CS348-Assignment 5

Sweeya Reddy 200101079
Vatsal Gupta 200101105

README

25th April 2023

ZIP FILE CONTENTS

1. A5_40_translator.cpp
2. A5_40_translator.h
3. A5_40.l
4. A5_40.y
5. Makefile
6. test-inputs (dir)
    a. A5_40_test1.nc
    b. A5_40_test2.nc
    c. A5_40_test3.nc
    d. A5_40_test4.nc
    e. A5_40_test5.nc
    f. A5_40_test6.nc
7. test-outputs (dir)
    a. A5_40_quads1.out
    b. A5_40_quads2.out
    c. A5_40_quads3.out
    d. A5_40_quads4.out
    e. A5_40_quads5.out
    f. A5_40_quads6.out
8. Readme.pdf

** Note some files are created on running the make command these are auto-generated intermediaries crucial to the running of the program, they are given in the submission and can be remade by using the make commands.*

PROGRAMMING LANGUAGE USED (upon UBUNTU 20.04 LTS)

nanoC, C,C++(with std Lex/Flex (lexer), (parsing)Bison

Key Points/Pre-requisites

- Please make sure the terminal is run in the same directory (cwd) as the two files.
- The program will first create A5_40.tab.c and A5_40.tab.h files from the parser A6_40.y by using the bison shell commands.
- The program will create a .c file - lex.yy.c from A5_40.l, lex file using the A5_40_translator.h & A5_40.tab.h file as index lister and headers and subsequently the required parser files like y.tab.h and auto generate token numbers and then use them to parse the input later.

  3 object files are created and subsequently used for finally generating the executable required as follows:

  - A5_40_translator.o
  - A5_40.tab.o
  - lex.yy.o

  These finally produce our nanoC executable using the g++ command:
  `g++ lex.yy.o A5_40.tab.o A5_40_translator.o -lfl`

- **The output is generated after creating & using an internal symbol table which is implicitly stored as a map in the c file. The printed output will display the tokens returned and the productions applied.**

Commands to enter and expected output:

```
make test
```

- Simply type make test into the command line and then the test-outputs directory is filled up with one file for each test case input, i.e, A5_40_quads<x>.out (for test case x) as given in test-inputs.
- **BONUS:** You may also use some debug statements as in Assignment-4 to print colored output onto the terminal, re-use assignment 4 code in such a scenario.

- You can also retest the output by running make clean first which removes the optional files and then run make again which recreates them proving the sample output is indeed correct. Make clean will essentially employ the following function:

```
"rm -f lex.yy.c *.tab.c *.tab.h *.output *.o a.out *.gch
test-outputs/*_quads*[^preserve].out"
```

```
make clean
```

- **BONUS**: Anything listed as _quads_preserve will be preserved and not erased, the user may manually change the file name in case they want to retain the output even through/after the make clean command is run. This was and can again be used extensively for debugging purposes.

Note:

- **BOUNS:** We have shown the output for all the 6 examples as given in the assignment they can be seen in the files labelled A5_40_quads<i>.out for the i^{th} example
- Our test files are aimed and targeted to test the following things (the codes mainly use the nanoC files already given in assignment-3)
    - The first is the code for binary search which can test appropriate symbol table creation during array access, input into array with appropriate offsets,print functionality for string as well as int and temporary local variable creation and recursion.
    - Second is the externalized swap function which checks for correct pointer access, calling and appropriate labels in goto statements of an external subroutine/function and address as the value for reference variables.
    - Third is factorial function by iteration which checks for repeated same integer access and modification subroutine and printing of the updated value by accessing the necessary address.
    - Fourth is checking max of n numbers through array checks for multi-array insertion and retrieval with appropriate comparison and new local variable storage
    - Fifth is the fibonacci program and it checks for conditionals within external subroutines and whether a return statement is appropriately tied up to different goto statements
    - Sixth simply checks if the same memory location can be accessed repeatedly with internal modifications and whether the updations in the symbol table for memory locale access is done correctly by using the bubble sort code and this in turn also aids in checking of nested for loops and temporary variable creation for incremental analysis.

Design of Grammar Augmentations & Explanation of the augmentations in the production rules in Bison Design of Attributes :

- The lexer follows a certain priority order as given in assignment 3, and the same lexer as submitted earlier has been used with minor to no modification.
- We have modified some grammar rules, to make the production rules correct and perform right. We have taken inspiration from several sources like the following segment of grammar on the right (A Grammar for the C-Programming Language (Version S21) March 23, 2021 1 Introduction)

1. $program \rightarrow declList$
2. $declList \rightarrow declList\ decl \mid decl$
3. $decl \rightarrow varDecl \mid funDecl$

- The changes are done for the translation_unit segment and the function_definition segment all the others are retained, and there are some intermediates defined as interim and interim_decl which are necessary from a code point of view to apply appropriate actions to the IDENTIFIER token when it is received before applying subsequent function_definition production and similarly interim_decl applies appropriate pre-processing to declarator.

