**Compiler Design**

**For**

**Subset of C language**

**Table of contents**

**1. The Project: overview……………………………………………………………………………..…03**

* 1. **Goals and motivation ……………………….………………………………………….…..03**
  2. **Implementation language………………….………………………………………….…....03**

**2. Required Tools …………………………………………………………………………………….…03**

**2.1 Flex –Scanner……………………………………………………………………….………03**

**2.2 Bison –Parser……………………………………………………………………………..…03**

**2.3 SPIM- A MIPS Simulator……………………………………………………………..……04**

**3. Language definition**

**3.1 Included in Sub set of C-………………………………………………………………...….05**

**3.2 Lexical elements……………………………………………………………………….…….05**

**3.3 Syntax………………………………………………………………………………….…..…05**

**4. Organization of the project…………………………………………………….………………….…06**

**4.1 Lexical analysis ……………………………………………………………....…..……….…06**

**4.2 Parser (Syntax analysis)……………………………………………...…………..……….…06**

**4.3 Generation of machine code (MIPS) …………………………………………………….…06**

## **5. How to run……….……………………………..………………………………….…………………..07**

**5.1. Folder containing files…….………………………………………….………………..……07**

**5.2. On windows system…….…………………………………………….………………..….…07**

**5.3. On Linux system ……………………………………………….……………………….…..07**

**5.4. Example ………………………………………………………....…….…………..………...08**

**6. Some test program…..………………………..…………………….……......………………………...11**

**7. References ……………………….……………………………………………..…………………..…..**13

**Compiler Design for subset of C language**

**The Project: overview**

* The project consists of-
  + lexical analysis
  + Syntax analysis (parsing and generation of syntax tree)
  + Semantic analysis
  + Code generation

**Goals and motivation:**

* A complete compiler from source code to machine code

Motivation:

* Understand all parts of a compiler
* Understand the execution of a program
* Experience of a larger programming project.

**Implementation language:**

Subset of C programming language (common feature of the C language)

**Required Tools:**

1. flex -Scanner
2. bison -Parser
3. SPIM- A MIPS Simulator

**TOOL DESCRIPTION:**

## **Flex Program:**

Flex is a tool for generating scanners: programs which recognized lexical patterns in text. Flex reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called rules. Flex generates as output a C source file, `lex.yy.c', which defines a routine `yylex ()'. This file is compiled and linked with the `-lfl' library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

## **Bison program:**

## Bison is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR(1) parser tables. As an experimental feature, Bison can also generate IELR(1) or canonical LR(1) parser tables. Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change.

Input files should follow the yacc convention of ending in .y. Unlike yacc, the generated files do not have fixed names, but instead use the prefix of the input file. For instance, a grammar description file named parse.y would produce the generated parser in a file named parse.tab.c, nstead of yacc's y.tab.c.

**SPIM program:**

Spim is a self-contained simulator that runs MIPS32 programs. It reads and executes assembly language programs written for this processor. Spim also provides a simple debugger and minimal set of operating system services. Spim does not execute binary (compiled) programs.

Spim implements almost the entire MIPS32 assembler-extended instruction set. (It omits most floating point comparisons and rounding modes and the memory system page tables.) The MIPS architecture has several variants that differ in various ways (e.g., the MIPS64 architecture supports 64-bit integers and addresses), which means that Spim will not run programs for all MIPS processors.

Spim comes with complete source code and documentation.

Spim implements both a terminal and windows interfaces. On Microsoft Windows, Linux, and Mac OS X, the spim program offers a simple terminal interface and the QtSpim program provides the windowing interface.

**QtSpim**

The newest version of Spim is called QtSpim, and unlike all of the other version, it runs on Microsoft Windows, Mac OS X, and Linux—the same source code and the same user interface on all three platforms! QtSpim is the version of Spim that currently being actively maintained. The other versions are still available, but are no longer maintained or updated.

