Sub^C Manual

Jisuk Byun

March 2020

1 Full Grammar

Comment → `/*` [`0-9A-Za-z _()+-*:;[]<>`]* `*/`

Program → *VarDecl** *GlobStmt*$^+$

VarDecl → *TypeExpr Var* `;`

GlobStmt → *FuncDef*
| *Stmt*

FuncDef → *TypeExpr Var* `(` *ArgDefList* `)` `{` *VarDecl** *Stmt** `}`
ArgDefList → ⟨*Empty-String*⟩
| *ArgDefList* [`,` *TypeExpr Var*]*

Stmt → `;`
Expr `;`
| `if (` *Expr* `)` `{` *Stmt** `}` [`else` `{` *Stmt** `}`]$^?$
| `switch (` *Expr* `)` `{` *CaseStmt*$^+$ *DefaultCase* `}`
| `while (` *Expr* `)` `{` *Stmt** `}`
| `for (`*Expr* `;` *Expr* `;` *Expr*)`{` *Stmt** `}`
| `continue ;`
| `break ;`
| `return` *Expr* `;`
CaseStmt → `case` *Int* `:` *Stmt**
DefaultCase → `default :` *Stmt**

$$
\begin{array}{rl}
\textit{Expr} \rightarrow & \textit{Int} \\
| & \textit{LValue} \\
| & \textit{LValue} \texttt{ = } \textit{Expr} \\
| & \textit{op}_u \ \textit{Expr} \\
| & \textit{Expr} \ \textit{op}_b \ \textit{Expr} \\
| & \textit{Var} \texttt{ (} \textit{ArgList} \texttt{)} \\
| & \textit{SysCall} \\
\textit{ArgList} \rightarrow & \langle \textit{Empty-String} \rangle \\
| & \textit{Expr}[\texttt{,} \ \textit{Expr}]^* \\
 & \\
\textit{LValue} \rightarrow & \textit{Var} \\
| & \texttt{(} * \textit{LValue} \texttt{)} \\
| & \texttt{(} \& \textit{LValue} \texttt{)} \\
| & \textit{LValue} \texttt{ [} \textit{Expr} \texttt{]} \\
| & \textit{LValue} \texttt{ .} \ \textit{Var} \\
 & \\
\textit{SysCall} \rightarrow & \texttt{malloc (} \textit{Expr} \texttt{)} \\
| & \texttt{free (} \textit{Expr} \texttt{)} \\
| & \texttt{read ( )} \\
| & \texttt{write (} \textit{Expr} \texttt{ ,} \ \textit{Expr} \texttt{)} \\
| & \texttt{random ( )} \\
 & \\
\textit{op}_u \rightarrow & \texttt{-} \ | \ \texttt{!} \\
\textit{op}_b \rightarrow & \texttt{+} \ | \ \texttt{-} \ | \ \texttt{*} \ | \ \texttt{/} \\
| & \texttt{\%} \ | \ \texttt{\&\&} \ | \ \texttt{||} \ | \ \texttt{==} \\
| & \texttt{!=} \ | \ \texttt{<} \ | \ \texttt{<=} \ | \ \texttt{>} \\
| & \texttt{>=} \\
 & \\
\textit{TypeExpr} \rightarrow & \texttt{int} \\
| & \textit{TypeExpr} \texttt{ [} \textit{Int} \texttt{]} \\
| & \textit{TypeExpr} \texttt{ *} \\
| & \texttt{struct} \ \textit{Var} \\
| & \texttt{struct \{} \ \textit{VarDecl}^* \ \texttt{\}} \\
| & \texttt{struct} \ \textit{Var} \texttt{ \{} \ \textit{VarDecl}^* \ \texttt{\}} \\
 & \\
\textit{Var} \rightarrow & [\texttt{A-Za-z\_}][\texttt{0-9A-Za-z\_}]^* \text{ (except Keywords)} \\
\textit{Int} \rightarrow & \texttt{0} \ | \ [\texttt{1-9}][\texttt{0-9}]^* \ | \ \texttt{-}[\texttt{1-9}][\texttt{0-9}]^*
\end{array}
$$

2 Sub^C Grammar Details

이 섹션은 Sub^{C}의 문법과 의미를 상향식으로 자세히 설명할 것이다.

2.1 Comment

/* 와 */ 사이에 있는 문자열은 주석이다. /* 와 */ 사이에 대부분의 문자가 들어갈 수 있으며, / 문자는 허용되지 않는다.

```
/* this is SubC comment example */
int x;
```

Listing 1: Comment

2.2 Variable and Integer

*Var*은 영문자로 시작하는 문자열이다. 변수와 식별자를 표시할 때 쓰인다. 첫번째 글자는 영문자와 밑줄만 허용하지만, 두번째 글자부터는 숫자도 포함할 수 있다.

