Source Language Reference Manual

Development of a compiler for a new programming language

TEAM-1

Team member 1: Hemant Kumar (CS18B014)

Team member 2: Rohit Shakya (CS18B029)

Team member 3: Sourabh Misal (CS18B022)

Team member 4: K Pavan Kumar (CS18B015)

03.05.2021

6TH sem compiler lab

Program Constructs -

Identifiers -

An identifier is a sequence of letters, digits, and underscores (_). The first character should be a letter. Uppercase and lowercase letters are distinct. Identifier max length is 20.

Data Types -

Only integers are allowed and can be declared as

- **Integer** var1;

  This line declares an integer named var1 but does not define its value and its value should not be assumed to be anything in particular.

- **Integer** var1 = 3;

  This line declares an integer named var1, and initializes it with the integer

value 3.

Arrays -

An array is a data structure that lets you store one or more elements consecutively in memory. array elements are indexed beginning at position zero, not one.

Only 1D integer arrays are allowed.

Declaring Arrays:

You declare an array by specifying the data type for its elements, its name, and the number of elements it can store. Here is an example that declares an array that can store 5 integers. the number of elements in an array must be positive.

- **Integer** array1[5];

Initializing Arrays:

You can initialize the elements in an array when you declare it by listing the initializing values, separated by commas, in a set of braces. Here is an example.

- **Integer** array1[5] = {1, 5, 3, 4, 6};

Accessing Arrays:

You can access the elements of an array by specifying the array name, followed by the element index, enclosed in brackets. Remember that the array elements are numbered starting with zero. examples:

- array1[2] = 8;

  This will assign the value 5 to the 3rd element in the array, which is at index 2nd.

- Var1 = array1[5];

  This will assign the value of the 5th element in the array, in var1 which is of type integer.

Expressions -

An *expression* consists of at least one operand and zero or more operators. Operands are typed objects such as constants, variables, and function calls that return values. Here are some examples:

Arithmetic Expression -

- 8 (constants)
- - 3 (with a unary operand)
- 2 + 3 (with a binary operand)
- nsum(8) (return of a function call)
- (2 * ((3 + 10) - (2 * 6))) (with Parentheses that groups subexpressions)
- 5 + 2 * 4 / 2 (with many operators)

  Innermost expressions are evaluated first, In the above example, 3 + 10 and 2 * 6 evaluate to 13 and 12, respectively. Then 12 is subtracted from 13, resulting in 1. Finally, 1 is multiplied by 2, resulting in 2. The outermost parentheses are completely optional.

  An *operator* specifies an operation to be performed on its operands.

Relational Expression -

- 1 < 2 (true)
- 1 > 2 (false)
- var1 <= 2 (depend on the value of var1)
- 5 >= var2 (depend on the value of var2)
- var1 == var2 (depend on the value of var1 and var2)

Operators -

- **Assignment operator -**

  integer var1 = 10;

  Assignment operators store values in variables.

- **Arithmetic operator -**

  Var1 = 2 + 3 (Addition)

  Var1 = 2 - 3 (Subtraction)

  Var1 = 2 * 3 (Multiplication)

  Var1 = 2 / 3 (Division)

  Var1 = - 3 (Negation)

- **Relational operator -**

  We use the relational operators to determine how two operands relate to each other (comparisons) : are they equal to each other, is one larger than the other, is one smaller than the other, and so on. When you use any of the comparison operators, the result is either 1 or 0, meaning true or false, respectively.

  The equal-to operator (==) tests its two operands for equality. The result is 1 if the operands are equal, and 0 if the operands are not equal.

  if(x==y){ output(“x and y are equal”); }

  Beyond equality and inequality, there are operators you can use to test if one value is less than, greater than, less-than-or-equal-to, or greater-than-or-equal-to another value. Here are some code samples that shows usage of these operators:

  if(x<y){ output(“x is less than y”); }

  if(x>y){ output(“x is greater than y”); }

```
if(x<=y){ output(“x is either less or equal to y”); }

if(x>=y){ output(“x is either greater or equal to y”); }
```

Comments -

- Single line comments - // any text
- Multi line comments - /* any text with new lines */

Statements -

- **If-Else statement -**

  You can use the if statement to conditionally execute part of your program, based on the truth value of a given expression. Here is the general form of an if statement:

  ```
  if ( condition ){ statement }

  else{ else-statement }
  ```

  Example -

  ```
  if(x>y){ output(“x is greater than y”); }

  else{ output(“x is either less or equal to y”); }
  ```

- **While statement -**

  The while statement is a loop statement with an exit condition at the beginning of the loop. Here is the general form of the while statement:

  ```
  while( condition ){ statement }
  ```

  The while statement first evaluates the condition. If condition evaluates to true, then statement is executed, and then condition is evaluated again. statement continues to execute repeatedly as long as condition is true after

each execution of statement.

