

Group - 9

Members:

190353	Hardik Sharma
190428	Ketan Chaturvedi
190652	Priyanshu Yadav
190758	Sanjay Pander

Source:C	Target: MIPS 64-bit
Implementation:Python	

Github link: <https://github.com/ketanch/Compiler>

Structure of Repo:

```
.git/  
src/  
  |-----> main.py (Currently empty)  
docs/  
  |-----> Specification.pdf  
tests/  
  |-----> .gitkeep (to be removed in future)  
Makefile  
README.md  
.gitignore
```

git show specs:

commit 7333ea6f518ab79c56710f89eced3ae1ebe1db41 (tag: specs)
Author: ketanch <56212393+ketanch@users.noreply.github.com>
Date: Fri Jan 21 20:01:49 2022 +0530

Initial commit

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..b6e4761
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,129 @@
+# Byte-compiled / optimized / DLL files
+__pycache__/
+*.py[cod]
+*\$py.class
+
+# C extensions
+*.so
+
+# Distribution / packaging
+.Python
+build/
+develop-eggs/
+dist/
+downloads/
+eggs/
+.eggs/
+lib/
+lib64/
+parts/
+sdist/
+var/

Source language features taken:

Basic

- Native Data types (integer, boolean, character).
- Variables and Expressions
- Control structures
 - Conditionals (if, if-then-else)
 - Loops (for, while).
- Input/Output statements
- Arrays
- Functions (Recursion should be supported)
- User defined types (class/struct)
- Pointers

Advanced features

- Multi level Pointers(upto level 2)
- File I/O
- Simple library functions, such as
 - math functions (sqrt,exp,floor,ceil,abs,pow)

Note: The compiler does **not** support any type-casting either implicit or explicit.

1. Lexical Elements

1.1. Identifiers

- Identifier is a string of characters used for naming variables, functions, structures, preprocessor macros, arrays, unions, enumerations, registers.
 - identifier contains letters [a-zA-Z], digits [0-9] and underscore '_'.
 - First character of an identifier cannot be a digit.
 - Identifiers are case sensitive.
 - Eg `_foo_`, `mynumber200`, `a1b2c3`.

1.2 Keywords

- Keywords are special identifiers reserved for use as part of our language itself.
- We can't use these keywords for naming any variable.
- List of keywords recognized by our compiler.
 - `int`, `float`, `double`, `long`, `char`, `short`, `signed`, `unsigned`, `bool`, `const`, `volatile`, `printf`, `scanf`, `fprintf`, `fscanf`, `true`, `false`
 - `if`, `else`, `break`, `continue`, `switch`, `default`, `case`
 - `for`, `while`, `do`
 - `void`, `return`
 - `struct`, `union`
 - `sqrt`, `exp`, `floor`, `ceil`, `abs`, `log`, `pow`
 - `read`, `write`

1.3 Constants

- Integer Constants
An integer constant is a sequence of digits. eg . `459`, `23901`
- Character Constants
A character constant is usually a single character enclosed within single quotation marks, such as `'Q'`. A character constant is of type "int" by default.
 - `\\` Backslash character.
 - `\?` Question mark character.
 - `\'` Single quotation mark.
 - `\"` Double quotation mark.
 - `\n` Newline character.
 - `\t` Horizontal tab.
 - `//` single line comment
 - `/*` starting of multi-line comment
 - `*/` ending of multi-line comment
 - `\0` end of file

- **Real Number Constants**
A real number constant is a value that represents a fractional number. It consists of a sequence of digits which represents the integer part of the number, a decimal point, and a sequence of digits which represents the fractional part.
Eg . 4.7, 0.00034
- **String Constants**
A string constant is a sequence of zero or more characters, digits, and escape sequences enclosed within double quotation marks. Basically String is an array of characters.
Eg. "This is my string"

