Christopher McDaniel

COSC 4316

Submitted: 30 April 2020

Grading Option: "**A**"

Table of Contents

--“C” Options--------------------------------------

-------------------C1 Java 0---------------

```
CLASS C1
{
   VAR ans, a, Bob, Jane, b, c, N, fact;

   /* 1. Read input and compute the result of the equation below. */
   READ a;
   READ b;
   READ c;
   READ Bob;
   READ Jane;

   ans = a * ((Bob + Jane - 10) / 2 * 4) / (b + c);

   WRITE ans;
}
```

----------------C1 Assembly----------------

```
sys_exit equ        1
sys_read equ        3
sys_write           equ        4
stdin               equ        0 ; default keyboard
stdout              equ        1 ; default terminal screen
stderr              equ        3

section .data                  ;used to declare constants
          userMsg              db  'Enter an integer(less than 32,765): '
          lenUserMsg           equ        $-userMsg
          displayMsg           db         'You entered: '
          lenDisplayMsg        equ        $-displayMsg
          newline              db         0xA        ; 0xA 0xD is ASCII <LF><CR>
          Ten       DW      10  ;Used converting to base ten.
          Result        db      'Ans = '
          ResultValue                     db         'aaaaa'
                                    db  0xA                    ;return
          ResultEnd      equ            $-Result   ; $=> here - address Result = length to print
          num                           times 6    db 'ABCDEF' ;cheat NASM
          numEnd             equ        $-num

section   .bss        ;used to declare uninitialized variables
          TempChar       RESB    1 ;1 byte temp space for use by GetNextChar
          testchar       RESB    1
          ;Temporary storage GetAnInteger.
          ReadInt        RESW    1 ;4 bytes
          ;Used in converting to base ten.
          tempint        RESW          1
          negflag        RESB    1        ;P=positive, N=negative
          ans       RESW 1
          a         RESW 1
          Bob       RESW 1
          Jane      RESW 1
          b         RESW 1
          c         RESW 1
          N         RESW 1
          fact      RESW 1
          T1        RESW 1
          T2        RESW 1
          T3        RESW 1
          T4        RESW 1
          T5        RESW 1
          T6        RESW 1
          T7        RESW 1
          T8        RESW 1
```

```
section .txt            ;Start of the main program-----------------------------------------.

   global _start
_start:   call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [a], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [b], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [c], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [Bob], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [Jane], ax

          mov ax, [Bob]
          add ax, [Jane]
          mov [T1], ax
          mov ax, [T1]
          sub ax, 10
          mov [T1], ax

          mov dx, 0
          mov ax, [T1]
          mov bx, 2
          div bx
          mov [T1], ax

          mov ax, [T1]
          mov bx, 4
          mul word bx
          mov [T1], ax

          mov ax, [a]
          mul word [T1]
          mov [T1], ax

          mov ax, [b]
          add ax, [c]
          mov [T2], ax

          mov dx, 0
          mov ax, [T1]
          mov bx, [T2]
          div bx
          mov [ans], ax

          mov ax, [ans]
          call ConvertIntegerToString
```

```
        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

fini:
        mov eax, sys_exit
        xor ebx, ebx
        int 80h

;PrintString   PROC
PrintString:
        push   ax          ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4                ;Linux print device register conventions
        mov ebx, 1                ; print default output device
        mov ecx, userMsg          ; pointer to string
        mov edx, lenUserMsg       ; arg1, where to write, screen
        int      80h              ; interrupt 80 hex, call kernel
        pop    dx          ;Restore registers.
        pop    ax
        ret
;PrintString   ENDP

;GetAnInteger   PROC

GetAnInteger:    ;Get an integer as a string
        ;get response
        mov eax,3           ;read
        mov ebx,2           ;device
        mov ecx,num         ;buffer address
        mov edx,6           ;max characters
        int 0x80

        ;print number    ;works
        mov edx,eax         ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0            ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num         ;pt - 1st or next digit of number as a string
                                    ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx              ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1           ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA          ;is it a <lf>
```

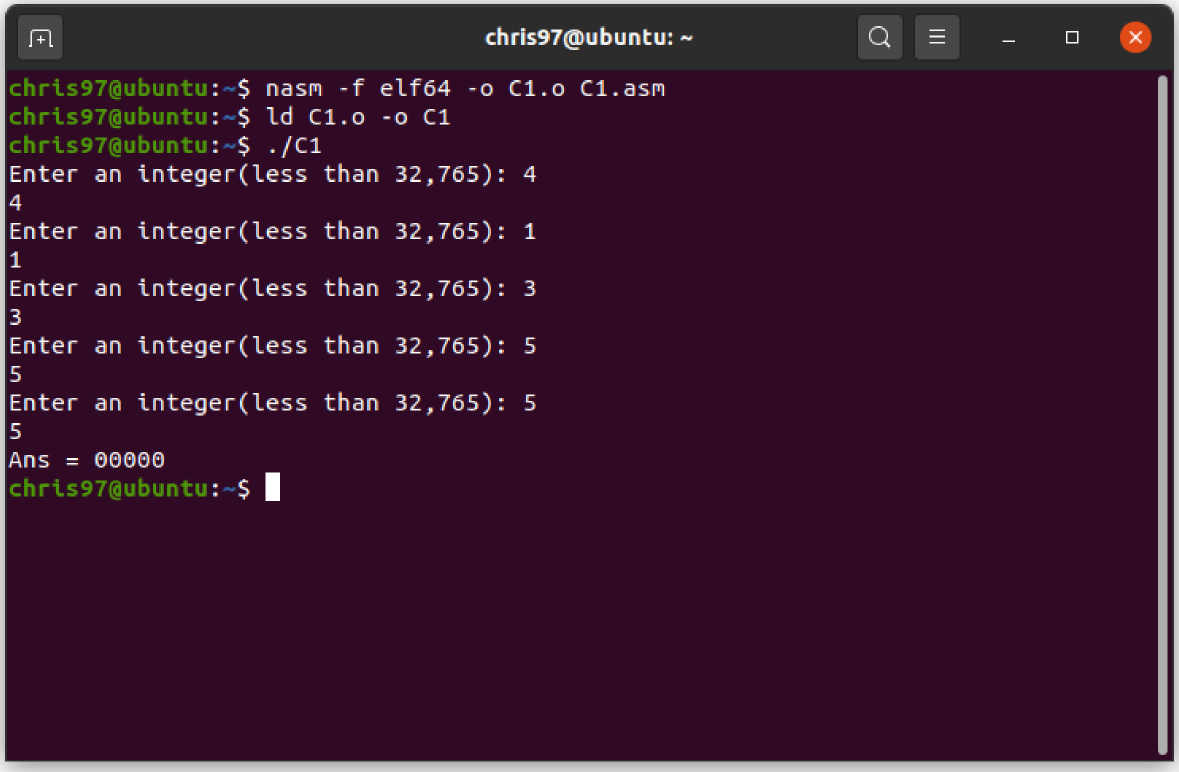
```
            jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7

            ret

;           ENDP GetAnInteger


;ConvertIntegerToString PROC


ConvertIntegerToString:

            mov ebx, ResultValue + 4   ;Store the integer as a five

                                        ; digit char string at Result for printing


ConvertLoop:

            sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number

            mov cx,10  ; as the remainder in the DX register.  Quotient in AX.

            div cx

            add dl,'0' ; Add '0' to dl to convert from binary to character.

            mov [ebx], dl

            dec ebx

            cmp ebx,ResultValue

            jge ConvertLoop


            ret


;ConvertIntegerToString  ENDP
```

----------C1 Symbol Table---------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
C1	<ProgramName>		0	CS
ans	<var>	?	0	DS
a	<var>	?	2	DS
Bob	<var>	?	4	DS
Jane	<var>	?	6	DS
b	<var>	?	8	DS
c	<var>	?	10	DS
N	<var>	?	12	DS
fact	<var>	?	14	DS
lit10	<integer>	10	16	DS
lit2	<integer>	2	18	DS

lit4	<integer>	4	20	DS
T1	<temp>	?	22	DS
T2	<temp>	?	24	DS
T3	<temp>	?	26	DS
T4	<temp>	?	28	DS
T5	<temp>	?	30	DS
T6	<temp>	?	32	DS
T7	<temp>	?	34	DS
T8	<temp>	?	36	DS

---------C1 Screenshot---------



```
chris97@ubuntu:~$ nasm -f elf64 -o C1.o C1.asm
chris97@ubuntu:~$ ld C1.o -o C1
chris97@ubuntu:~$ ./C1
Enter an integer(less than 32,765): 4
4
Enter an integer(less than 32,765): 1
1
Enter an integer(less than 32,765): 3
3
Enter an integer(less than 32,765): 5
5
Enter an integer(less than 32,765): 5
5
Ans = 00000
chris97@ubuntu:~$ 
```