*Int*는 정수값을 표현할 때 사용한다. 십진수 표기법을 따르며, '0' 만 표현하지 않는 이상 맨 앞 글자는 0이 될 수 없다. Sub^{C}의 정수값은 유한한 범위의 부호 있는 정수이다.

```
int _x;
int[100] compilerGrade_;
int zero;

compilerGrade_[10] = 100;
_x = -10;
zero = 0;
```

Listing 2: Variable and integer

2.3 Type Expression

Sub^{C}의 기본 타입은 `int` 하나밖에 없으며, 이외에는 배열, 구조체, 포인터 타입을 쓸 수 있다. `typedef` 키워드는 지원되지 않지만, 대신 `struct [name]` 의 사용자 정의 구조체 이름으로 새 타입이름을 만들 수 있다.

- 배열

  다른 타입 표현 뒤에 대괄호와 숫자를 이용해서 배열 타입을 만들 수 있다. 예를 들어, `int[10 ]`은 크기 10의 정수 배열 타입을 의미한다. C의 배열 타입은 `int v [10 ]`처럼 변수 뒤에 대괄호가 위치하지만, Sub^{C}의 배열 타입은 변수 앞에 대괄호가 위치한다.

- 포인터

  다른 타입 표현 뒤에 별표를 넣어 포인터 타입을 만들 수 있다. 예를 들어, `int*`는 정수 포인터 타입이다. Sub^{C}의 포인터 타입 크기는 항상 1이다.

- 구조체 및 사용자 정의 구조체 타입

  구조체 타입은 C의 구조체 문법과 유사하게 표현 가능하다. `struct`키워드로 시작해서 이름이 있는 구조체 타입, 이름이 없는 구조체 타입 모두 표현 가능하며, 한 번 등장한 이름이 있는 구조체 타입은 같은 이름 범위(scope) 안에서는 `struct [name]` 과 같이 중괄호 부분을 생략해 적을 수 있다.

```
int x;         /* integer */
int[10] xs;    /* length-10 integer array */
int* p;        /* integer-pointer */

/* struct type with no name */
struct {
  int xx;
  int yy;
} point2d;

/* custom type struct-student array. The length of this type is 30.
      */
struct student {
  int age;
  int grade;
  /* recursively defined struct type is allowed only if the size of
     the type is predictable. */
  struct student * friends;
}[10] students;

/* the usage of defined struct type name */
struct student jisuk;
```

Listing 3: Type expression examples

2.4 Integer Operator

Sub^{C}의 값 연산자들은 전부 정수 값을 다루는 연산자들이며, 정수 연산, 논리 연산, 비교 연산 세 종류로 이루어져 있다. 연산자의 의미는 C와 같다. 정수 연산에는 사칙연산과 나머지 연산이 있다. 논리 연산에는 and, or, not 연산자가 있으며 0은 거짓, 0 아닌 값은 참으로 해석된다. 비교 연산에는 less-than, less-or-equal, greater-than, great-or-equal 연산자가 있다. 비트 연산자는 없다.

```
int x; int y; int z;

x = 3; y = 5;

z = x + y;  /* z -> 8 */
z = y % x;  /* z -> 2 */
z = x && y; /* z -> true */
z = ! z;    /* z -> false (0) */
```

Listing 4: Operator examples

2.5 Left Value

문법에서 *LValue*로 표현하는 Left Value는 대입문의 왼쪽에 등장할 수 있는 값이어서 Left Value라고 부른다. 미리 선언되었던 변수들이 대표적이며, 역참조, 참조, 배열 주소 접근, 구조체 이름 접근 방식으로 주소값을 만들 수 있다. 구문 해석의 모호함을 없애기 위해 역참조와 참조 연산자를 사용할 때에는 괄호로 감싸 사용하도록 설계되었다. 주의할 점으로는 포인터 타입의 주소에 배열 주소 접근을 할 때는 암묵적으로 포인터 값을 역참조 한 뒤 배열 주소 접근을 한다는 점이 있다. 또한 일반적으로 Left Value가 대입문의 왼쪽이 아닌 값으로 쓰일 때에는 대부분 메모리의 해당 주소값이 가리키는 값을 내놓지만, 배열 타입의 Left Value는 주소값 자체를 내놓는 점을 주의해야 한다.

```
int[3] a;               /* consider that the address of _a_ is 5 */
int *b;

/* consider that the address of _v1_ is 9  */
struct vec2 {int x; int y;} [3] v1;
struct vec2 * v2;      /* consider that the address of _v2_ is 15 */

b = malloc(3);         /* consider that b points to 1021 */
b[0] = 30;             /* memory[1021] = 30       */
b[1] = 300;            /* memory[1022] = 300      */
b[2] = 3000;           /* memory[1023] = 3000     */

a[0] = b + 10000;      /* memory[5]    = 11021    */
a[1] = (*b) + 20000;   /* memory[6]    = 20030    */
a[2] = b[1] + 40000;   /* memory[7]    = 40300    */

v1[0].x = 4;           /* memory[9]    = 4        */
v1[0].y = 5;           /* memory[10]   = 5        */
v2 = v1;               /* memory[15]   = 9        */
v2[1].x = 10;          /* memory[11]   = 10       */
v2[2].x = 20;          /* memory[13]   = 20       */
```