Cases Handled:

I. Quads are generated correctly for each opcode token needed and the local as well as function and temporary symbol table creations are handled exactly as needed and in the right fashion.
II. Local arrays work correctly, global arrays are easy to handle and their quad generation is handled correctly but the goto statements are skipped for brevity and avoiding ambiguous access in assignment-6.
III. Local arrays are accessed for read and write.
IV. Pointer access for read and write and de-referencing is working in directly tying up to the symbol table and is working right.
V. sub-routines and external function calling is working flawlessly by extensive use of goto statements and backpatching
VI. If-else conditional and for loop blocks are working correctly for the scope as mentioned in the assignment.
VII. All the test-cases and what they test for is working correctly and the local data structures created in the code mimic the printed symbol table and they are used for the TAC generation as required. The working code and comments explain each data structure, member and functionality and can be reviewed.

Appendix

Output Screenshots:

1. Test case 1:

```
-------------------------------------------------------------------------------------------------------------
Symbol Table: Global                                    Parent Table: NULL
-------------------------------------------------------------------------------------------------------------
Name                    Type                    Initial Value       Size          Offset        Nested
-------------------------------------------------------------------------------------------------------------
arr                     arr(10, int)            -                   40            0             NULL
binarySearch            int                     -                   4             40            binarySearch
main                    int                     -                   4             44            main
-------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
Symbol Table: binarySearch                              Parent Table: Global
-------------------------------------------------------------------------------------------------------------
Name                    Type                    Initial Value       Size          Offset        Nested
-------------------------------------------------------------------------------------------------------------
l                       int                     -                   4             0             NULL
r                       int                     -                   4             4             NULL
x                       int                     -                   4             8             NULL
return                  int                     -                   4             12            NULL
binarySearch.$0         block                   -                   4             16            binarySearch.$0
t0                      int                     1                   4             20            NULL
t1                      int                     -                   4             24            NULL
-------------------------------------------------------------------------------------------------------------
```

2. Test case 2:

```
-------------------------------------------------------------------------------------------------------------
Symbol Table: Global                                    Parent Table: NULL
-------------------------------------------------------------------------------------------------------------
Name                    Type                    Initial Value       Size          Offset        Nested
-------------------------------------------------------------------------------------------------------------
swap                    void                    -                   0             0             swap
main                    int                     -                   4             0             main
-------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
Symbol Table: swap                                      Parent Table: Global
-------------------------------------------------------------------------------------------------------------
Name                    Type                    Initial Value       Size          Offset        Nested
-------------------------------------------------------------------------------------------------------------
p                       ptr(int)                -                   4             0             NULL
q                       ptr(int)                -                   4             4             NULL
t                       int                     -                   4             8             NULL
t0                      int                     -                   4             12            NULL
t1                      int                     -                   4             16            NULL
t2                      int                     -                   4             20            NULL
t3                      int                     -                   4             24            NULL
-------------------------------------------------------------------------------------------------------------
```

3. Test case 3:

```
Symbol Table: Global                                    Parent Table: NULL
Name                    Type                    Initial Value       Size            Offset          Nested
main                    int                     -                   4               0               main

Symbol Table: main                                      Parent Table: Global
Name                    Type                    Initial Value       Size            Offset          Nested
return                  int                     -                   4               0               NULL
n                       int                     -                   4               4               NULL
i                       int                     0                   4               8               NULL
t0                      int                     0                   4               12              NULL
r                       int                     1                   4               16              NULL
t1                      int                     1                   4               20              NULL
readInt                 int                     -                   4               24              NULL
t2                      ptr(int)                -                   4               28              NULL
t3                      int                     -                   4               32              NULL
main.FOR$0              block                   -                   4               36              main.FOR$0
printInt                int                     -                   4               40              NULL
t4                      int                     -                   4               44              NULL
printStr                int                     -                   4               48              NULL
t5                      ptr(char)               "! = "              4               52              NULL
t6                      int                     -                   4               56              NULL
t7                      int                     -                   4               60              NULL
t8                      ptr(char)               "\n"                4               64              NULL
t9                      int                     -                   4               68              NULL
t10                     int                     0                   4               72              NULL
```