------------------C2 Java 0---------------

```
CLASS C2

{

    VAR a, b;


    /* 2. Read two integers from the terminal and print the largest value. */

    READ a;

    READ b;


    IF a > b THEN WRITE a;

    IF b > a THEN WRITE b;

}
```

----------------C2 Assembly----------------

```
sys_exit equu        1

sys_readequ          3

sys_write            equ      4

stdin                equ      0 ; default keyboard

stdout               equ      1 ; default terminal screen

stderr               equ      3


section .data                 ;used to declare constants

             userMsg          db  'Enter an integer(less than 32,765): '

             lenUserMsg       equ      $-userMsg

             displayMsg       db       'You entered: '

             lenDisplayMsg    equ      $-displayMsg

             newline          db       0xA      ; 0xA 0xD is ASCII <LF><CR>


             Ten      DW     10  ;Used converting to base ten.

             Result        db     'Ans = '

             ResultValue               db       'aaaaa'

                                   db  0xA                ;return

             ResultEnd      equ        $-Result   ; $=> here - address Result = length to print

             num                       times 6    db 'ABCDEF' ;cheat NASM
```

```
            numEnd          equ       $-num


section     .bss        ;used to declare uninitialized variables
            TempChar        RESB    1 ;1 byte temp space for use by GetNextChar
            testchar        RESB    1
            ;Temporary storage GetAnInteger.
            ReadInt         RESW    1 ;4 bytes
            ;Used in converting to base ten.
            tempint         RESW          1
            negflag         RESB    1         ;P=positive, N=negative
            a           RESW 1
            b           RESW 1
            T1          RESW 1
            T2          RESW 1
            T3          RESW 1
            T4          RESW 1
            T5          RESW 1
            T6          RESW 1
            T7          RESW 1
            T8          RESW 1

section .txt           ;Start of the main program-------------------------------------------.

   global _start
_start:     call PrintString
            call GetAnInteger
            mov ax, [ReadInt]
            mov [a], ax

            call PrintString
            call GetAnInteger
            mov ax, [ReadInt]
```

```
        mov [b], ax

        mov ax, [a]
        cmp ax, [b]
        jle L1

        mov ax, [a]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

L1:     nop
        mov ax, [b]
        cmp ax, [a]
        jle L2

        mov ax, [b]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

L2:     nop
fini:
        mov eax, sys_exit
```

```
          xor ebx, ebx
          int 80h

;PrintString    PROC
PrintString:
          push   ax          ;Save registers;
          push   dx
; subpgm:
          ; prompt user
          mov eax, 4                  ;Linux print device register conventions
          mov ebx, 1                  ; print default output device
          mov ecx, userMsg            ; pointer to string
          mov edx, lenUserMsg         ; arg1, where to write, screen
          int      80h                ; interrupt 80 hex, call kernel
          pop    dx          ;Restore registers.
          pop    ax
          ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:    ;Get an integer as a string
          ;get response
          mov eax,3       ;read
          mov ebx,2       ;device
          mov ecx,num     ;buffer address
          mov edx,6       ;max characters
          int 0x80

          ;print number   ;works
          mov edx,eax     ; eax contains the number of character read including <lf>
          mov eax, 4
```

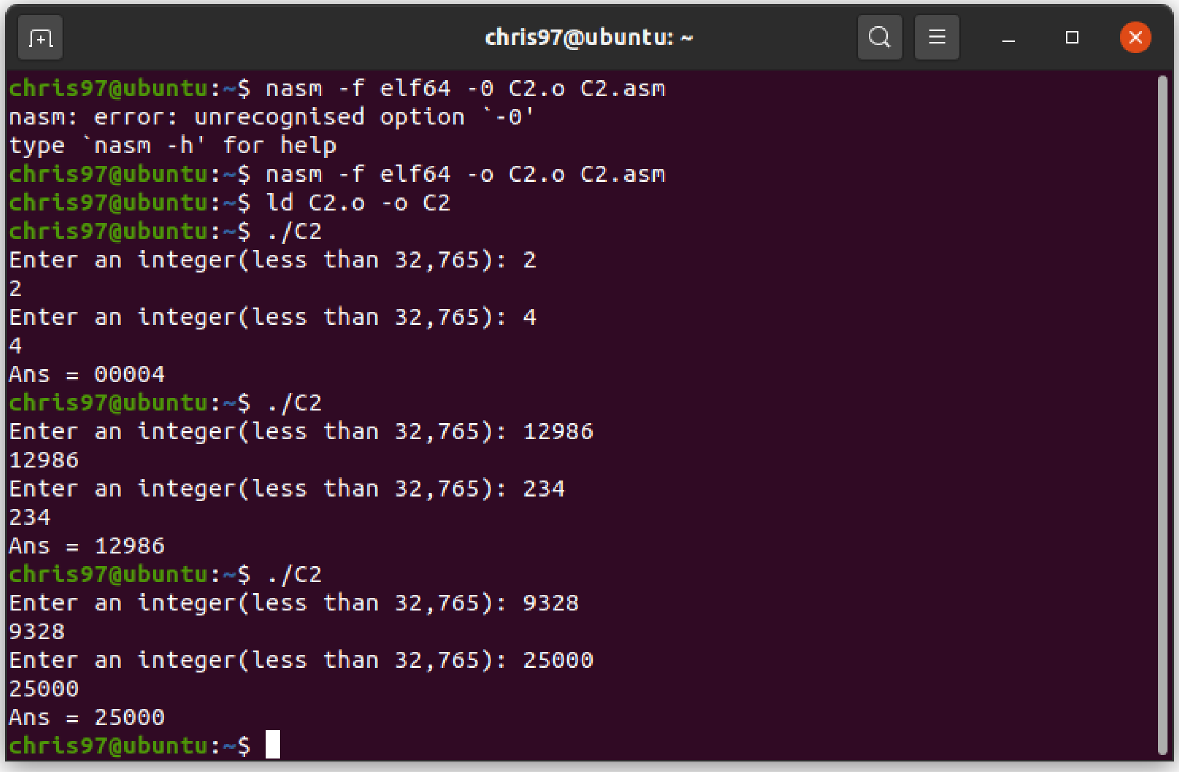
```
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0            ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num         ;pt - 1st or next digit of number as a string
                                    ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx              ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1           ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA          ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                     ; digit char string at Result for printing
```

```
ConvertLoop:

        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number

        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.

        div cx

        add dl,'0' ; Add '0' to dl to convert from binary to character.

        mov [ebx], dl

        dec ebx

        cmp ebx,ResultValue

        jge ConvertLoop


        ret


;ConvertIntegerToString  ENDP
```

--------------------C2 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
C2	<ProgramName>		0	CS
a	<var>	?	0	DS
b	<var>	?	2	DS
T1	<temp>	?	4	DS
T2	<temp>	?	6	DS
T3	<temp>	?	8	DS
T4	<temp>	?	10	DS
T5	<temp>	?	12	DS
T6	<temp>	?	14	DS
T7	<temp>	?	16	DS
T8	<temp>	?	18	DS

--------------------C2 Screenshot--------------------



```
chris97@ubuntu:~$ nasm -f elf64 -O C2.o C2.asm
nasm: error: unrecognised option `-O'
type `nasm -h' for help
chris97@ubuntu:~$ nasm -f elf64 -o C2.o C2.asm
chris97@ubuntu:~$ ld C2.o -o C2
chris97@ubuntu:~$ ./C2
Enter an integer(less than 32,765): 2
2
Enter an integer(less than 32,765): 4
4
Ans = 00004
chris97@ubuntu:~$ ./C2
Enter an integer(less than 32,765): 12986
12986
Enter an integer(less than 32,765): 234
234
Ans = 12986
chris97@ubuntu:~$ ./C2
Enter an integer(less than 32,765): 9328
9328
Enter an integer(less than 32,765): 25000
25000
Ans = 25000
chris97@ubuntu:~$
```

------------------C3 Java 0---------------

```
CLASS C3
{
    VAR a, b, c;

    /* 3. Get three integers as input, print the largest (nested IF). */
    READ a;
    READ b;
    READ c;

    IF a > b THEN
    {
        IF a > c THEN
              {
            WRITE a;
        }
    }

    IF b > a THEN
    {
        IF b > c THEN
              {
            WRITE b;
        }
    }

    IF c > a THEN
    {
        IF c > b THEN
              {
            WRITE c;
        }
    }
}
```

