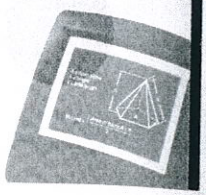
section

A.1 소개

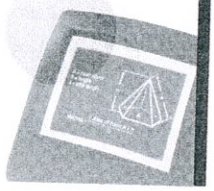


부록 A는 이 책에서 전반적으로 설명된 Mini C 언어에 대한 내용을 담고 있다. 여기서 Mini C를 소개하는 취지는 ANSI C와 같은 실제적인 언어에 대한 컴파일러를 구현하려면 복잡한 점이 많이 발생하므로 보다 간단하고 C 언어와 비슷한 형태를 갖는 언어의 실험용 컴파일러를 구현해 보려는 데 있다고 할 수 있다.

Mini C는 기존의 ANSI C 문법으로부터 몇 가지 언어 구조를 축소하여 고안하였다. 전체적인 프로그램 구조는 유사하지만 자료형은 정수형만 있고 다중 배열과 같은 구조는 존재하지 않는다. 또한, 비트 관련 연산자는 제외하였다. 그러나, 일반적으로 실험용 컴파일러를 설계해 보는 데는 교육적으로 큰 효과가 있다고 생각된다.

section

A.2 어휘 구조



Mini C 언어의 어휘 구조를 요약하면 다음과 같다.

1. 주석문의 종류는 2 가지이며 라인 주석문과 텍스트 주석문이 있다. 라인 주석문은 //부터 라인 끝까지가 주석문이며 텍스트 주석문은 /*와 */ 사이에 나타내며 주석 내에서 다시 주석문을 사용할 수 없다. 그리고 주석은 프로그램의 어느 부분에서나 나타날 수 있다.
2. 명칭의 형태는 아래의 정규표현에 의해 정의되는데 실제로 컴파일러를 구현할 때에는 길이에 제한을 두는 것이 좋다.

   letter = 'a' | 'b' | … … | 'z'
   digit = '0' | '1' | … … | '9'
   <ident> = (letter + _)(letter + digit + _)*

3. 상수는 정수 상수만이 있으며 ANSI C 언어와 마찬가지로 진법에 따라 8진수, 10진수 16진수로 나뉜다. 8진수는 0으로 시작하고 16진수는 0X(또는 0x)로 시작한다.
   ① 526은 10진수(decimal) 526이다.
   ② 0526은 8진수(octal) 526이다.
   ③ 0xFF나 0Xff는 모두 16진수(hexa-decimal)를 나타낸다.
4. 예약어는 명칭으로 사용할 수 없다.
5. 계산을 위한 연산자의 종류는 다음과 같다.
   ① 사칙 연산자 : +, -, *, /, %
   ② 배정 연산자 : =, +=, -=, *=, /=, %=
   ③ 논리 연산자 : !, &&, ||
   ④ 관계 연산자 : ==, !=, <, >, <=, >=
   ⑤ 증감 연산자 : ++, --
6. 지정어는 7개가 있으며 그 종류는 다음과 같다.

   const else if int return void while

section

A.3 구문 구조

Mini C 언어는 ANSI C 언어를 축소하여 만들었으며 ANSI C 언어에 완전한 서브셋이다. 따라서 Mini C로 짠 프로그램은 ANSI C 컴파일러를 이용하여 실행해 볼 수 있다.

Mini C 프로그램의 구조는 크게 선언 부분과 함수 정의 부분으로 나누어진다. 선언 부분은 함수 밖에서 선언되는 외부 선언(external declaration)을 의미하며 함수 정의 부분은 C 프로그램과 마찬가지로 프로그램의 기본 단위(basic unit)이다. 변수의 형은 오직 정수형만 가능하며 배열도 1차원 배열만 허용하고 있다. 매개 변수 전달 방법은 단순 변수(simple variable)일 때는 값-호출이고 복합 변수(complex variable)인 경우에는 주소-호출이다.

