COMPILER DESIGN PROJECT TEST PLAN:

TEAM 5:

Sripranav Mannepali	(CS18036)
Chirag Gupta	(CS18006)
AVS Hrudai	(CS18001)
SRVS Maheswara Reddy	(CS18032)

TEST PLAN :

INTRODUCTION :

We have developed a custom compiler using lex , yacc and nasm tools. We have defined our language and did all the stepwise analysis:

1) lexical analysis
2) syntax analysis
3) semantic analysis
4) Assembly (x86 64 bit nasm) code generation.

CONTENTS:

The Project code is entirely written in a folder named “CUSTOM_COMPILER”.

constituents:

Lex.l is the lex file.
Yacc.y is the yacc file.
Main.cpp and **registers.cpp** are cpp files.
Main.h is a header file.
Make file
Test_script.py is a python file
TEST folder

TEST ITEMS :

We have all the test programs in the "TEST" folder. We have 33 programs.

FEATURES TO BE TESTED :

- Integer declaration
- scanning
- printing
- arithmetic and bitwise operations on integers
- float declaration
- arithmetic operations on floats
- character declaration
- character printing
- if
- if-else
- while loops
- for loops
- list (1D array) declarations
- list element accessing and modification
- list printing
- list arithmetics
- list size operator
- matrix (2D array) declarations

- matrix printing
- matrix arithmetics
- functions with integer return values with any number of arguments
- Least recently used register policy to reduce memory calls.

APPROACH :

We wrote all the test cases in the TEST folder.
Run “make clean “
“Make”
“./a.out < TEST/ {file name} “
“Make run nasm”
({file name} is the name of the file that needs to be tested)

To check the output of the particular test case.

The generated assembly code is in the “**gen.asm**” file.

We also wrote a python script to automate the testing process.
Run “ make clean “
“ make “
“ python3 test_script.py “ **or** “ make autorun “

This script iterates over all the test cases and automatically runs all the general test files.

Item pass/fail criteria :

If the output matches the expected output for the program , then we can say that the test case is passed. (assuming the program follows our specified grammar. If not, an error is generated. Also , scope of our project is to be kept in mind.We discussed our scope in our language manual and report)

Testing tasks:

We made the following test programs to test all the features of our compiler.

General test cases

In the TEST folder,

From test 1 to test 30 covers all the basic general test cases.

1) test 1 to test 8 checks basic integer and float operations.
2) test 9 to test 14 chesks list operations.
3) test 15 to test 19 checks Matrix operations.
4) test 20 to test 23 checks loops.
5) test 24 to test 26 checks functions.
6) Test 28 checks character printing.
7) Test 29 and test 30 checks list sorting in 1 dimension.
8) Test 27 checks register allocation and with reduced memory calls.

Real Life Test Cases

In the TEST folder,

1) test_BubbleSort_scan_ : takes Array(1D) input from the user and outputs the sorted array and also the given input. We used the bubble sort algorithm to do this.

2) test_Bubble Sort2: checks the code bubble sort on a given array.

3) test_NthFibonacci : finds the Fibonacci of a given number which is scanned (user gives the input) and prints the nth Fibonacci .

4) test_NFactorial : finds the Factorial of given number which is scanned (user gives the input) and prints the factorial .

5) test_Gcd_scan : finds the GCD of two numbers which are scanned (user gives the input) and prints the GCD .

6) test_List_Palindrome :checks if list is palindrome or not.

We also have a python script "test_script.py" which iterates over all the general test cases and shows their output on shell.(One at a time with 3 seconds gap)
This script automates the testing process.

Environmental needs :

Linux OS 64 bit.

DEVELOPER ENVIRONMENT :

OS NAME : Ubuntu 20.04.2 LTS
OS TYPE : 64-bit
Processor : Intel® Core™ i7-8550U CPU @ 1.80GHz × 8