----------------C3 Assembly----------------

```
sys_exit equu     1

sys_read equ      3

sys_write         equ       4

stdin             equ       0 ; default keyboard

stdout            equ       1 ; default terminal screen

stderr            equ       3


section .data               ;used to declare constants

        userMsg             db  'Enter an integer(less than 32,765): '

        lenUserMsg          equ       $-userMsg

        displayMsg          db        'You entered: '

        lenDisplayMsg       equ       $-displayMsg

        newline             db        0xA       ; 0xA 0xD is ASCII <LF><CR>


        Ten       DW      10  ;Used converting to base ten.

        Result          db      'Ans = '

        ResultValue                  db        'aaaaa'

                                  db  0xA                 ;return

        ResultEnd       equ           $-Result   ; $=> here - address Result = length to print

        num                           times 6    db 'ABCDEF' ;cheat NASM

        numEnd              equ       $-num


section   .bss        ;used to declare uninitialized variables

        TempChar        RESB    1 ;1 byte temp space for use by GetNextChar

        testchar        RESB    1

        ;Temporary storage GetAnInteger.

        ReadInt         RESW    1 ;4 bytes

        ;Used in converting to base ten.

        tempint         RESW         1
```

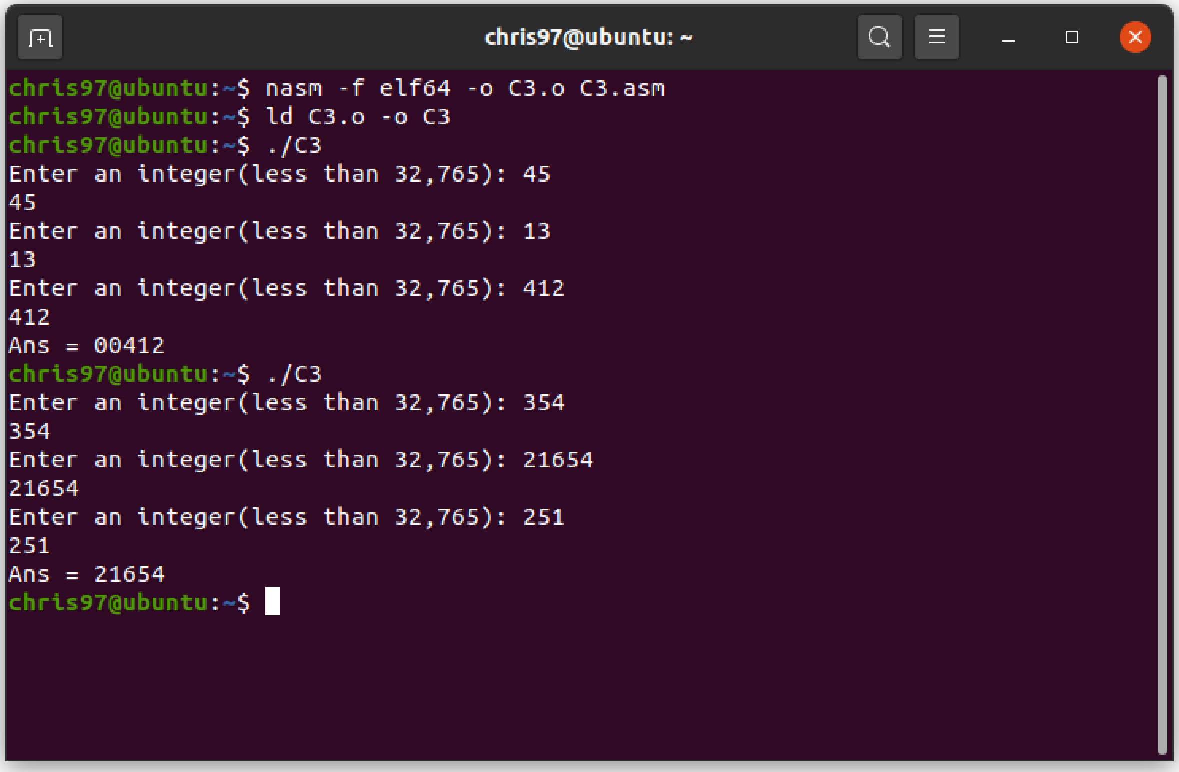
```
          negflag     RESB   1      ;P=positive, N=negative
          a         RESW 1
          b         RESW 1
          c         RESW 1
          T1        RESW 1
          T2        RESW 1
          T3        RESW 1
          T4        RESW 1
          T5        RESW 1
          T6        RESW 1
          T7        RESW 1
          T8        RESW 1

section .txt          ;Start of the main program------------------------------------------.

  global _start
_start:   call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [a], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [b], ax

          call PrintString
          call GetAnInteger
          mov ax, [ReadInt]
          mov [c], ax

          mov ax, [a]
```

```
        cmp ax, [b]
        jle L1

        mov ax, [a]
        cmp ax, [c]
        jle L2

        mov ax, [a]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

L2:     nop
L1:     nop
        mov ax, [b]
        cmp ax, [a]
        jle L3

        mov ax, [b]
        cmp ax, [c]
        jle L4

        mov ax, [b]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

L4:     nop
L3:     nop
        mov ax, [c]
        cmp ax, [a]
        jle L5

        mov ax, [c]
        cmp ax, [b]
        jle L6

        mov ax, [c]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

L6:     nop
L5:     nop
fini:
        mov eax, sys_exit
        xor ebx, ebx
        int 80h

;PrintString   PROC
PrintString:
        push   ax         ;Save registers;
        push    dx
; subpgm:
        ; prompt user
        mov eax, 4              ;Linux print device register conventions
        mov ebx, 1              ; print default output device
        mov ecx, userMsg        ; pointer to string
        mov edx, lenUserMsg     ; arg1, where to write, screen
        int     80h             ; interrupt 80 hex, call kernel
        pop     dx          ;Restore registers.
        pop     ax
        ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:    ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number   ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num      ;pt - 1st or next digit of number as a string
                                 ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx           ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1        ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA       ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                   ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl

        dec ebx

        cmp ebx,ResultValue

        jge ConvertLoop


        ret


;ConvertIntegerToString  ENDP
```

--------------------C3 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
C3	<ProgramName>		0	CS
a	<var>	?	0	DS
b	<var>	?	2	DS
c	<var>	?	4	DS
T1	<temp>	?	6	DS
T2	<temp>	?	8	DS
T3	<temp>	?	10	DS
T4	<temp>	?	12	DS
T5	<temp>	?	14	DS
T6	<temp>	?	16	DS
T7	<temp>	?	18	DS
T8	<temp>	?	20	DS

--------------------C3 Screenshot--------------------



```
chris97@ubuntu:~$ nasm -f elf64 -o C3.o C3.asm
chris97@ubuntu:~$ ld C3.o -o C3
chris97@ubuntu:~$ ./C3
Enter an integer(less than 32,765): 45
45
Enter an integer(less than 32,765): 13
13
Enter an integer(less than 32,765): 412
412
Ans = 00412
chris97@ubuntu:~$ ./C3
Enter an integer(less than 32,765): 354
354
Enter an integer(less than 32,765): 21654
21654
Enter an integer(less than 32,765): 251
251
Ans = 21654
chris97@ubuntu:~$ 
```

---------------------C4 Java 0---------------------

```
CLASS C4
{
   /* 4. Compute N! using a while loop. */

   VAR N, fact;

   fact = 1;

   READ N;

   WHILE N > 1 DO
   {
      fact = fact * N;
      N = N - 1;
   }

   WRITE fact;
}
```

---------------------C4 Assembly---------------------

```
sys_exit equ        1
sys_readequ         3
sys_write           equ      4
stdin               equ      0 ; default keyboard
stdout              equ      1 ; default terminal screen
stderr              equ      3

section .data                ;used to declare constants
          userMsg            db  'Enter an integer(less than 32,765): '
          lenUserMsg         equ      $-userMsg
          displayMsg         db       'You entered: '
          lenDisplayMsg      equ      $-displayMsg
          newline            db       0xA      ; 0xA 0xD is ASCII <LF><CR>
```

```
        Ten      DW     10  ;Used converting to base ten.
        Result       db     'Ans = '
        ResultValue              db       'aaaaa'
                              db  0xA               ;return
        ResultEnd     equ         $-Result   ; $=> here - address Result = length to print
        num                       times 6    db 'ABCDEF' ;cheat NASM
        numEnd          equ       $-num

section .bss      ;used to declare uninitialized variables
        TempChar      RESB    1 ;1 byte temp space for use by GetNextChar
        testchar      RESB    1
        ;Temporary storage GetAnInteger.
        ReadInt       RESW    1 ;4 bytes
        ;Used in converting to base ten.
        tempint       RESW        1
        negflag       RESB    1       ;P=positive, N=negative
        N        RESW 1
        fact     RESW 1
        T1       RESW 1
        T2       RESW 1
        T3       RESW 1
        T4       RESW 1
        T5       RESW 1
        T6       RESW 1
        T7       RESW 1
        T8       RESW 1

section .txt          ;Start of the main program-----------------------------------------.

   global _start
_start:  mov word [fact], 1
```

```
        call PrintString
        call GetAnInteger
        mov ax, [ReadInt]
        mov [N], ax

W1:     nop
        mov ax, [N]
        cmp ax, 1
        jle L1
        mov ax, [fact]
        mul word [N]
        mov [fact], ax

        mov ax, [N]
        sub ax, 1
        mov [N], ax

        jmp W1
L1:     nop
        mov ax, [fact]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

fini:
        mov eax, sys_exit
        xor ebx, ebx
        int 80h
```