**Language definition:**

**Included in Sub set of C-**

* data types: void, int, bool, char, string
* variables, functions
* if-statements, while loops, for loops
* expressions: constants, identifiers,
* literals: characters and integers
* Logical Expressions involving '&&' and '||'
* Relational operators: '>', '<', '>=', '<=', '==', '<>', '!='
* Arithmetic operators : '+', '-', '\*', '/', '%'
* Unary Operators : '+', '-'
* read (only int)and write(int and string literal) operation

**Lexical elements**

* Decimal integer constants and character literals. A character literal contains either a single printable character, or the \n escape sequence (line break). A character literal denotes an integer constant whose value is the representation code of the character.
* Alphanumeric identifiers: non-empty sequences of letters or digits starting with a letter. An underscore is treated as a letter.
* Keywords: char, else, if, int, return, void, and while.
* Special symbols: !=, !, &&, (, ), \*, +, , (comma), -, /, ;, <=, <, ==, =, >=, >, [, ], {, }.
* White space characters: blank (32), newline (10), carriage return (13), form feed (12), and tab (9). The numbers are the ASCII representation codes for the characters.

Comments: /\* followed by anything and terminated by \*/, and // followed by anything and terminated by end of line.

**Syntax**

* Primary expressions: constants, identifiers, function calls, array indexing, and expressions within parentheses.

Unary expressions with the! and - unary operators.

Binary expressions with the +, -, \*, /, <, >, <=, >=, ==, !=, &&, and = operators.

Statements: expression statements, the empty statement, if statements with or without else, while statements, return statements, and compound statements (blocks), i.e., statements enclosed in { }.

Local variable declarations are only permitted at the top-level function body block, not in nested blocks.

* Variable declarations: base type followed by variable name, and for arrays followed by the array size (an integer constant) in square brackets.

Multi-dimensional arrays, pointers, and structures are not included.

Initializes in variable declarations are not included.

* Function definitions: return type or void, function name, parameter list, and body (compound statement) in that order.

The parameter list is either void, meaning no parameters, or a sequence of variable declarations separated by, (comma). An array parameter in a function head is written without array size, i.e., with only the brackets.

An external (library) function can be declared by writing a function definition without body, terminated with a ; (semi-colon).

**Program execution**

* Execution starts at the user-defined function main which takes no parameters and returns int. Execution ends when main returns.

**Organization of the project**

* lexical analysis
* parser
* semantic analysis
* generation of machine code

**Lexical analysis**

Input: A C source file

Output: A "stream" of tokens, errors

**Parser (Syntax analysis)**

Input: A stream of tokens

Output: Syntax tree, errors

**Semantic analysis**

Input: Syntax tree

Output: errors (or a syntax tree that is known to be semantically correct)

**Generation of machine code (MIPS)**

Input: Abstract Syntax tree

Output: A file containing MIPS assembly code

**How to run**

**Folder containing files**

* 1. l.l - lexer file contains language pattern
  2. y.y - parser file contains grammar rules and necessary code convert c to MIPS
  3. CtoMIPS.h – Abstract syntax tree, symbol table, ctoMIPS code all integrated into one file

**On Windows system**

1. Open command prompt and go to the folder containing the files and run the following command
2. bison -d y.y
3. flex l.l
4. g++ -g -std=c++11 lex.yy.c y.tab.c -o a.exe
5. a.exe < code.c (file containg code)

or

* Simply run the a.cmd file and then type in the command prompt
* a.exe < code.c (file containg code)

**On Linux system**

1. Open terminal and go to the directory containing the files and run the following command
2. bison -d y.y
3. flex l.l
4. g++ -g -std=c++11 lex.yy.c y.tab.c -o a
5. ./a < code.c (file containg code)

Or

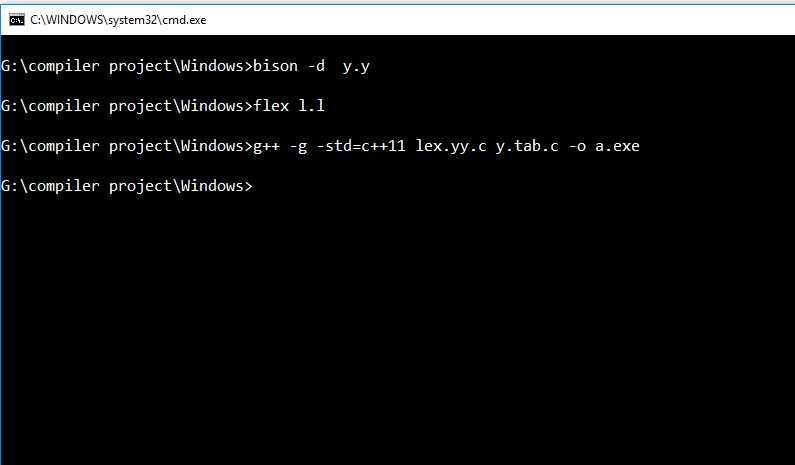
* Simply open terminal and go to the directory containing the files and just type
* make
* and then type
* ./a < code.c (file containg code)

