)

Punctuator -> ';' | ',' | '(' | ')'

2.6 공백(White Space)

스페이스바, 탭, 개행 코드 등이 있다. 변수를 선언할 때 타입과 명칭을 공백으로 구분한다.

2.7 주석(Comments)

/* 로 시작해서 */ 로 끝나는 것. 그리고 // 뒤에 있는 것들은 컴파일러에 의해 무시된다.

3. Grammer

3.1 expression : expression + term
| expression – term
| term
| assignment_expression
| equality_expression
| logical_expression;
;

3.2 Term : term * factor
| term / factor
| term % factor
| term
;

3.3 factor : id
| int
| intset
| char
| charset
| bitset
| tuple
| '(' expression ')'

3.4 assignment_operator
: '='
| '*='
| '/='
| '%='
| '+='
| '-='
| '&='
| '|='
;

3.5 assignment_expression

: ID assignment_operator expression

3.6 logical_expression

: or_expression

| and_expression

| inclusive_or_expression '|' exclusive_or_expression

| exclusive_or_expression

|

3.7 or_expression

: logical_and_expression

| logical_or_expression OR_OP logical_and_expression

| and_expression

3.8 and_expression

: inclusive_or_expression

| logical_and_expression AND_OP inclusive_or_expression

;

3.9 inclusive_or_expression

: exclusive_or_expression

| inclusive_or_expression '|' exclusive_or_expression

;

3.10 exclusive_or_expression

: and_expression

| exclusive_or_expression '^' and_expression

;

3.11 and_expression

: equality_expression

| and_expression '&' equality_expression

;

3.12 equality_expression

| expression "==" expression

| expression "!=" expression

;

3.13 Type

: CHAR

| INT

| BOOL

| INTSET
| CHARSET
| BITSET
| TUPLE
;

3.14 statement
: compound_statement
| expression_statement
| selection_statement
| iteration_statement
| touch_statement

3.15 compound_statement
: '{' '}'
| '{' block_item_list '}'
;

3.16 block_item_list
: block_item
| block_item_list block_item
;

3.17 block_item
: declaration
| statement
;

3.18 expression_statement
: ';'
| expression ';'
;

3.19 selection_statement
: IF '(' expression ')' statement ELSE statement
| IF '(' expression ')' statement

3.20 iteration_statement
: WHILE '(' expression ')' statement
| FOR '(' expression_statement expression_statement ')' statement
| FOR '(' expression_statement expression_statement expression ')'
statement
| FOR '(' declaration expression_statement ')' statement

| FOR '(' declaration expression_statement expression ')' statement
;

3.21 touch_statement -> TOUCH '(' ID ';' function_id ';' function_id ')' statement

3.22 declaration-list : dec | dec-list

3.23 declaration : type ' ' id-list

3.24 id-list : id | id-list , id

4. Sample Program

```
 bool extract_even_num(int a) {
   if(a % 2 == 0) return true;
   else return false;
}
bool criteria_sort(int a, int b) {
   if(a < b) return true;
   else return false;
}
main {
int a;
int i, temp;
int sum;
char b;
intset as = {15, 17, 23, 1, 3, 100};
intset bs = {};

a = 5;
// 10개를 입력 받고 집합 bs에 넣음.
for(I = 0; I < 10; ++i) {
cin<int> >> temp;
bs += temp; // 넣을 때 마다 원소가 추가됨.
}
intset cs = as + bs; // 집합 as와 집합 bs의 합집합
cout<intset> << cs; // cs 출력;
// cs에서 짝수인 수들을 전부 더하고 출력함.
touch(cs, extract_even_num, criteria_sort) {
// touch 구문 아래 순회 하면서 할 일 작성. 각 원소는 tempobj로 접근 가능함.
```

```
    sum += tempobj;
}
cout<int> << sum;
}
```

5. Remarks

언어 설계의 전반적인 컨셉은 STL에서 사용되는 주요 컨테이너 중 일부(set, tuple)을 라이브러리가 아닌 언어 차원에서 지원해보려는 것이다. 멤버 함수나 템플릿 함수 형태로 지원되는 컨테이너에 대한 연산 중 일부를 언어의 기본 구성 요소로 넣었기 때문에 헤더 파일을 포함할 필요 없이 바로 사용할 수 있다.

set과 tuple의 차이점에 대해 한번 더 강조한다. Set과 tuple은 둘 다 원소들의 모음이라는 점에서는 같지만 개념적으로 set은 원소들의 순서가 없고, tuple은 원소들이 순서지어져 있다. 컴퓨터에서는 당연히 set도 메모리 상에서 엄격히 따지면 순서가 있지만 여기서는 없다고 가정한다. Set 컨테이너에 대해 순회를 하면서 어떤 일을 하려면 set은 순서를 가져야 한다. 이때 set의 부분 집합을 추출하고 순서를 매겨주며 하는 일을 작성할 수 있게 하는 새로운 구문 "touch" 구문을 고안하였다. Touch는 직관적으로 봤을 대 순서 없이 막 늘어서 있는 집합을 touch함으로써 원하는 원소를 뽑아내고 그곳에 질서를 부여한다는 의미에서 이름을 지었다. 위에서 예로든 것과 유사하게 어떤 집합 A = {2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13}의 짝수 원소에 대해 어떤 작업을 하고 싶다고 한다면, touch(A, 추출함수, 정렬함수) { /*하는일*/} 이런 식으로 작성해주면 된다. 추출함수와 정렬함수의 리턴타입은 bool이다. 추출함수는 하나의 원소를 받아서 if문으로 기준에 맞는지 확인하고 나서 맞으면 true 틀리면 false를 리턴하여 A로부터 원하는 부분집합을 추출해낸다. 만약 조건이 짝수라면 새롭게 만들어진 부분집합은 A' = {2,4,6,8,10,12}이 된다. A'도 역시 순서가 없어서 순회 순서를 정할 수 없다. 여기서 이 문제를 해결하기 위해 우리는 정렬을 사용하였다. C++에서 정렬함수를 인자로 넘겨주는 STL의 함수들을 참고하였다. 두 개의 원소를 뽑아 이들을 비교해야 하므로 정렬함수는 두 개의 인자가 필요하다. 여기서 정렬 함수의 바디가 greater than('>')을 사용하고 있다면 정렬된 집합(=튜플)은 A" = (12,10,8,6,4,2)로 바뀐다. 문법적으로도 집합은 중괄호로 원소를 묶는 반면, 튜플은 소괄호로 원소를 묶는다.