1.4 Separators

A Separator separates the tokens
() [] { } ; , . : white space

2. Data Types

- 2.1. Integer Types: We would be implementing all integer data types. The source language(C) would be supporting following:
 - 2.1.1. signed char : It would be 8 bit and hold values ranging from -128 to 127.
Examples:
char x; x='a';
char y='a';
 - 2.1.2. short int : It's size would be 16 bit and support values in range of -32,768 to 32,767
Examples:
short int x; x=32;
short int y=-20;
 - 2.1.3. unsigned short int : It's size would be 16 bit and support values in range of 0 to 65,535
Examples:
unsigned short int x; x=32;
unsigned short int y=20;
 - 2.1.4. int : It's size would be 32 bit and support values in range of -2,147,483,648 to 2,147,483,647
Examples:
int x; x=32;

int y=-20;

- 2.1.5. unsigned int : It's size would be 32 bit and support values in range of 0 to 4,294,967,295

Examples:

unsigned int x; x=65538;

unsigned int y=20;

- 2.1.6. long int : It's size would be 32 bit and support values in range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Examples:

long int x; x=3842;

long int y=-205;

- 2.1.7. unsigned long int : It's size would be 32 bit and support values in range of 0 to 18,446,744,073,709,551,615

Examples:

Unsigned long int x; x=65538;

unsigned long int y=0;

- 2.1.8. long long int : It's size would be 64 bit and support values in range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Examples:

long long int x; x=3842;

long long int y=-205;

- 2.1.9. unsigned long long int : It's size would be 32 bit and support values in range of 0 to 18,446,744,073,709,551,615

Examples:

unsigned long long int x; x=65538;

unsigned long long int y=0;

- 2.1.10. Bool : Our compiler supports boolean data type originally not present in C. It can take value false or true. Only bitwise operations are supported on variables with data type bool.

Examples:

bool x; x=true;

bool y=false;

2.2. Real Number Types:

- 2.2.1. float : It's size would be 32 bit

Examples:

float x; x=65.538;

```
float y=0.0;
float z=-4.35;
```

2.2.2. double : It's size would be 64 bit

Examples:

```
double x; x=6.38;
double y=2.0;
float z=-567.95;
```

2.3. Structs : It is a user defined data type. It's size would be the sum of all of its members. Our compiler does **not** support Bit Fields.

Examples:

```
struct tmp{
    int a;
    float b;
}
```

2.4. Arrays : Array indexing starts at 0. Multi dimensional array support is provided by the compiler. Arrays can have any data type including user defined data types.

Examples:

```
int arr1[10];
int arr2[5]={0,1,2,3,4};
char arr[3]={'a','b','c'};
```

2.5. Pointers : Pointers hold memory addresses of stored constants or variables. Multi level pointers i.e. pointers to pointers are supported but only upto **two** levels.

Examples:

```
int i; int j; int *p1=&i; p1=&j;
int **p2; p2=&p1; //Pointer to pointer
```

2.6. Type qualifiers

2.6.1. const : It makes the variable read only

Examples:

```
const int j=10;
```

2.6.2. volatile : It informs the compiler that no optimization should be performed on code involving this variable and that variable is explicitly changeable.

```
volatile float interest=7.5;
```

3. Expressions and Operators

3.1 Operators

- 3.1.1 Assignment Operators : =, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=
int a=20 // a variable named 'a' is declared of type int
a+=10 // a gets assigned the value after addition of 10, a = 20
- 3.1.2 Unary Operators : ++, --, &, sizeof()
int a=20
a++ //a gets increased by 1(post)
++a //a gets increased by 1(pre)
- 3.1.3 Arithmetic Operators : +, -, *, /, %, -, ~
int a = 4+5*3/1-2 //evaluated using standard precedence rules, a=17
- 3.1.4 Comparison Operators : ==, !=, <, >, <=, >=
if(a==b), if(a<=b)
- 3.1.5 Logical Operators : &&, ||, !
if(a==b || a<b && c>=b)
- 3.1.6 Bit Shift Operators: <<, >>
int a = 20<<1 // a gets assigned value after left shift of 1, a = 40
- 3.1.7 Bitwise Logical Operators : &, |, ^, ~
int a = 2&1 // 2&1 evaluated as 0.
int a = 2|1 // 2|1 evaluated as 3.
int a = 2^1 // 2^1 evaluated as 3.
int a = ~2 // ~2 is one's complement i.e. ~2 = -3
- 3.1.8 Pointer Operators : *, **, &
int* ptr = &x // * is used for pointer, ** is used for pointer to pointer
// & is used to dereference a variable to get its address
// in memory
int **ptr2=&ptr
- 3.1.9 Comma Operator : ,
x++, y = x; // will be treated as 2 different statements by the compiler
// x++ //1st statement
// y=x //2nd statement

3.1.10 Member Access Expressions

You can also access the members of a structure or union variable via a pointer by using the indirect member access operator ->.

x->y is equivalent to (*x).y.

