CS335A
Group 8
Compiler Design

Akshat Sharma, Arpit Singh, Ayush Kumar, Subhrojyoti Chatterjee
190090, 190177, 190213, 190866

January 24, 2022

---

1 Language Specification

- Identifiers : Identifiers are the sequence of characters used for naming variables, functions. Identifier can consists of letters, decimal digits, and underscore character, with no spaces and the first character of identifier cannot be a digit.
- Keywords : Keywords are identifiers reserved for use as part of the programming language itself. You cannot use them for any other purpose like naming varaibles and functions.

  Here is a list of keywords recognized by our language:

  auto, char, int, float, bool, struct, const, for, while, if, else, return, void.
- Constants
  - integer constant
  - char constant
  - float constant
- Data types
  - int: This is a 32-bit integer data type which can hold integer values in the range of -2,147,483,648 to 2,147,483,647.
  - bool: This data type uses 8 bits to store boolean value (1 or 0)
  - char: The is a 8-bit unsigned char data type can hold integer values in the range of 0 to 255.
  - Float: This data type uses 32 bits to store signed floating point number in range -1e37 to 1e37.
- Separators : Separator are single character tokens required to separate tokens. Some examples include ( ) [ ] ; , . :

- White Space : Whitespaces are ignored outside of strings and character constants. White space is the collective term used for several characters: the space character, the tab character, the newline character, the vertical tab character, and the form-feed character.
- Expressions An expression contains at least one operand and zero or more operator.

  Example: $100, 5 + 6, max(10, 4)$.
- Operators
  - Assignment Operators : Used to store values in variables. e.g. int y = 5; where '=' is an assignment operator.
  - Incrementing and Decrementing : ++ operator is used to increment the value of a variable by 1 whereas - - operator is used to decrement the value of a variable by 1. In our implementation both prefix and postfix incrementer and decrementer are valid. e.g int x = y++; and int x = ++y; are both valid statements.
  - Arithmetic Operator : These operators help in carrying out standard arithmetic operations. e.g. addition, subtraction, multiplication, and division, along with modular division and negation operators.
  - Comparison Operators : These operators are used to determine how two operands relate to each other. e.g ==,<=,>=,<,> .
  - Logical Operators : These operators test the truth value of a pair of operands. In our implementation, any expression that evaluates to non-zero value is considered true. e.g. if(x == 2)
  - Bit Shifting : These operators are used to shift the operand's bits to the left or the right. e.g « , »
  - Bitwise Logical Operators : Bitwise Operators are used for manipulating data at the bit level. They are bitwise and , bitwise or, bitwise Xor and binary one's complement.
  - Pointer Operators : The pointer in C language is a variable which stores the address of another variable. Eg int *a; a = &b;
  - Comma Operators : Comma acts as a separator when used with function calls and definitions, function like macros, variable declarations, enum declarations, and similar constructs. Eg int a, b; Here a and b are defined as integers.
  - sizeof Operator : Sizeof is a compile time unary operator which can be used to compute the size of its operand. Eg sizeof(int);
  - Type Casts Operator : A type cast is a special operator that forces one data type to be converted into another. Eg float b; int c; c = (int)b; Here, b is converted to integer and assigned to c.

- Statement - Statement supported by this language are:
  - Expression Statements We can turn any expression into an expression statement by just adding a semicolon at the end of expression.
  - if Statement We use if statement to conditional branching ,based on truth value of the condition expression. General syntax of if Statement is like:

```
1        if( condition ) // If condition is true then the then - statement will execute
2        {
3            then_statement
4        }
5        else // If above condition was false then the else - statement will execute
6        {
7            else_statement
8        }
```

Note: Both the "then_statement" as well as "else_statement" must be enclosed in a scope. And this language does not support *else if* statement.

  - while Statements While statement is a loop statement with a condition. General syntax of while Statement is like:

```
1        while( condition ) // If condition is true then the loop_statement will be execute
2        {
3            loop_statement
4        }
```

Note : The "loop_statement" must be enclosed in a scope.

  - for Statements

```
1        for( initialize ; condition; step )
2        {
3            loop_statement
4        }
```

The for statement first evaluates the expression (initialize). Then it evaluates the expression (condition). If (condition) is false, then the loop ends and program control resumes

after statement. Otherwise, if (condition) is true, then the loop statement is executed. At the end of an iteration (step) is executed, and the next iteration of the loop begins with evaluating (condition) again.
Note : The "loop_statement" must be enclosed in a scope.

- Blocks A block is a set of zero or more statements enclosed within curly braces. This is used to group statements together, and is used as the body of if, else, for and while statement. Variables defined inside a block are local to that block.
- return Statement : We can use a return statement to end the execution of a function and return the flow of program to the function which called it. General form of the return statement:

```
return return_value;
```

- Scope A scope is the part of the program that accesses a declared object. A global variable is accessible throughout the program file, a variable declared within the scope of function or a block is accessible only within that scope after it's declaration.
- Arrays - Fixed size arrays are supported with size specified during declaration of the array.Dynamic sized arrays are not allowed in our implementation of the compiler.
  int arr[4], char a[10] are examples of valid declarations of arrays .
  Arrays can be initialized later but inline initialization is not supported and value at a certain index can be retrieved as in regular C syntax, e.g. int x = arr[2]; , arr[3] = 1;
- Functions - Our compiler supports recursive functions. Only function declaration followed by separate function definition is invalid according to our syntax. Function declaration and definition should be done together.
  Return type allowed in a function is void , int , float , char , bool.

```
int id(int x); // invalid function declaration and definition
int fib(int i){ // valid function declaration and definition
     if(i<=1)
     {
         return i;
     }
     else
     {
         return fib(i-1) + fib(i-2);
     }
}
int id(int x){
```

```
13          return x;
14          }
```

fib is a valid function declaration and definition.Whereas, id is not a valid declaration for our compiler.
Functions can be called just like in regular C syntax, e.g. int x = fib(3); .