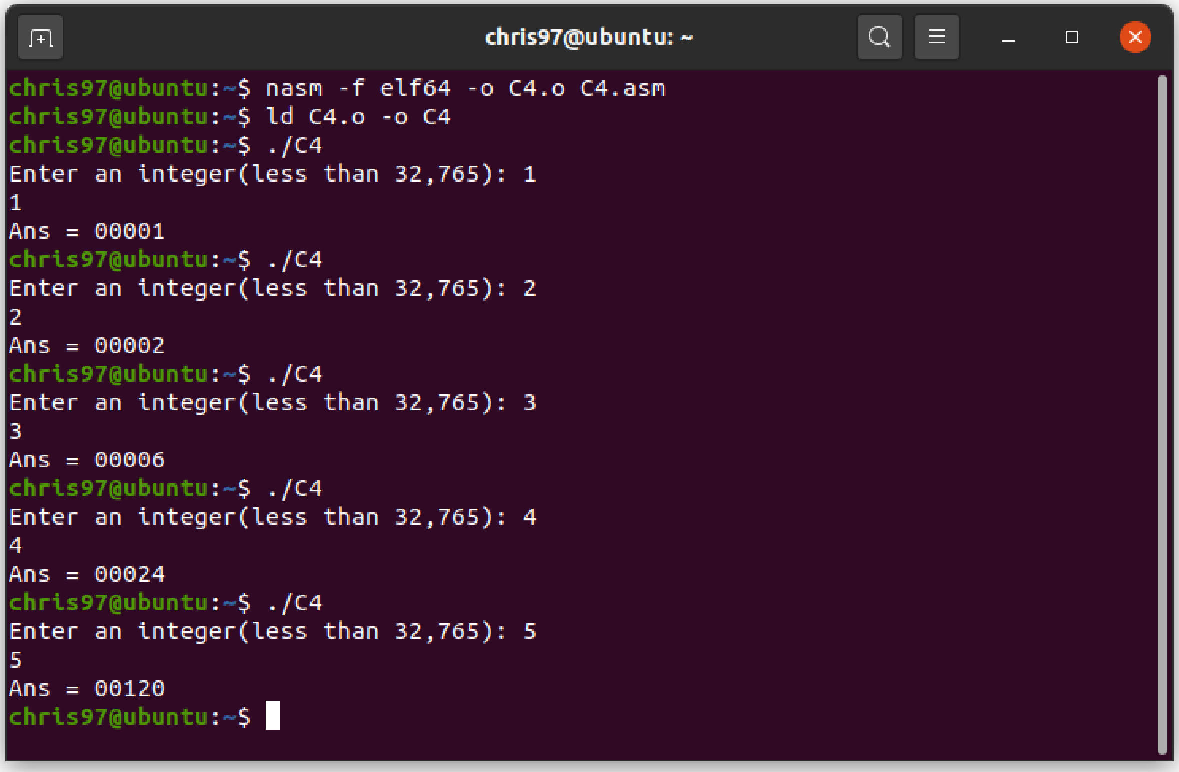
```
;PrintString     PROC
PrintString:
            push    ax              ;Save registers;
            push    dx
; subpgm:
            ; prompt user
            mov eax, 4                      ;Linux print device register conventions
            mov ebx, 1                      ; print default output device
            mov ecx, userMsg                ; pointer to string
            mov edx, lenUserMsg             ; arg1, where to write, screen
            int         80h                 ; interrupt 80 hex, call kernel
            pop     dx              ;Restore registers.
            pop     ax
            ret
;PrintString     ENDP

;GetAnInteger    PROC

GetAnInteger:     ;Get an integer as a string
            ;get response
            mov eax,3           ;read
            mov ebx,2           ;device
            mov ecx,num         ;buffer address
            mov edx,6           ;max characters
            int 0x80

            ;print number    ;works
            mov edx,eax         ; eax contains the number of character read including <lf>
            mov eax, 4
            mov ebx, 1
            mov ecx, num
```

```
        int 80h

ConvertStringToInteger:
        mov ax,0            ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num         ;pt - 1st or next digit of number as a string
                                    ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx              ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1           ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA          ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                     ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

--------------------C4 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
C4	<ProgramName>		0	CS
N	<var>	?	0	DS
fact	<var>	?	2	DS
lit1	<integer>	1	4	DS
T1	<temp>	?	6	DS
T2	<temp>	?	8	DS
T3	<temp>	?	10	DS
T4	<temp>	?	12	DS
T5	<temp>	?	14	DS
T6	<temp>	?	16	DS
T7	<temp>	?	18	DS
T8	<temp>	?	20	DS

--------------------C4 Screenshot--------------------



```
chris97@ubuntu:~$ nasm -f elf64 -o C4.o C4.asm
chris97@ubuntu:~$ ld C4.o -o C4
chris97@ubuntu:~$ ./C4
Enter an integer(less than 32,765): 1
1
Ans = 00001
chris97@ubuntu:~$ ./C4
Enter an integer(less than 32,765): 2
2
Ans = 00002
chris97@ubuntu:~$ ./C4
Enter an integer(less than 32,765): 3
3
Ans = 00006
chris97@ubuntu:~$ ./C4
Enter an integer(less than 32,765): 4
4
Ans = 00024
chris97@ubuntu:~$ ./C4
Enter an integer(less than 32,765): 5
5
Ans = 00120
chris97@ubuntu:~$ 
```

---“B” Options---------------------------------

---------------------B5 Java 0---------------------

```
CLASS B5
{
    /* 5. Compute N! for M iterations using a nested while loop. */

    VAR M, fact, knt, N;

    fact = 1;
    knt = 0;
    READ M;

    WHILE knt < M DO
    {
              READ N;

        WHILE N > 1 DO
              {
            fact = fact * N;
            N = N - 1;
        }

              WRITE fact;
        fact = 1;
        knt = knt + 1;
    }
}
```

---------------------B5 Assembly---------------------

```
sys_exit equu       1
sys_read equ        3
sys_write           equ     4
stdin               equ     0 ; default keyboard
```

```
stdout          equ     1 ; default terminal screen

stderr          equ     3


section .data           ;used to declare constants

        userMsg         db  'Enter an integer(less than 32,765): '

        lenUserMsg      equu    $-userMsg

        displayMsg      db      'You entered: '

        lenDisplayMsg   equ     $-displayMsg

        newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>


        Ten     DW     10  ;Used converting to base ten.

        Result      db     'Ans = '

        ResultValue             db      'aaaaa'

                            db  0xA             ;return

        ResultEnd     equ       $-Result   ; $=> here - address Result = length to print

        num                     times 6   db 'ABCDEF' ;cheat NASM

        numEnd          equ     $-num


section   .bss      ;used to declare uninitialized variables

        TempChar      RESB   1 ;1 byte temp space for use by GetNextChar

        testchar      RESB   1

        ;Temporary storage GetAnInteger.

        ReadInt       RESW   1 ;4 bytes

        ;Used in converting to base ten.

        tempint       RESW        1

        negflag       RESB   1       ;P=positive, N=negative

        M       RESW 1

        fact    RESW 1

        knt     RESW 1

        N       RESW 1

        T1      RESW 1

        T2      RESW 1
        T3      RESW 1
        T4      RESW 1
        T5      RESW 1
        T6      RESW 1
        T7      RESW 1
        T8      RESW 1

section .txt        ;Start of the main program------------------------------------------.

  global _start
_start: mov word [fact], 1
        mov word [knt], 0
        call PrintString
        call GetAnInteger
        mov ax, [ReadInt]
        mov [M], ax

W1:     nop
        mov ax, [knt]
        cmp ax, [M]
        jge L1
        call PrintString
        call GetAnInteger
        mov ax, [ReadInt]
        mov [N], ax

W2:     nop
        mov ax, [N]
        cmp ax, 1
        jle L2
        mov ax, [fact]
        mul word [N]
        mov [fact], ax

        mov ax, [N]
        sub ax, 1
        mov [N], ax

        jmp W2
L2:     nop
        mov ax, [fact]
        call ConvertIntegerToString

        mov eax, 4
        mov ebx, 1
        mov ecx, Result
        mov edx, ResultEnd
        int 80h

        mov word [fact], 1
        mov ax, [knt]
        add ax, 1
        mov [knt], ax

        jmp W1
L1:     nop
fini:
        mov eax, sys_exit
        xor ebx, ebx
        int 80h