4. Test case 4:

```
Symbol Table: main.FOR$0                                Parent Table: main
Name                    Type                    Initial Value       Size            Offset          Nested
t0                      int                     0                   4               0               NULL
t1                      int                     1                   4               4               NULL
t2                      int                     -                   4               8               NULL
t3                      ptr(char)               "Input next element\n"4             12              NULL
t4                      int                     -                   4               16              NULL
t5                      ptr(int)                -                   4               20              NULL
t6                      int                     -                   4               24              NULL
t7                      int                     -                   4               28              NULL

Symbol Table: main.FOR$1                                Parent Table: main
Name                    Type                    Initial Value       Size            Offset          Nested
t0                      int                     1                   4               0               NULL
t1                      int                     1                   4               4               NULL
t2                      int                     -                   4               8               NULL
t3                      int                     -                   4               12              NULL
t4                      int                     -                   4               16              NULL
t5                      int                     -                   4               20              NULL
t6                      int                     -                   4               24              NULL
```

5. Test case 5:

Symbol Table: Global — Parent Table: NULL

Name	Type	Initial Value	Size	Offset	Nested
f_odd	int	-	4	0	f_odd
f_even	int	-	4	4	f_even
fibonacci	int	-	4	8	fibonacci
main	int	-	4	12	main

Symbol Table: f_odd — Parent Table: Global

Name	Type	Initial Value	Size	Offset	Nested
return	int	-	4	0	NULL
n	int	-	4	4	NULL
t0	int	1	4	8	NULL
t1	int	1	4	12	NULL
t2	int	1	4	16	NULL
t3	int	-	4	20	NULL
t4	int	-	4	24	NULL
t5	int	2	4	28	NULL
t6	int	-	4	32	NULL
t7	int	-	4	36	NULL
t8	int	-	4	40	NULL
t9	int	-	4	44	NULL

Symbol Table: f_even — Parent Table: Global

Name	Type	Initial Value	Size	Offset	Nested
return	int	-	4	0	NULL
n	int	-	4	4	NULL
t0	int	0	4	8	NULL
t1	int	0	4	12	NULL
t2	int	1	4	16	NULL
t3	int	-	4	20	NULL
t4	int	-	4	24	NULL
t5	int	2	4	28	NULL
t6	int	-	4	32	NULL
t7	int	-	4	36	NULL
t8	int	-	4	40	NULL
t9	int	-	4	44	NULL

Symbol Table: fibonacci — Parent Table: Global

Name	Type	Initial Value	Size	Offset	Nested
n	int	-	4	0	NULL
return	int	-	4	4	NULL
t0	int	2	4	8	NULL
t1	int	-	4	12	NULL
t2	int	0	4	16	NULL
t3	int	-	4	20	NULL
t4	int	-	4	24	NULL
t5	int	-	4	28	NULL

6. Test case 6:

```
Symbol Table: Global                                    Parent Table: NULL
Name                Type                Initial Value   Size        Offset      Nested
swap                void                -               0           0           swap
readArray           void                -               0           0           readArray
printArray          void                -               0           0           printArray
bubbleSort          void                -               0           0           bubbleSort
arr                 arr(20, int)        -               80          0           NULL
main                int                 -               4           80          main

Symbol Table: swap                                      Parent Table: Global
Name                Type                Initial Value   Size        Offset      Nested
p                   ptr(int)            -               4           0           NULL
q                   ptr(int)            -               4           4           NULL
t                   int                 -               4           8           NULL
t0                  int                 -               4           12          NULL
t1                  int                 -               4           16          NULL
t2                  int                 -               4           20          NULL
t3                  int                 -               4           24          NULL
```

__Note:__ All the symbol tables can be seen in the output directory. All of them are not shown here.

Explanation of the attributes in the respective %token and %type in Bison Design and Implementation of Symbol Table & Supporting Data Structures is done in code along with the explanation of the definitions of ST & other Data Structures & Design and Implementation of Quad Array:

Explain with definition of QA Design and Implementation of Global Functions is done above as well as in the code.

Explain i/p, o/p, algorithm & purpose for every function Design and Implementation of Semantic Actions:

Explain with every action in Bison Expression Phase is done in assignment code.

Correct handling of operators, type checking & conversions Declaration Phase: Handling of variable declarations, function definitions in ST Statement Phase: Correct handling of statements External Definition Phase

Correct handling of function definitions Design of Test files and correctness of outputs is done above.

“Test at least 5 i/p files covering all rules Shortcoming and / or bugs, if any, should be highlighted” this is done extensively in readme as above.