Listing 5: Left value examples 1

```
int a; int b;
a = 100;  b = 200;

int swap(int * x, int * y) {
  int tmp;
  tmp = (*x);
  (*x) = (*y);
  (*y) = tmp;

  return 0;
}

swap((&a), (&b));
```

Listing 6: Left value examples 2

2.6 Expression

Sub^{C}의 표현식은 계산 이후 값을 가지는 모든 표현을 의미한다. 일반적인 표현식처럼 정수값, 변수값(암묵적인 주소 참조), 연산자가 들어있는 식들을 포함한다. 또한 부작용이 있는 대입문이나 시스템 함수, 일반 함수들 또한 계산 이후 값을 가지므로 표현식으로 여겨진다.

```
int a; int b; int c; int d;
a = 1; b = 2; c = 3; d = 4;

int f (int x, int y) {
  return x + y;
}

a = 3;         /* integer expression           */
b = a;         /* lvalue expression            */
c = 3 - 2;     /* binary operation expression  */
c = (! a);     /* unary operation expression   */
a = d = (3 + a);  /* multiple assigments in a line is allowed */
d = f(a, c);   /* function call expression     */
```

Listing 7: Expression examples

2.7 System Call

Sub^{C}에는 다섯가지 시스템 콜 함수가 있다.

- `malloc(n)` 은 n만큼(메모리 한 칸에는 하나의 정수값이 들어갈 수 있다) 힙 메모리를 할당하고 그 첫 주소를 반환한다.
- `free(n)` 는 주소 n에 할당된 메모리를 회수하고 값 n을 반환한다.
- `read()` 는 standard input 으로 정수값을 읽어 반환한다.
- `write(n, 0)` 은 정수 n을 standard output 으로 출력한다.
- `write(0, 1)` 은 whitespace character 하나를 standard output 으로 출력한다.
- `write(0, 2)` 는 newline character 하나를 standard output 으로 출력한다.
- `random()` 은 0 부터 1023 사이의 정수값 하나를 무작위로 반환한다.

2.8 Statement

Sub^{C}의 명령문은 함수를 제외한 프로그램 실행 순서를 결정하는 표현들을 의미한다. 문법적으로는 모두 ; 이나 } 로 끝난다. `break`는 가장 가깝게 명령문을 감싸고 있는 `switch`, `while`, `for`문을 벗어나고, `continue`는 가장 가깝게 명령문을 감싸고 있는 `while`, `for`문에 다시 진입한다. `switch`문장은 default case 하나와 적어도 하나의 case 문장을 가지고 있어야 한다. 또한 `switch`내부의 문장은 기본적으로 fallthrough 의미를 가진다.

```
int i;
int a; int b; int c; int d;
a = 2; b = 4; c = 7; d = 10;
if(a == b) {
  b = 2;  /* the program does not reach here */
} else {
  for (i = 1; i < 7; i = i + 1)
  {
    b = b + i;
    if ((b % 3) == 0) {
      break; /* the program does not reach here */
    }
  }
}
while(i > 0)
{
  b = b + (a * i);
  if(b < 100) {continue;}
  i = i - 1;
}

/* b will be 151 at this point */
switch (b)
{
  case 151:
    d = b;
    break;
  default:
    d = random();
}
```

Listing 8: Expression examples

2.9 Function Definition

Sub^{C}의 함수 정의는 C와 거의 다르지 않다. 타입 표현이 제한적이어서 void return type을 가질 수 없으며, 명령문 등장 전에 모든 지역 변수들이 선언되어야 한다.