;PrintString     PROC
PrintString:
        push    ax          ;Save registers;
```

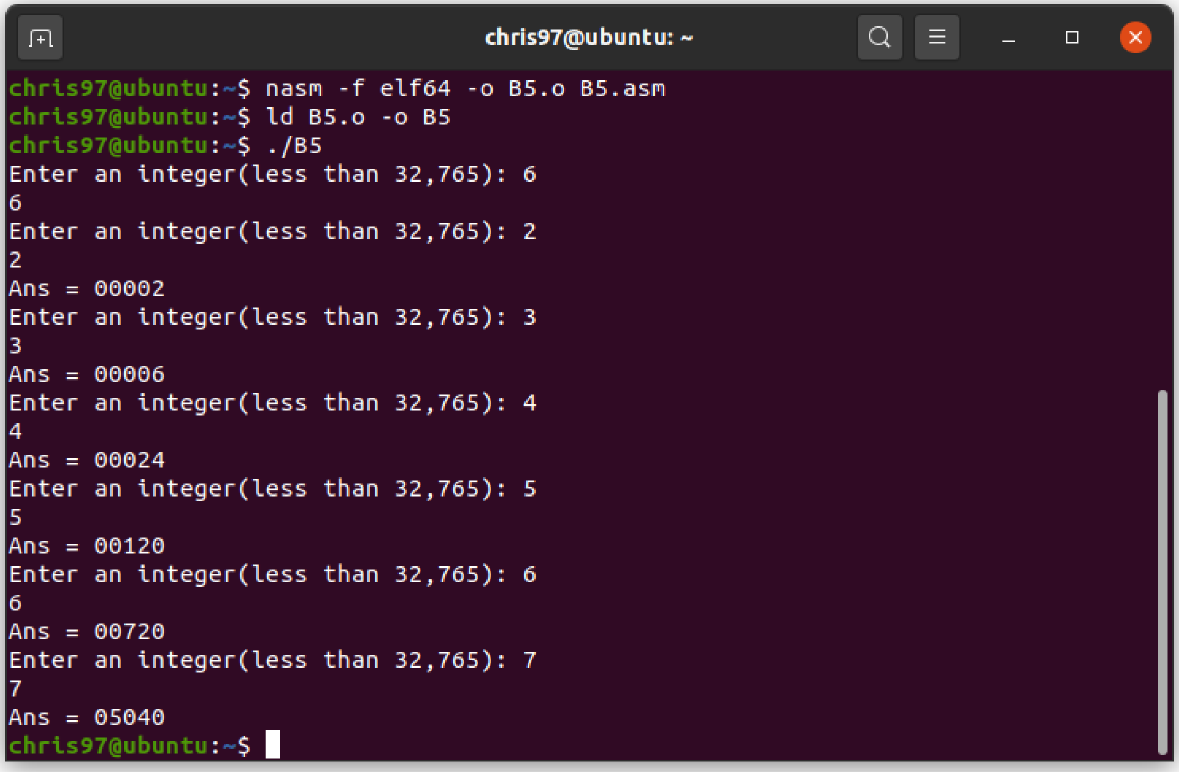
```
            push    dx
; subpgm:
            ; prompt user
            mov eax, 4              ;Linux print device register conventions
            mov ebx, 1              ; print default output device
            mov ecx, userMsg        ; pointer to string
            mov edx, lenUserMsg     ; arg1, where to write, screen
            int     80h             ; interrupt 80 hex, call kernel
            pop     dx      ;Restore registers.
            pop     ax
            ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:   ;Get an integer as a string
            ;get response
            mov eax,3       ;read
            mov ebx,2       ;device
            mov ecx,num     ;buffer address
            mov edx,6       ;max characters
            int 0x80

            ;print number   ;works
            mov edx,eax     ; eax contains the number of character read including <lf>
            mov eax, 4
            mov ebx, 1
            mov ecx, num
            int 80h

ConvertStringToInteger:
            mov ax,0        ;hold integer
```

```
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num      ;pt - 1st or next digit of number as a string
                                ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx           ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1        ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA       ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                   ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

--------------------B5 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
B5	<ProgramName>		0	CS
M	<var>	?	0	DS
fact	<var>	?	2	DS
knt	<var>	?	4	DS
N	<var>	?	6	DS
lit1	<integer>	1	8	DS
lit0	<integer>	0	10	DS
T1	<temp>	?	12	DS
T2	<temp>	?	14	DS
T3	<temp>	?	16	DS
T4	<temp>	?	18	DS
T5	<temp>	?	20	DS
T6	<temp>	?	22	DS
T7	<temp>	?	24	DS
T8	<temp>	?	26	DS

--------------------B5 Screenshot--------------------



```
chris97@ubuntu:~$ nasm -f elf64 -o B5.o B5.asm
chris97@ubuntu:~$ ld B5.o -o B5
chris97@ubuntu:~$ ./B5
Enter an integer(less than 32,765): 6
6
Enter an integer(less than 32,765): 2
2
Ans = 00002
Enter an integer(less than 32,765): 3
3
Ans = 00006
Enter an integer(less than 32,765): 4
4
Ans = 00024
Enter an integer(less than 32,765): 5
5
Ans = 00120
Enter an integer(less than 32,765): 6
6
Ans = 00720
Enter an integer(less than 32,765): 7
7
Ans = 05040
chris97@ubuntu:~$ 
```

---------------------B6 Java 0---------------------

```
CLASS B6
{
   /* Calculate N! using a function. */

   VAR N, fact;

   READ N;
   fact = 1;

   PROCEDURE factorial()
   {
      WHILE N > 1 DO
           {
         fact = fact * N;
         N = N - 1;
           }
   }

   CALL factorial();
   WRITE fact;
}
```

---------------------B6 Assembly---------------------

```
sys_exit equ        1
sys_readequ         3
sys_write           equ      4
stdin               equ      0 ; default keyboard
stdout              equ      1 ; default terminal screen
stderr              equ      3

section .data                ;used to declare constants
          userMsg            db  'Enter an integer(less than 32,765): '
          lenUserMsg         equ       $-userMsg
```

```
        displayMsg      db      'You entered: '
        lenDisplayMsg   equ     $-displayMsg
        newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>

        Ten     DW      10  ;Used converting to base ten.
        Result      db      'Ans = '
        ResultValue             db      'aaaaa'
                                    db  0xA             ;return
        ResultEnd     equ       $-Result   ; $=> here - address Result = length to print
        num                     times 6    db 'ABCDEF' ;cheat NASM
        numEnd          equu    $-num

section .bss        ;used to declare uninitialized variables
        TempChar        RESB    1 ;1 byte temp space for use by GetNextChar
        testchar        RESB    1
        ;Temporary storage GetAnInteger.
        ReadInt         RESW    1 ;4 bytes
        ;Used in converting to base ten.
        tempint         RESW        1
        negflag         RESB    1       ;P=positive, N=negative
        N       RESW 1
        fact    RESW 1
        T1      RESW 1
        T2      RESW 1
        T3      RESW 1
        T4      RESW 1
        T5      RESW 1
        T6      RESW 1
        T7      RESW 1
        T8      RESW 1

section .txt        ;Start of the main program-----------------------------------------.
```

```
   global _start
_start:    call PrintString
           call GetAnInteger
           mov ax, [ReadInt]
           mov [N], ax

           mov word [fact], 1
           call factorial
           mov ax, [fact]
           call ConvertIntegerToString

           mov eax, 4
           mov ebx, 1
           mov ecx, Result
           mov edx, ResultEnd
           int 80h

fini:
           mov eax, sys_exit
           xor ebx, ebx
           int 80h

factorial:          nop
W1:        nop
           mov ax, [N]
           cmp ax, 1
           jle L1
           mov ax, [fact]
           mul word [N]
           mov [fact], ax
```

```
            mov ax, [N]
            sub ax, 1
            mov [N], ax

            jmp W1
L1:         nop
            ret
;PrintString    PROC
PrintString:
            push    ax              ;Save registers;
            push    dx
; subpgm:
            ; prompt user
            mov eax, 4                      ;Linux print device register conventions
            mov ebx, 1                      ; print default output device
            mov ecx, userMsg                ; pointer to string
            mov edx, lenUserMsg             ; arg1, where to write, screen
            int         80h                 ; interrupt 80 hex, call kernel
            pop     dx              ;Restore registers.
            pop     ax
            ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:   ;Get an integer as a string
            ;get response
            mov eax,3           ;read
            mov ebx,2           ;device
            mov ecx,num         ;buffer address
            mov edx,6           ;max characters
            int 0x80
```