1. 상수 선언 : `const int max = 100;`
2. 변수 선언 : `int i,j;`
   `int vector[max];`
3. 함수 정의 :

```
int factorial(int n)
{
    if (n == 1) return 1;
        else return n*factorial(n-1);
}
```

Mini C 언어에서 지원하는 문장은 배정문과 호출문을 포함하고 있는 수식문, 복합문, if 문, while 문, return 문 등이 있다. 다만 for 문장은 지원되지 않는다.

1. Expression st ..... var = exp; factorial(n);
2. Compound st ..... { }
3. If st ..... if (<condition>) <statement>
   if (<condition>) <statement> else <statement>
4. While st ..... while (<condition>) <statement>
5. Return st ..... return <opt_expression>;

section

A.4 예제 프로그램

- prime.mc :

```
const int max = 100;
void main()
{
   int i, j, k;
   int rem, prime;      // rem : remainder

   i = 2;
   while (i <= max) {
      prime = 1;
      k = i / 2;
      j = 2;
      while (j <= k) {
         rem = i % j;
         if (rem == 0) prime = 0;
         ++j;
      }
      if (prime == 1) write(i);
      ++i;
   }
}
```

- bubble.mc :

```
void main()
{
  int list[100];
  int element;
  int total, i, top;
  int temp;
  i = 1;
```

```
    read(element);
    while (element != 0) {          // read a list
        list[i] = element;
        ++i;
        read(element);
    }

    top = total = i - 1;
    while (top > 1) {               // sorting the list
        i = 1;
        while (i < top) {
            if (list[i] > list[i+1]) {
                temp = list[i];
                list[i] = list[i+1];
                list[i+1] = temp;
            }
            ++i;
        }
        top--;
    }

    i = 1;
    while (i <= total) {            // print the sorted list
        write(list[i]);
        ++i;
    }
}
```

- perfect.mc :

```
/*
    A perfect number is an integer which is equal to the sum of all
    its divisors including 1 but excluding the number itself.
    연습문제 2.18 참조
*/

const int max = 500;

void main()
{
```

```
    int i, j, k;
    int rem, sum;        // rem: remainder

    i = 2;
    while (i <= max) {
      sum = 0;
      k = i / 2;
      j = 1;
      while (j <= k) {
          rem = i % j;
          if (rem == 0) sum += j;
          ++j;
      }
      if (i == sum) write(i);
      ++i;
    }
  }
```

- pal.mc :

```
/*
  A palindromic number is unchanged if its digits are reversed.
  121 or 1221 is a palindrome.
*/

void main()
{
  int  org, rev;      // org: original, rev: reverse
  int  i, j;

  read(org);
  if (org < 0) org = (-1) * org;
  i = org;
  rev = 0;
  while  (i != 0)  {
    j = i % 10;
    rev = rev * 10 + j;
    i /= 10;
  }
  if (rev == org) write(org);
}
```

- factorial.mc :

```
/*
   factorial program by recursive call
*/

void main()
{
   int n, f;
   read(n);
   write(n);
   f = factorial(n);
   write(f);
}
int factorial(int n)
{
  if (n == 1) return 1;
    else return n*factorial(n-1);
}
```

section

A.5 문법 형태

다음은 Mini C 문법을 LALR(1) 문법으로 작성한 것이다. 생성 규칙의 개수는 모두 97개이며 ANSI C 언어의 기본 문법을 따랐다. 또한, 이 문법은 AST를 효과적으로 생성할 수 있도록 설계한 형태이다. 10.3절에서는 이 문법에 AST 정보를 결합하고 스탠포드 PGS에 적합한 형태로 기술하였다.