Example -

```
Integer i = 0;

while( i < 10 ){ i = i + 1 }

output( i );
```

- **For statement -**

  The for statement is a loop statement whose structure allows easy variable initialization, expression testing, and variable modification. It is very convenient for making counter-controlled loops. Here is the general form of the for statement:

  ```
  for ( initialize; condition; incr/decr ){ statement }
  ```

  Example -

  ```
  Integer i, s;

  for ( i =0; i < 10; i = i + 1 ){ s = s + i; }

  output( s );
  ```

- **Assignment statement -**

  We use Assignment operator to store values in variables

  ```
  id = arithmetic expression

  id = call(id,param1,param2)

  array [ index ] = expr

  array [ index ] =call(id,param1,param2)
  ```

- **I/O statement -**

  We use I/O statements to print the values of variable or for taking the input from console to store:

  Input(id); | **input**(id[index]);

Output(id); | **output**(id[index]); | **output**(“string”);

Functions -

Every program requires at least one function, called main. That is where the program’s execution begins.

A function is a group of statements that together perform a task. Every program has at least one function, which is main(), and in programs we can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

- **Functions can be declared as**

  return_type function_name (param1, param2, ...){

  statements…

  }

- **Call statement for functions**

  call(“function_name”, param1, param2, ...);

- **Return statement**

  send expr;

- **Recursion**

```
return_type   func (param1, param2,...){
      statements…
      call(func, param1, param2,...);
}
```

Block Structured -

A block is a set of zero or more statements enclosed in braces. Blocks are also known as compound statements. Often, a block is used as the body of an if statement or a loop statement, to group statements together.

```
for ( i =0; i < 10; i = i + 1 )
{
s = s + i;
}
```

You can also put blocks inside other blocks:

```
for ( i =0; i < 10; i = i + 1 ){
      if ( i < 5 ){
            output("i is less than 5");
      }
      else{
            output("i is either greater or equal to 5");
      }
}
```

Sample Programs -

Some beginner level programs that can help in understanding syntax of our language

1. **Factorial - recursive**

```
void main(){
    integer n;
    output("Enter Val :");

    input( n );
    integer factn;
    factn = call(factorial,n);

    output("Val of Fact(");
    output(n);
    output(") : ");
    output(factn);
    output("\n");
}
```

```
integer factorial(integer  n){

    integer one;
        one = 1;

        if(n <= 0){
                send one;
        }

        integer factn;
        integer new_n;
        new_n = n - 1;
        factn = call(factorial,new_n);
        factn = n*factn;
        send factn;
}
```

2. **Bubble Sort - iterative**

```
void main(){
   integer array[8], n, c, d, swap;
   n = 8;

   for (c = 0; c < n; c = c + 1){
      input( array[c] );
   }

   for (c = 0 ; c < n - 1; c = c + 1){

      for (d = 0 ; d < n - c - 1; d = d + 1){

         if (array[d] > array[d+1]){
            swap       = array[d];
            array[d]   = array[d+1];
            array[d+1] = swap;
         }
      }
   }

   for (c = 0; c < n; c = c + 1){
      output( array[c] );
      output("   ");
   }
}
```

3. **Selection Sort - iterative**

```
void main(){

   integer a[5],temp,index,i,j,n;
   n=5;

   for (i = 0; i < n; i = i + 1){
      input( a[i] );
   }

   for(i=0; i<n-1; i=i+1){
      index=i;
      for(j=i+1; j<n; j=j+1){
         if(a[index]>a[j]){
            index=j;
         }
      }
      temp=a[i];
      a[i]=a[index];
      a[index]=temp;
   }

   for (i = 0; i < n; i = i + 1){
      output( a[i] );
      output("   ");
   }
}
```

4. **Sum of N number - iterative**

```
void main() {

    integer n,i,total;
    total=0;

    output("enter n\n");
    input(n);

    //for loop start
    for(i=n; i>=1; i=i-1){
        total=total+i;
    }
    /*
    for loop end
    */

    output("sum:");
    output(total);
}
```

5. **Fibonacci - recursive**

```
void main(){

  integer n;

  output("Enter n:");
  input( n );

  integer fibn;
  fibn = call(fib,n);

  output("result: ");
  output(fibn);

}
```

```
integer fib(integer  n){

        if(n <= 1){
           send n;
        }

        integer a,b,sum,t1,t2;
        t1=n-1;
        t2=n-2;

        a = call(fib,t1);
        b = call(fib,t2);

        sum=a+b;
        send sum;

}
```

6. **Lower Triangle - iterative**

```
void main(){

   integer n,m,i,p;

   output("enter n\n");
   input( n );

   m=n;

   for(i=1;i<=n;i = i + 1){

      for(p=1;p<=m;p =p + 1){

         output("*");

      }

      m = m - 1;
      output("\n");

   }

}
```