Check MIPS file in QtSIMP

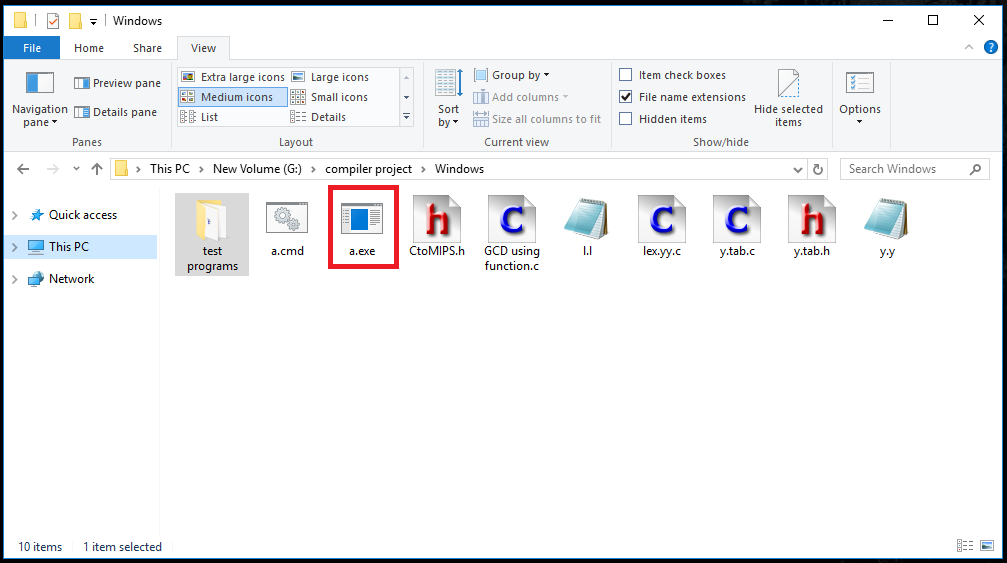
Open QtSIMP and load the MIPS file and run

**Example**

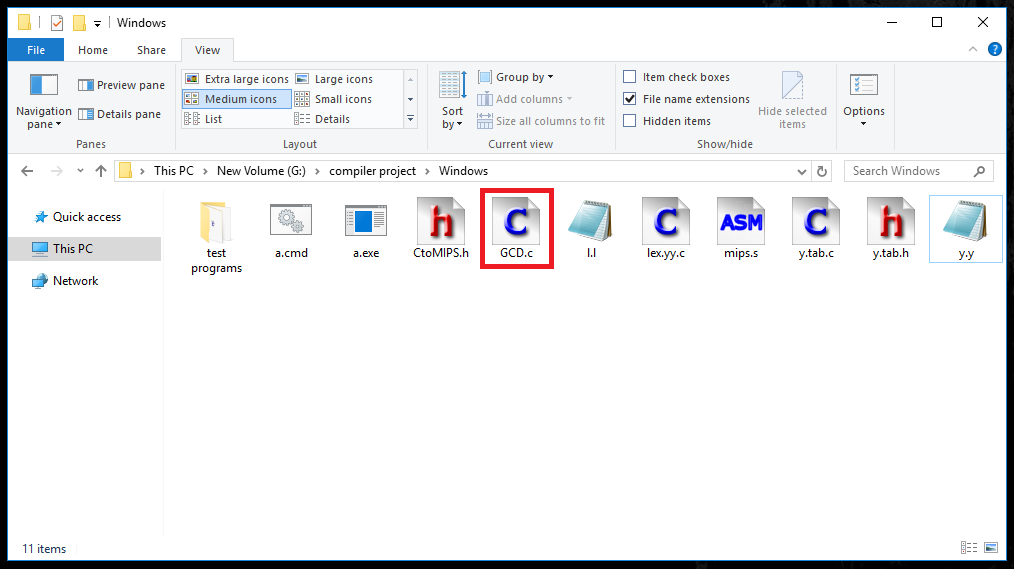
* 1. Open command prompt and go to the folder containing the files and run the following command
  2. bison -d y.y
  3. flex l.l
  4. g++ -g -std=c++11 lex.yy.c y.tab.c -o a.exe