- Structs - Structs can be declared,defined and used as in regular C language.We also provide private and public access specifier options. Declaration of default constructor, copy constructor and parameterised constructor is also allowed. An example of a valid implementation is as follows :

```
1         struct person{
2               char name[50];
3               int Aadhar_no;
4         };
5         int main(){
6               struct person p1;
7               p1.name[0] = 'R';
8               p1.name[1] = 'a';
9               p1.name[2] = 'j';
10              p1.Aadhar_no = 12345678;
11              return 0;
12        }
```

- Pointers - Syntax to use pointers is same as that in C language. Pointer arithmetic is allowed and can be used to traverse an array. Example usage of pointer is as follows:

```
1         int main(){
2               int x = 5;
3               int *p = &x;
4               int arr[5];
5               int *x = arr;
6               int *y = x + 1;
7               return 0;
8         }
```

This is a valid use of pointers in our language.

- Shorthand Operators - Shorthand operators is provided for all kind of operators.

a += 2; is equivalent to a = a+2;
Other such operators for which shorthand operators are possible are *,-,/,modulo,bitwise and,bitwise or,bitwise xor,left shift,right shift etc.

- Global Variables - We can declare variables outside the main function. However in our compiler structs cannot be declared as global variables. For example,

```
1       int var;
2       struct y{
3       int x;
4       int z;
5       }
6       struct y myVar;
7       int main(){
8       return 0;
9       }
```

In the above example, var is a valid global variable for our compiler but myVar is an invalid global variable.

---

2 Features

Basic Features -

- Native Data types (integer, boolean, character,float) - These four data types would be supported by our compiler with the addition of strings as a complex data type.
- Variables and Expressions - Variable may be created using aforementioned rules for creating a variable. An expression is a combination of operators and operands that is used to produce some other value.
- Control structures
  - Conditionals (if, if-then-else) - Conditional Statements in may be used to make decisions based on the conditions. The execution flow of the program may change based on the result evaluated by the condition.
  - Loops (for, while) - A loop statement allows us to execute a statement or group of statements multiple times.
- Input/Output statements - printf and scanf are the functions for input and output respectively.

- Arrays - An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together.
- Functions (Recursion supported) - A function is a group of statements that together perform a task.
- User defined types (class/struct) - A structure creates a data type that can be used to group items of possibly different types into a single type.
- Pointers - The pointer in C language is a variable which stores the address of another variable.
- Shorthand Operators - Shorthand Assignment Operators are Binary Operators which require 2 values or 1 variable and another value/expression.

Advanced Features -

- Library functions string - Some library functions defined within string.h would be supported, namely strcpy(), strcmp(), strlen(), etc.
- Global variables - Declaration of variables to be used in all the functions would be supported. Only declaration of global variable is possible not initialization globally. For example,

```
1       int _global_variable_;
2       int main(){
3            _global_variable_ = 10;
4            return 0;
5       }
```

- Dynamic memory allocation - Allocating user specified memory would be supported. Memory would be allocated using functions like malloc(), calloc() and freed using free(). Amount of memory to be allocated should not be another variable (even if it's value is already declared). For example,

```
1       int main(){
2            int* p = (int *)malloc(10 * sizeof(int));
3            free(p);
4            return 0;
5       }
```

- Multi level pointers - Pointers to pointers are possible. For example,

```
int main(){
    int s=2,*r=&s,**q=&r,***p=&q;
    printf("%d",p[0][0][0]);
    return 0;
}
```

- Function overloading - Overloading functions on the basis of different number of arguments is supported.

```
int add(int a, int b) {
    int ans;
    ans = a+b;
    return ans;
}
int add(int a, int b, int c) {
    int ans;
    ans = a+b+c;
    return ans;
}

int main()
{
    int a = 3, b = 4, c = 5;
    int d, e;
    d = add(a, b);
    e = add(a, b, c);
    return 0;
}
```

- Auto-type inference - Using keyword "auto" would be supported for primitive supported types. For example,

```
int main(){
    int i = 5; //variable i will be of type int.
    auto k = i; //variable k will be of type int.
    auto d=1.4445; // variable d will be of type float.
    return 0;
}
```

- Multi Dimensional Array (if time permits): We can make multidimensional arrays, i.e. arrays of arrays. By adding brackets and array lengths for every dimension in Multi Dimensional array.

```
1         int main(){
2             int a[10][10] , b[10][10][10];
3             return 0;
4         }
```

- Code optimizations (If time permits) : Dead Code Optimizations and short circuit evaluation would be supported. Further optimizations would be added if time permits.