Project Proposal
CS335- Compiler Design

Spring-2022
Prof. Amey Karkare
Prof. Subhajit Roy

Group- 6

Members:

190224, Meena Banavathu
190646, Prince Kumar Ahirwar
190824 Shreyasi Prasad
180623 Rohit Kulhari

Repository: https://github.com/Shreyasi3521/Compiler_CS335

Source:	**C++**
Implementation :	**Python3**
Target:	**MIPS**

We're using python3 for implementation as it is a programming language that prioritizes readability, making it easier to understand and use. Also, most of the team members are familiar with it.
Target is MIPS as we've studied it in great detail in previous courses.

Features Planning to implement:

1. Native Data Types

1.1 Integer

- "Int" is the keyword used for numerical values. Max. size of int is 4 bytes(32 bits).
- Some examples using int:
  int var1;
  int var2 = 320043;
- var1 is 'declared' only or 'uninitialised' while var2 is initialized properly.

1.2 Boolean

- "bool" is the keyword used for storing a logical value or a binary value
- true = 1 and false = 0
- *bool* is335cool = false ;
- Is335cool is set to 0 here.

1.3 Character

- "Char" is the keyword used for storing single characters.
- There are 128 characters defined in the C++ ASCII list.
- *char* foo = 'C' ;
  char foo1 = 65 ;
- foo has the character 'C' stored(without the apostrophe) while foo1 stores a character corresponding to 65 ASCII value ('A' here).

2. **Variables and Expressions**

2.1 **Variables**

- Variables are the basic unit of storage used as names given to certain memory locations.
- Declaring variables:
  var_type var_name;
  var_type var_name1, var_name2;
- Multiple variables of the same data type can be declared in a single statement.
- Here *var_type* is the data type we saw above. Var_name, var_name1, var_name2 etc are names of variables.
- With certain exceptions listed below, a variable name can be any combination of characters.
  - It can't start with a number.
  - They're case-sensitive.
  - There're certain reserved keywords in C++ which can't be used.
  - No whitespaces can be used in between.

2.2 **Expressions**

- An expression consists of operands and operators.
- Operands can be constants, variables, and functions.
- Operators are operations to be performed on the operands.
- Types of Operators:
  - Assignment Operators : =, +=, -=, *=, /=, %=, &=
  - Arithmetic Operators: +, -, *, /, &, %, ^, <<, >>
  - Logical Operators: &&, ||, !
  - Comparisons Operators ==, ! =, <, >, <=, >=
  - Increment-Decrements Operators ++, −−
  - Member access Operators a[b], a.b, a− > b, *a, &a

3. Input/Output statements

3.1 **Input**

- Standard input in C++ by default is the keyboard.
- For input operations, cin is used together with the ">>" operator. This operator is then followed by the variable where the extracted data is stored.
  Example:
  int roll_num;
  cin>>roll_num;

3.2 **Output**

- The standard output by default is the screen.

- Stream object defined to access it is cout used with “<<” operator. This operator is followed by the variable or string to be displayed. “/n” will be used to end the line.
  Example:
  ```
  cout<< ”Hello World\n”<<  104 << x;
  ```
  This will print
  ```
  Hello World
  10420            //x = 20
  ```

4. **Conditionals: “if”, “if-else”**

4.1 ***if*** **Statement**

“*if*” Statement can be used to conditionally execute part of the program, depending upon the Truth value of the conditional expression.

- The general form of an *if* Statement:

```
if (test)
    then-statement
```

If the *test* evaluates to true, only then *then-statement* is executed.

4.2 ***if-else*** **Statement**

“*if-else*” is similar to “*if*” but it starts executing *else-statement* if the *test* evaluates to false.

- Generic form of an *if-else* Statement:

```
if (test)
    then-statement
else
    else-statement
```

5. Loops: “for”, “while”

5.1 *for* Loop

“*for*” loop statement has a structure which allows easy variable initialization, expression testing, and variable modification.

- Generic form of a *for* loop:

```
for ( initialization ; test ; modification step )
        loop-statements
```

The *for* statement first goes through *initialization*. Then it evaluates the expression *test*. If *test* is false, then the loop will end. Otherwise, if *test* is true, then *loop-statements* are executed. Finally, *modification step* is evaluated, and the next iteration of the loop begins with evaluating *test* again.