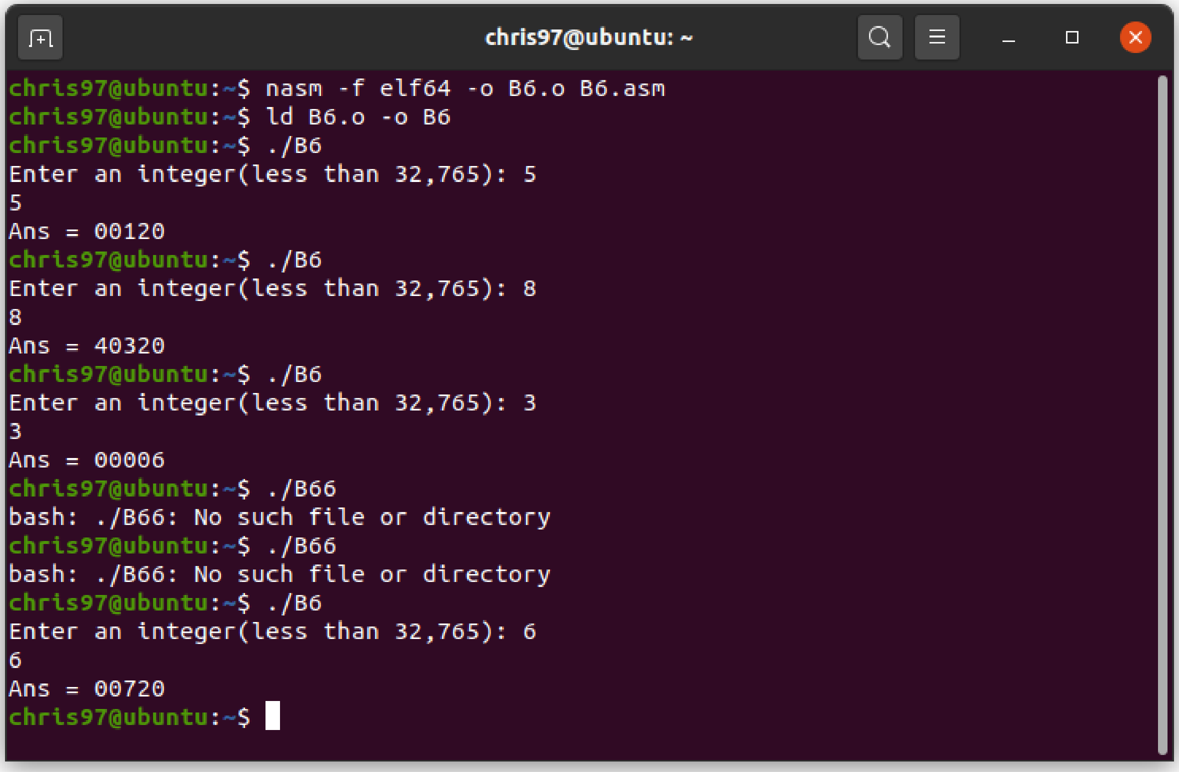
```
        ;print number   ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                                ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC
```

```
ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                   ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

--------------------B6 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
B6	<ProgramName>		0	CS
N	<var>	?	0	DS
fact	<var>	?	2	DS
lit1	<integer>	1	4	DS
factorial	<PROCEDURE>		2	CS
T1	<temp>	?	6	DS
T2	<temp>	?	8	DS
T3	<temp>	?	10	DS
T4	<temp>	?	12	DS
T5	<temp>	?	14	DS
T6	<temp>	?	16	DS
T7	<temp>	?	18	DS
T8	<temp>	?	20	DS

--------------------B6 Screenshot--------------------



```
chris97@ubuntu:~$ nasm -f elf64 -o B6.o B6.asm
chris97@ubuntu:~$ ld B6.o -o B6
chris97@ubuntu:~$ ./B6
Enter an integer(less than 32,765): 5
5
Ans = 00120
chris97@ubuntu:~$ ./B6
Enter an integer(less than 32,765): 8
8
Ans = 40320
chris97@ubuntu:~$ ./B6
Enter an integer(less than 32,765): 3
3
Ans = 00006
chris97@ubuntu:~$ ./B66
bash: ./B66: No such file or directory
chris97@ubuntu:~$ ./B66
bash: ./B66: No such file or directory
chris97@ubuntu:~$ ./B6
Enter an integer(less than 32,765): 6
6
Ans = 00720
chris97@ubuntu:~$ 
```

---"A" Options-------------------------------------

---------------------A7 Java 0---------------------

```
CLASS A7

{

  /* Calculate N! using a recursive function. */

  VAR N, fact;


  READ N;

  fact = 1;


  PROCEDURE RecursiveFactorial()

  {

    IF N != 1 THEN

        {

          fact = fact * N;

      N = N - 1;


      CALL RecursiveFactorial();

    }

  }


  CALL RecursiveFactorial();

  WRITE fact;


}
```

---------------------A7 Assembly---------------------

```
sys_exit equ        1
sys_read equ        3
sys_write           equ     4
stdin               equ     0 ; default keyboard
stdout              equ     1 ; default terminal screen
stderr              equ     3

section .data               ;used to declare constants
        userMsg             db  'Enter an integer(less than 32,765): '
        lenUserMsg          equ     $-userMsg
```

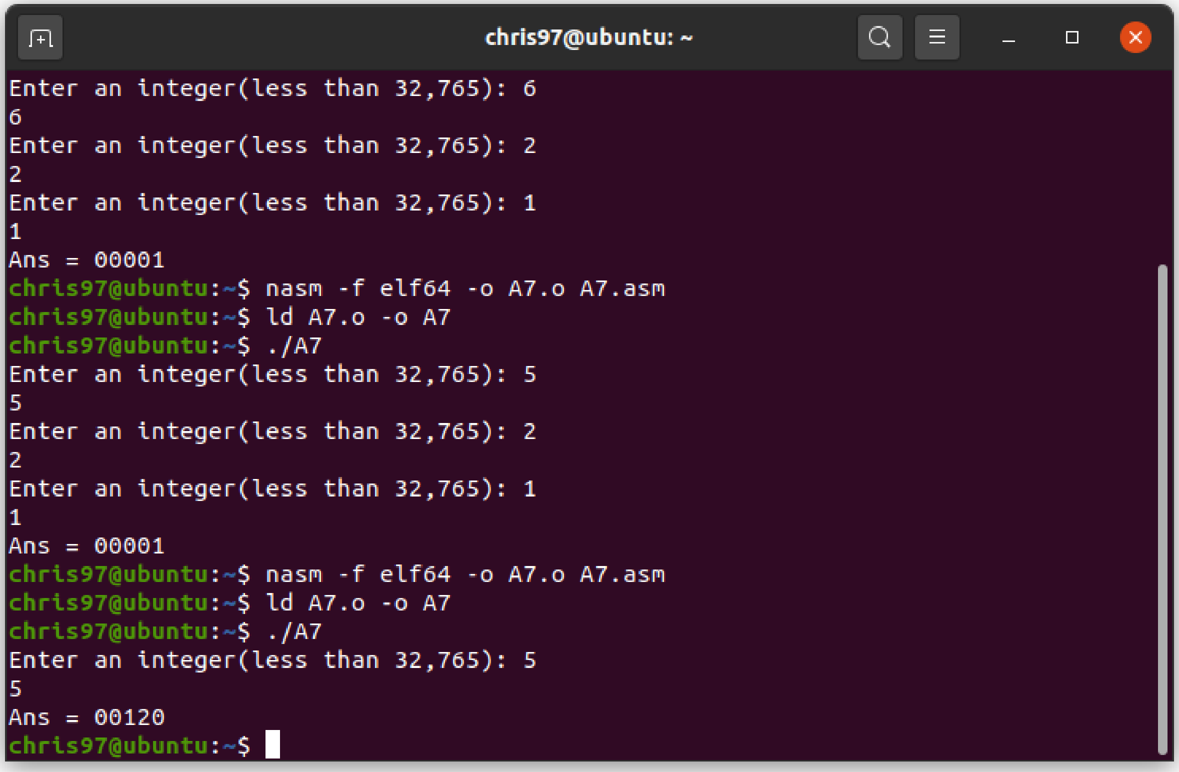
```
            displayMsg      db      'You entered: '
            lenDisplayMsg   equ     $-displayMsg
            newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>

            Ten     DW      10  ;Used converting to base ten.
            Result      db      'Ans = '
            ResultValue             db      'aaaaa'
                                db  0xA                 ;return
            ResultEnd     equu          $-Result   ; $=> here - address Result = length to print
            num                         times 6    db 'ABCDEF' ;cheat NASM
            numEnd          equ         $-num

section     .bss        ;used to declare uninitialized variables
            TempChar        RESB    1 ;1 byte temp space for use by GetNextChar
            testchar        RESB    1
            ;Temporary storage GetAnInteger.
            ReadInt         RESW    1 ;4 bytes
            ;Used in converting to base ten.
            tempint         RESW        1
            negflag         RESB    1       ;P=positive, N=negative
            N           RESW 1
            fact        RESW 1
            T1          RESW 1
            T2          RESW 1
            T3          RESW 1
            T4          RESW 1
            T5          RESW 1
            T6          RESW 1
            T7          RESW 1
            T8          RESW 1

section .txt            ;Start of the main program-----------------------------------------.