a.exe will be created in the same folder

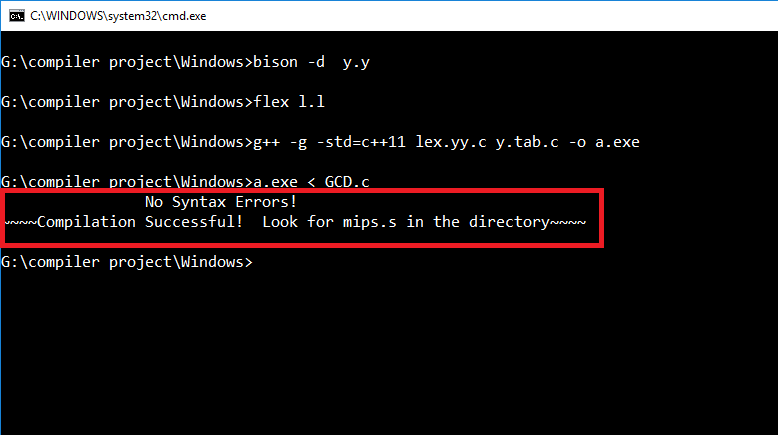


Run a code through the a.exe executable e.g. GCD.c

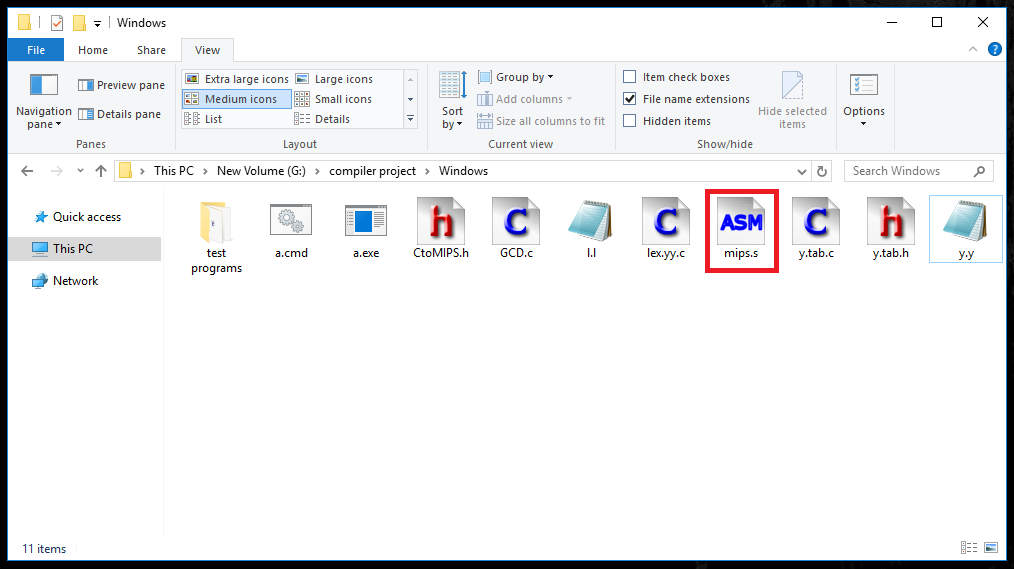


a.exe < code.c (file containg code)