5.2 ***while*** **Loop**

"*while*" loop statement has structure with an exit *test* at the beginning of the loop.

- Generic form of a *while* loop:

```
while (test)
        loop-statements
```

The while statement first evaluates *test.* If *test* evaluates to true, then *loop-statements* are executed. After that *test* is evaluated again. *loop-statements* will execute repeatedly till *test* is true.

6. Arrays

An array is a data structure that stores one or more elements consecutively in memory.

6.1 Array Declaration

An array is declared by specifying the data type of its elements, the name of the array, and the number of elements it can store.

Example:

```
int my_arr[20];
```

Here an integer array is created which can contain 20 integer values.

6.2 Array Initialization

Elements in the array can be initialized while declaring it by listing the values, separated by commas, in a set of braces.

Example:

```
int my_arr[4] = { 1, 2, 3, 4} ;
```

Here it isn't necessary to initialize all value of array here. We can initialize them later by accessing it's elements.

6.3 Accessing elements of an array

Array elements are accessed by specifying the array name, followed by the index of element enclosed in brackets.

Example:

```
my_arr[2] =9;
my_arr[0]=12;
cout<< "second element of my_arr" << arr [1] <<endl;
```

Here arrays have zero-based numbering where initial element of the array assigned index 0.

7. Function and recursion

7.1 Function

Functions are written to separate parts of program into distinct subparts which can be used in program by calling the function.

7.1.1 Function Declaration

Functions are declared by specifying the name of a function, a list of parameters, and the function's return type.

- Generic form of a function declaration:

```
return-type function_name (parameter_ist);
```

Here function name should be a valid identifier.

7.1.2 Function Definition

Function definition consists of information regarding the function's name, return type, and all the parameters, along with the body of the. Function body can have series of statements.

- Generic for of definition:

```
return-type
function_name (parameter_list)
{
        function-body
}
```

Example:

```
int add (int n1, int n2) {   // function to calculate sum of 2 numbers.
        int sum= n1+n2;
        return sum ;
}
```

Function parameters can be any expression, a literal value, a value stored in variable, an address in memory, or a more complex expression built by combining these.

7.1.3 Function Calling

Functions are called by using its name and passing any needed parameters.

- Generic form of function call:

```
function_name ( parameters )
```

Example:

```
add( 9, 23 ); // calling above function add.
```

7.2 Recursion

Recursive functions are almost similar to normal functions as above. In recursion the function calls itself recursively multiple times.

Example:

```
int fib ( int n ) {         // A simple recursive function to find fibonacci number
        if (n == 0)
                return 0;
        if (n ==1 || n ==2)
                return 1;
        else
                return  fib( n-1 ) + fib( n-2 ) ;
```

8. Pointers

- Pointers represent addresses in C++. They help to create and manipulate dynamic data structures.
- Defining:
  var_type *var_name;

8.1 **Multi Level Pointers**

- These are pointers pointing to pointers that can further point to some pointers.

8.2 **Function Pointers**

- A function pointer points to the start of code of a function and not data.
- For instance, we define a function

  ```
  void fun(int a) { … }
  ```

  Then,

  ```
  void (*fun_ptr)(int) = &fun;
  ```

  fun_ptr is function pointer and we can call function fun as

  ```
  (*fun_ptr)(20);
  ```

9. User-defined Data Types

9.1 **Structure**

- A *struct* is a user-defined data type in C++ that is used to group various variables as one.
- Structure is defined as:

  ```
  struct struct_name{
          var_type var_name1;
          var_type var_name2;
          …
          var_type var_nameN;
  };
  ```
- Example:

  ```
  struct student{                //defining
  ```

```
        int roll;
        int age;
        int marks;
};
struct student s1;              //initializing
st.roll = 1;                    //accessing members of struct
```

9.2 **Class**

- It holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- Template to define class:
```
class ClassName{
        Member1;
        Member2;
        ...
        MemberN;
};
```
- Initializing:
```
ClassName ObjectName;
```