   global _start
_start:     call PrintString
            call GetAnInteger
            mov ax, [ReadInt]
            mov [N], ax

            mov word [fact], 1
            call RecursiveFactorial
            mov ax, [fact]
            call ConvertIntegerToString

            mov eax, 4
            mov ebx, 1
            mov ecx, Result
            mov edx, ResultEnd
            int 80h

fini:
            mov eax, sys_exit
            xor ebx, ebx
            int 80h
```

```
RecursiveFactorial:         nop
            mov ax, [N]
            cmp ax, 1
            je L1

            mov ax, [fact]
            mul word [N]
            mov [fact], ax

            mov ax, [N]
            sub ax, 1
            mov [N], ax

            call RecursiveFactorial
L1:         nop
            ret
;PrintString    PROC
PrintString:
            push    ax              ;Save registers;
            push    dx
; subpgm:
            ; prompt user
            mov eax, 4                      ;Linux print device register conventions
            mov ebx, 1                      ; print default output device
            mov ecx, userMsg                ; pointer to string
            mov edx, lenUserMsg             ; arg1, where to write, screen
            int         80h                 ; interrupt 80 hex, call kernel
            pop     dx              ;Restore registers.
            pop     ax
            ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:     ;Get an integer as a string
            ;get response
            mov eax,3           ;read
            mov ebx,2           ;device
            mov ecx,num         ;buffer address
            mov edx,6           ;max characters
            int 0x80

            ;print number   ;works
            mov edx,eax         ; eax contains the number of character read including <lf>
            mov eax, 4
            mov ebx, 1
            mov ecx, num
            int 80h

ConvertStringToInteger:
            mov ax,0            ;hold integer
            mov [ReadInt],ax ;initialize 16 bit number to zero
            mov ecx,num         ;pt - 1st or next digit of number as a string
```

```
                                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx              ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1           ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA          ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                   ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

---------------------A7 Symbol Table---------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
A7	<ProgramName>		0	CS
N	<var>	?	0	DS
fact	<var>	?	2	DS
lit1	<integer>	1	4	DS
RecursiveFactorial	<PROCEDURE>		2	CS
T1	<temp>	?	6	DS
T2	<temp>	?	8	DS
T3	<temp>	?	10	DS
T4	<temp>	?	12	DS
T5	<temp>	?	14	DS
T6	<temp>	?	16	DS
T7	<temp>	?	18	DS
T8	<temp>	?	20	DS



```
Enter an integer(less than 32,765): 6
6
Enter an integer(less than 32,765): 2
2
Enter an integer(less than 32,765): 1
1
Ans = 00001
chris97@ubuntu:~$ nasm -f elf64 -o A7.o A7.asm
chris97@ubuntu:~$ ld A7.o -o A7
chris97@ubuntu:~$ ./A7
Enter an integer(less than 32,765): 5
5
Enter an integer(less than 32,765): 2
2
Enter an integer(less than 32,765): 1
1
Ans = 00001
chris97@ubuntu:~$ nasm -f elf64 -o A7.o A7.asm
chris97@ubuntu:~$ ld A7.o -o A7
chris97@ubuntu:~$ ./A7
Enter an integer(less than 32,765): 5
5
Ans = 00120
chris97@ubuntu:~$ 
```

---------------------A7 Screenshot---------------------

---------------------A8 Java 0---------------------

```
CLASS LCD{
   CONST M = 7, N = 85;
   VAR X, Y, Z, Q, R;

   PROCEDURE Multiply(){
       VAR A, B;

       A = X;  B = Y;  Z = 0;
       WHILE B > 0 DO {
            IF ODD B THEN Z = Z + A;
            A = 2 * A;  B = B / 2;
       }
   }

   PROCEDURE Divide(){
       VAR W;

       R = X;  Q = 0;  W = Y;
       WHILE W <= R DO W = 2 * W;
       WHILE W > Y DO {
            Q = 2 * Q;  W = W / 2;
            IF W <= R THEN {
                R = R - W;
                Q = Q + 1;
            }
       }
   }

   PROCEDURE GCD(){
       VAR F, G;
```

```
          F = X;  G = Y;

          WHILE F != G DO {

               IF F < G THEN G = G - F;

               IF G < F THEN F = F - G;

          }

          Z = F;

     }


     /*  Main Program.  */

     X = M;  Y = N;  CALL Multiply();

     X = 25;  Y = 3;  CALL Divide();

     X = 84;  Y = 36;  CALL GCD();

     WRITE Z;

}
```

---------------------A8 Assembly---------------------

```
sys_exit equu       1
sys_read equ        3
sys_write           equ       4
stdin               equ       0 ; default keyboard
stdout              equ       1 ; default terminal screen
stderr              equ       3

section .data                 ;used to declare constants
          userMsg             db  'Enter an integer(less than 32,765): '
          lenUserMsg          equ       $-userMsg
          displayMsg          db        'You entered: '
          lenDisplayMsg       equ       $-displayMsg
          newline             db        0xA       ; 0xA 0xD is ASCII <LF><CR>

          Ten       DW      10  ;Used converting to base ten.
          Result         db      'Ans = '
          ResultValue              db         'aaaaa'
                                   db  0xA                  ;return
          ResultEnd      equ            $-Result   ; $=> here - address Result = length to print
          num                           times 6    db 'ABCDEF' ;cheat NASM
          numEnd              equ       $-num
          M         DW7
          N         DW85
          LIT0      DW 0
          LIT2      DW 2
          LIT1      DW 1
          LIT25     DW 25
          LIT3      DW 3
          LIT84     DW 84
```

```
         LIT36    DW 36

section  .bss       ;used to declare uninitialized variables
         TempChar       RESB    1 ;1 byte temp space for use by GetNextChar
         testchar       RESB    1
         ;Temporary storage GetAnInteger.
         ReadInt        RESW    1 ;4 bytes
         ;Used in converting to base ten.
         tempint        RESW          1
         negflag        RESB    1        ;P=positive, N=negative
         X        RESW 1
         Y        RESW 1
         Z        RESW 1
         Q        RESW 1
         R        RESW 1
         A        RESW 1
         B        RESW 1
         W        RESW 1
         F        RESW 1
         G        RESW 1
         T1       RESW 1
         T2       RESW 1
         T3       RESW 1
         T4       RESW 1
         T5       RESW 1
         T6       RESW 1
         T7       RESW 1
         T8       RESW 1

section .txt          ;Start of the main program-----------------------------------------.

  global _start
_start:  mov ax, [M]
         mov [X], ax

         mov ax, [N]
         mov [Y], ax

         call Multiply
         mov word [X], 25
         mov word [Y], 3
         call Divide
         mov word [X], 84
         mov word [Y], 36
         call GCD
         mov ax, [Z]
         call ConvertIntegerToString

         mov eax, 4
         mov ebx, 1
         mov ecx, Result
         mov edx, ResultEnd
         int 80h
```

```
fini:
            mov eax, sys_exit
            xor ebx, ebx
            int 80h

Multiply:        nop
            mov ax, [X]
            mov [A], ax

            mov ax, [Y]
            mov [B], ax

            mov word [Z], 0
W1:         nop
            mov ax, [B]
            cmp ax, 0
            jle L1
            mov ax, [B]
            test al, 1
            jz L2

            mov ax, [Z]
            add ax, [A]
            mov [Z], ax

L2:         nop
            mov ax, 2
            mul word [A]
            mov [A], ax

            mov dx, 0
            mov ax, [B]
            mov bx, 2
            div bx
            mov [B], ax

            jmp W1
L1:         nop
            ret
Divide:     nop
            mov ax, [X]
            mov [R], ax

            mov word [Q], 0
            mov ax, [Y]
            mov [W], ax

W2:         nop
            mov ax, [W]
            cmp ax, [R]
            jg L3
            mov ax, 2
            mul word [W]
            mov [W], ax
```