3.2 Expressions - At Least One operand and zero or more operators

3.2.1 Conditional Expressions : a = (x == 5) ? y : z;

3.2.2 Function Expressions : Some of the library functions which will be used in

programs are defined below.

printf("An int variable is printed here %d",a) //a is an integer variable

scanf("An int variable is scanned here %d",&a) //a is an integer variable

fprintf(fp,"Integer printed in opened file = %d",a) //fp is the pointer to

//open file with w flag.

fscanf(fp,"Integer printed in opened file=%d",&a) //fp is the pointer to

//open file with w flag.

3.2.3 Simple math library functions:

sqrt

int a=sqrt(4) //gives square root of input which is 2 in this case

exp:

//exp only supports **integer** arguments

float b=exp(2) //computes e(2.71828) raised to the power of the

//given argument

pow:

// pow only supports **integer** arguments

int a=pow(3,2) //computes first argument raised to the power of the

//second argument which is 9 here.

abs:

int a=abs(-10) //computes absolute value of argument which is 10

floor:

int a=floor(10.24) //computes greatest integer less than argument

ceil:

int a=ceil(10.24) //computes smallest integer greater than argument

- Standard C operator Precedence order will be followed.

4. Statements

4.1 Labels - Does not interfere with other identifier names. C standard - label must be followed by at least one statement, possibly a null statement. GCC will compile code that does not meet this requirement, but be aware that if you violate it, your code may have portability issues.

4.2 Expression Statements:
Ex- x++; or y = y+1;

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

4.4 for loop
for (initialize; test; step)
 statement

4.5 switch statement
switch (test)
{
 case compare-1:
 If-equal-statement-1
 case compare-2:
 If-equal-statement-2
 ...
 default:
 default-statement
}

4.6 while loop
while (test)
 statement

4.7 do while loop - Is it necessary?
do
 statement
while (test);

4.8 Blocks

A block is a set of zero or more statements enclosed in braces. Blocks are also known as compound statements.

Example:

```
{  
    Statements;  
}
```

4.10 goto statement
goto label;

4.11 break
You can use the break statement to terminate a while, do, for, or switch statement.

4.12 continue

4.13 return statement

5. Functions

5.1 Function declaration
Syntax:
return_type function_name(parameter list);
Example:
int hello(int a);

5.2 Function definition
Syntax -
return_type func_name(parameters here) {
 ...
}
Example:
int hello(int a){
 a=3;
 return a;
}

5.3 Functions calls
Syntax -
return_type variable_name = func_name(parameters list)

Example:

```
Int tmp=hello(1);
```

5.4 Function parameters

The compiler does **not** support variable length function parameters.

5.5 main function

Every program to be compiled must have a main function. There is no need for a declaration of main function

5.7 Recursive function

Example:

```
int f1(int a){
    if(a<0){
        return 0;
    }
    else{
        a=a-1;
        f1(a);
    }
}
```

5.8 Nested function

Example:

```
Int f2(int b){
    b=3;
    return b;
}
int f1(int a){
    f2(a);
    return 0;
}
```

5.9 I/O Functions (printf,scanf)

5.10 File I/O Functions (fprintf,fscanf)

Format identifier for I/O

%c	char
%d	int
%f	float

%hi	signed integer (short)
%hu	unsigned Integer (short)
%ld	long
%lf	double
%lu	unsigned long
%lld	long long
%llu	unsigned long long
%s	string
%u	unsigned int
%b	bool

6. Macros

The compiler would support macros defined using `#define`.

7. Scope rules

Standard C scope rules are followed.

References:

- <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Program-Structure>
- <https://www.tutorialspoint.com/format-specifiers-in-c>
- <https://codeforwin.org/2015/05/list-of-all-format-specifiers-in-c-programming.html>