1. <mini_c> → <translation_unit>
2. <translation_unit> → <external_dcl>
3. | <translation_unit> <external_dcl>
4. <external_dcl> → <function_def>
5. | <declaration>
6. <function_def> → <function_header> <compound_st>
7. <function_header> → <dcl_spec> <function_name> <formal_param>
8. <dcl_spec> → <dcl_specifiers>
9. <dcl_specifiers> → <dcl_specifier>
10. | <dcl_specifiers> <dcl_specifier>
11. <dcl_specifier> → <type_qualifier>
12. | <type_specifier>
13. <type_qualifier> → 'const'
14. <type_specifier> → 'int'
15. | 'void'
16. <function_name> → <ident>
17. <formal_param> → '(' <opt_formal_param> ')'
18. <opt_formal_param> → <formal_param_list>
19. | ε
20. <formal_param_list> → <param_dcl>
21. | <formal_param_list> ',' <param_dcl>
22. <param_dcl> → <dcl_spec> <declarator>

23. <compound_st> → '{' <opt_dcl_list> <opt_stat_list> '}'
24. <opt_dcl_list> → <declaration_list>
25. | ε
26. <declaration_list> → <declaration>
27. | <declaration_list> <declaration>
28. <declaration> → <dcl_spec> <init_dcl_list> ';'
29. <init_dcl_list> → <init_declarator>
30. | <init_dcl_list> ',' <init_declarator>
31. <init_declarator> → <declarator>
32. | <declarator> '=' <number>
33. <declarator> → <ident>
34. | <ident> '[' <opt_number> ']'
35. <opt_number> → <number>
36. | ε
37. <opt_stat_list> → <statement_list>
38. | ε
39. <statement_list> → <statement>
40. | <statement_list> <statement>
41. <statement> → <compound_st>
42. | <expression_st>
43. | <if_st>
44. | <while_st>
45. | <return_st>
46. <expression_st> → <opt_expression> ';'
47. <opt_expression> → <expression>
48. | ε
49. <if_st> → 'if' '(' <expression> ')' <statement>
50. | 'if' '(' <expression> ')' <statement> 'else' <statement>
51. <while_st> → 'while' '(' <expression> ')' <statement>
52. <return_st> → 'return' <opt_expression> ';'
53. <expression> → <assignment_exp>
54. <assignment_exp> → <logical_or_exp>
55. | <unary_exp> '=' <assignment_exp>

56. | <unary_exp> '+=' <assignment_exp>
57. | <unary_exp> '-=' <assignment_exp>
58. | <unary_exp> '*=' <assignment_exp>
59. | <unary_exp> '/=' <assignment_exp>
60. | <unary_exp> '%=' <assignment_exp>
61. <logical_or_exp> → <logical_and_exp>
62. | <logical_or_exp> '||' <logical_and_exp>
63. <logical_and_exp> → <equality_exp>
64. | <logical_and_exp> '&&' <equality_exp>
65. <equality_exp> → <relational_exp>
66. | <equality_exp> '==' <relational_exp>
67. | <equality_exp> '!=' <relational_exp>
68. <relational_exp> → <additive_exp>
69. | <relational_exp> '>' <additive_exp>
70. | <relational_exp> '<' <additive_exp>
71. | <relational_exp> '>=' <additive_exp>
72. | <relational_exp> '<=' <additive_exp>
73. <additive_exp> → <multiplicative_exp>
74. | <additive_exp> '+' <multiplicative_exp>
75. | <additive_exp> '-' <multiplicative_exp>
76. <multiplicative_exp> → <unary_exp>
77. | <multiplicative_exp> '*' <unary_exp>
78. | <multiplicative_exp> '/' <unary_exp>
79. | <multiplicative_exp> '%' <unary_exp>
80. <unary_exp> → <postfix_exp>
81. | '-' <unary_exp>
82. | '!' <unary_exp>
83. | '++' <unary_exp>
84. | '--' <unary_exp>
85. <postfix_exp> → <primary_exp>
86. | <postfix_exp> '[' <expression> ']'
87. | <postfix_exp> '(' <opt_actual_param> ')'
88. | <postfix_exp> '++'
89. | <postfix_exp> '--'

90. <opt_actual_param> → <actual_param>
91. | ε
92. <actual_param> → <actual_param_list>
93. <actual_param_list> → <assignment_exp>
94. | <actual_param_list> ',' <assignment_exp>
95. <primary_exp> → <ident'>
96. | <number>
97. | '(' expression ')'