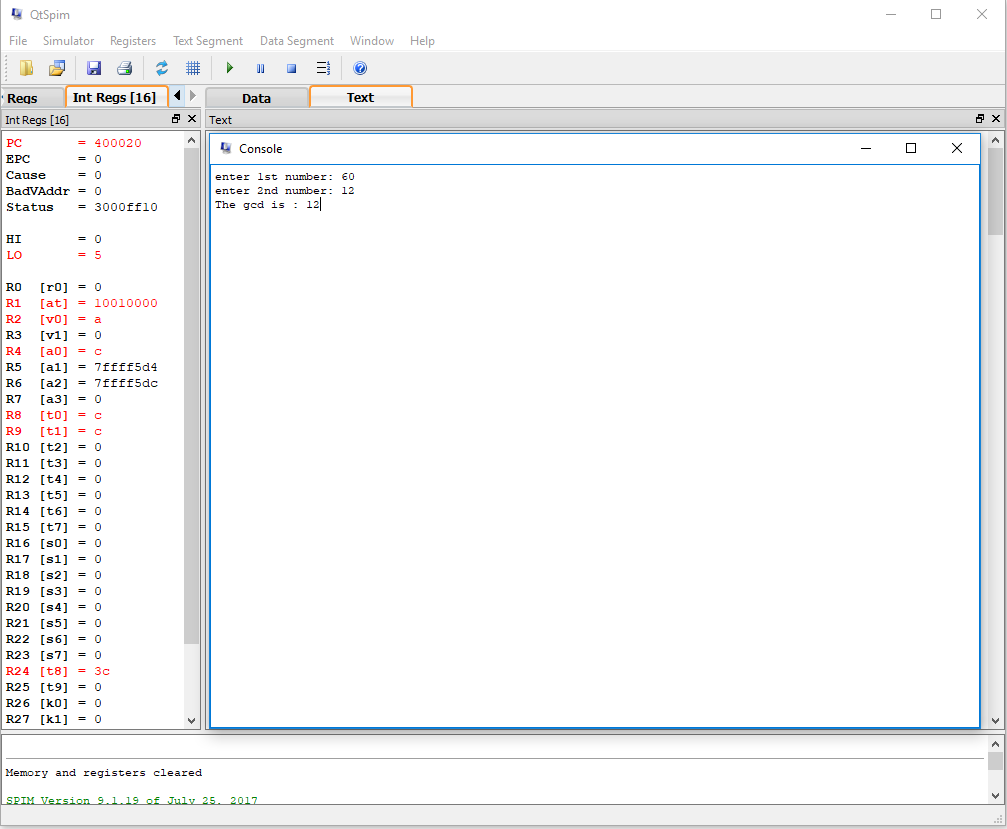
If the program is correct according to the language definition there will be a compilation successful message.



And mips.s file will be created in the same folder.



Run the mips.s through the QtSIMP program



**Some test program**

* 1. Hello World

#include<stdio.h>

main(){

puts("hello world")

return 0;

}

* 1. GCD of two number

#include<stdio.h>

int find\_gcd(int x, int y){

if(x%y == 0){

puts("The gcd is :");

put(y);

}

else{

return find\_gcd(y,x%y);

}

}

main(){

int a,b,c;

puts("enter 1st number: ");

get(a);

puts("enter 2nd number: ");

get(b);

if(a<b){

c=a;

a=b;

b=c;

}

find\_gcd(a,b);

return 0;

}

* 1. Multiplication table

#include<stdio.h>

main(){

int a,b,i;

puts("enter a number: ");

get(a);

for(i=1;i<11;i=i+1){

b=a\*i;

put(a);

puts("X");

put(i);

puts("=");

put(b);

puts(" ");

}

return 0;

}

* 1. Sum of n Numbers

#include<stdio.h>

main(){

int a;

int s=0;

int i=1;

puts("enter a number: ");

get(a);

while(i<=a){

s=s+i;

i=i+1;

}

puts("Sum is = ");

put(s);

return 0;

}

* 1. Even or Odd

#include<stdio.h>

main(){

int a,c;

puts("enter a number: ");

get(a);

c=a%2;

if(c==0){

puts("number is even");

}

if(c!=0){

puts("number is odd");

}

return 0;

}

References:

<http://www.cs.man.ac.uk/~pjj/cs211/ho/node28.html>

<https://lambda.uta.edu/cse5317/notes/node1.html>

<http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm>

<https://www.assemblylanguagetuts.com/mips-assembly-programming-tutorials/>