2.10 Program

Sub^{C}프로그램은 세 요소로 구성된다. 전역 변수 선언, 함수 정의, 그리고 명령문이다. 전역 변수는 항상 프로그램의 제일 앞에 모두 선언되어야만 한다. 전역 변수 선언들 뒤에 올 함수 정의와 명령문들은 순서에 상관없이 먼저 쓰인 내용이 먼저 실행된다.

```
int[8] a;
int    i;

for (i = 0; i < 8; i = i + 1) {
  a[i] = random();
  write(a[i], 0);
  write(0   , 1);    /* second_argument 1 whitespace */
}
write(0, 2);         /* second_argument 2 newline    */

```

```
int quicksort (int* v, int low, int high) {
  int pivot; int l; int h; int tmp;

  if (low >= high) { return 0; }

  if ((low + 1) == high) {
    if(v[low] > v[high]) {
      pivot   = v[low];
      v[low]  = v[high];
      v[high] = v[low];
    }
    return 0;
  }

  /* initialize */
  pivot = v[low];
  l     = low + 1;
  h     = high;

  /* loop */
  while (l < h) {
    /* move low high pointers */
    while ((v[l] < pivot) && (l < high)) { l = l + 1; }
    while ((v[h] >= pivot) && (h > low)) { h = h - 1; }
    if (l >= h) { break; }

    tmp  = v[l];
    v[l] = v[h];
    v[h] = tmp;
  }

  /* change pivot and v_h ___ since v_l will be greater than v_h */
  if (h > low) {
    v[low] = v[h];
    v[h]   = pivot;

    quicksort (v, low, (h - 1));
    if (h < high) { quicksort (v, (h + 1), high); }
  }
  else {
    quicksort (v, (low + 1), high);
  }

  return 0;
}

quicksort(a, 0, 7);

for (i = 0; i < 8; i = i + 1) {
  write(a[i], 0);
  write(0   , 1);    /* second_argument 1 whitespace */
}
write(0, 2);         /* second_argument 2 newline    */
```

Listing 9: Quicksort function example

3 Sub^C Testcases for Compiler Assignment

이 섹션은 Sub^{C}언어를 CMachine 에서 사용할 수 있는 기계어로 바꾸는 과제 채점에 쓰일 Sub^{C}프로그램의 모습을 다룬다.

- main 함수가 따로 없다.

  코드는 위에서 아래로 주욱 실행되는 것으로 여겨지며, 다 실행된 뒤에는 오류 없이 멈춰야 한다.

- 정수는 부호 있는 63-bit 정수형이며, overflow와 underflow가 일어나는 프로그램은 없다.

  Sub^{C}프로그램의 정수값은 OCaml 기본 정수형 구현을 따라간다. OCaml 기본 정수형은 63, 31 bit 부호있는 정수이며(시스템에 따라 다르다), 채점 환경은 63 bit 부호 있는 정수형을 사용한다. 채점에 쓰이는 프로그램들은 31bit 정수값 계산에 대해 overflow, underflow가 일어나지 않을 것이다.

- 함수를 값처럼 사용하지 않는다.

  Sub^{C}문법은 함수의 타입을 표시할 수 있는 방법을 제공하지 않으며, 따라서 변수에 함수 포인터를 담을 수도 없다. 시스템 콜로 미리 정의된 함수들과 프로그램에서 정의한 함수들은 *Expr*에서 인자를 넘겨주며 호출하는 방식으로만 사용한다.

- 함수를 정의할 때 항상 `return` 명령문을 통해서 함수 내부 실행이 끝난다.

- 함수 정의할 때 정해진 인자 갯수와 타입을 지켜 함수를 호출한다.

- 배열 타입값을 인자로 받는 함수는 정의하지 않는다. 포인터나 구조체는 인자로 받을 수 있다.

- `if`, `for`, `while`, `switch`문장의 조건문을 계산하면 항상 정수값이 나온다.

- `if`명령문에 대해 `break`나 `continue`명령문을 사용할 수 없다.

- `switch`명령문에 대해 `break`명령문은 사용할 수 있지만 `continue`명령문은 사용할 수 없다.

- `switch`명령문에 들어가는 case들은 그 순서나 연속성이 전혀 보장되지 않는다. case 문장의 조건 정수의 격차는 아주 크지는 않다.

  예를 들어, `switch`명령문 안에 5, 1, (-1), 7 조건의 case 문장들과 default case 문장이 들어있을 수 있다. case 문장의 조건 정수의 격차가 아주 크지 않다는 뜻은 조건 정수의 최대 크기차가 1000을 넘지 않는다는 뜻이다. 예를 들어, `switch`명령문 안에 (-1000), 5, 1000, (-1) 조건의 case 문장들이 들어간 프로그램은 조건 정수의 최대 크기차가 2000 이므로 채점 프로그램으로 쓰이지 않는다.

- 대입문의 왼쪽 자리에 `)&`*LValue*`(`는 사용하지 않는다.