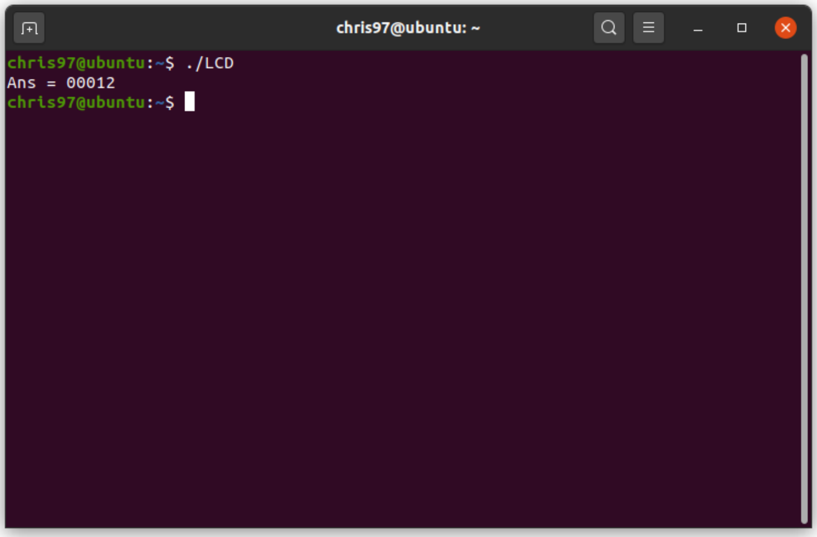
```
        jmp W2
L3:     nop
W3:     nop
        mov ax, [W]
        cmp ax, [Y]
        jle L4
        mov ax, 2
        mul word [Q]
        mov [Q], ax

        mov dx, 0
        mov ax, [W]
        mov bx, 2
        div bx
        mov [W], ax

        mov ax, [W]
        cmp ax, [R]
        jg L5

        mov ax, [R]
        sub ax, [W]
        mov [R], ax

        mov ax, [Q]
        add ax, 1
        mov [Q], ax

L5:     nop
        jmp W3
L4:     nop
        ret
GCD:    nop
        mov ax, [X]
        mov [F], ax

        mov ax, [Y]
        mov [G], ax

W4:     nop
        mov ax, [F]
        cmp ax, [G]
        je L6
        mov ax, [F]
        cmp ax, [G]
        jge L7

        mov ax, [G]
        sub ax, [F]
        mov [G], ax

L7:     nop
        mov ax, [G]
        cmp ax, [F]
        jge L8

        mov ax, [F]
        sub ax, [G]
        mov [F], ax

L8:     nop
        jmp W4
L6:     nop
        mov ax, [F]
        mov [Z], ax

        ret
;PrintString   PROC
PrintString:
        push   ax          ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4                  ;Linux print device register conventions
        mov ebx, 1                  ; print default output device
        mov ecx, userMsg            ; pointer to string
        mov edx, lenUserMsg         ; arg1, where to write, screen
        int        80h              ; interrupt 80 hex, call kernel
        pop    dx          ;Restore registers.
        pop    ax
        ret
;PrintString   ENDP

;GetAnInteger   PROC

GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3          ;read
        mov ebx,2          ;device
        mov ecx,num        ;buffer address
        mov edx,6          ;max characters
        int 0x80

        ;print number   ;works
        mov edx,eax        ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0           ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num        ;pt - 1st or next digit of number as a string
                                    ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
```

```
Next:   sub bl,'0';convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                                   ; digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

--------------------A8 Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
LCD	<ProgramName>		0	CS
M	$CONST	7	0	DS
N	$CONST	85	0	DS
X	<var>	?	0	DS
Y	<var>	?	2	DS
Z	<var>	?	4	DS
Q	<var>	?	6	DS
R	<var>	?	8	DS
Multiply	<PROCEDURE>		6	CS
A	<var>	?	10	DS
B	<var>	?	12	DS
lit0	<integer>	0	14	DS
lit2	<integer>	2	16	DS
Divide	<PROCEDURE>		8	CS
W	<var>	?	18	DS
lit1	<integer>	1	20	DS
GCD	<PROCEDURE>		10	CS
F	<var>	?	22	DS
G	<var>	?	24	DS
lit25	<integer>	25	26	DS
lit3	<integer>	3	28	DS
lit84	<integer>	84	30	DS
lit36	<integer>	36	32	DS
T1	<temp>	?	34	DS
T2	<temp>	?	36	DS
T3	<temp>	?	38	DS
T4	<temp>	?	40	DS
T5	<temp>	?	42	DS
T6	<temp>	?	44	DS
T7	<temp>	?	46	DS
T8	<temp>	?	48	DS

--------------------A8 Screenshot--------------------

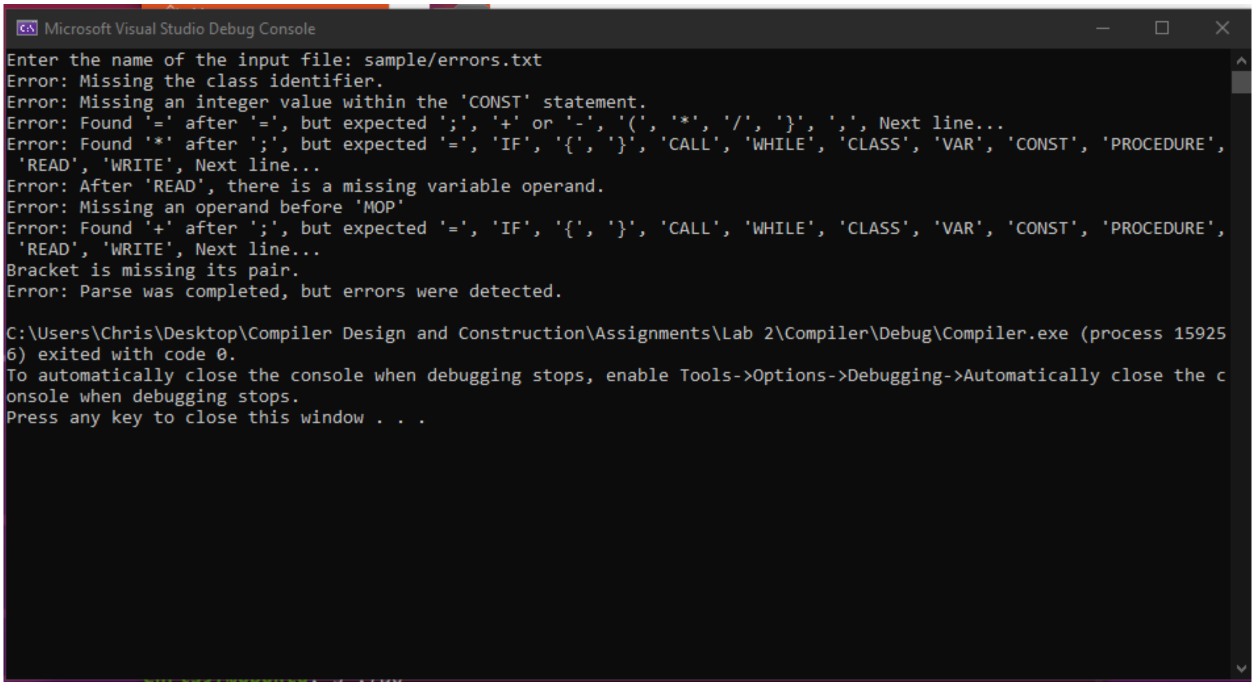


```
chris97@ubuntu:~$ ./LCD
Ans = 00012
chris97@ubuntu:~$ 
```

--------------------A8(1) Java 0--------------------

```
CLASS  {
    /* No class identifier */
    /* Left off value for B */
    CONST A = 3, B = ;
    /* Missing semicolon after C*/
    VAR C, D, E;
    D = C
    E = 5;
    D *= E;
    /* No variable provided for input */
    READ ;
    C = 5 + D;
    /* Missing operand */
    E =  * 6;
    /* Started statement with addop */
    A + B;
    /* Missing closing bracket */
```

--------------------A8(1) Screenshot--------------------



```
Enter the name of the input file: sample/errors.txt
Error: Missing the class identifier.
Error: Missing an integer value within the 'CONST' statement.
Error: Found '=' after '=', but expected ';', '+' or '-', '(', '*', '/', '}', ',', Next line...
Error: Found '*' after ';', but expected '=', 'IF', '{', '}', 'CALL', 'WHILE', 'CLASS', 'VAR', 'CONST', 'PROCEDURE',
 'READ', 'WRITE', Next line...
Error: After 'READ', there is a missing variable operand.
Error: Missing an operand before 'MOP'
Error: Found '+' after ';', but expected '=', 'IF', '{', '}', 'CALL', 'WHILE', 'CLASS', 'VAR', 'CONST', 'PROCEDURE',
 'READ', 'WRITE', Next line...
Bracket is missing its pair.
Error: Parse was completed, but errors were detected.

C:\Users\Chris\Desktop\Compiler Design and Construction\Assignments\Lab 2\Compiler\Debug\Compiler.exe (process 15925
6) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the c
onsole when debugging stops.
Press any key to close this window . . .
```

---------------------A8(2) Java 0---------------------

```
CLASS UndeclaredVar

{

   VAR A, B, C;

   A = 1;

   B = 2;



   /* Attempt to use undeclared var D */

   D = A + B;

}
```

---------------------A8(2) Assembly---------------------

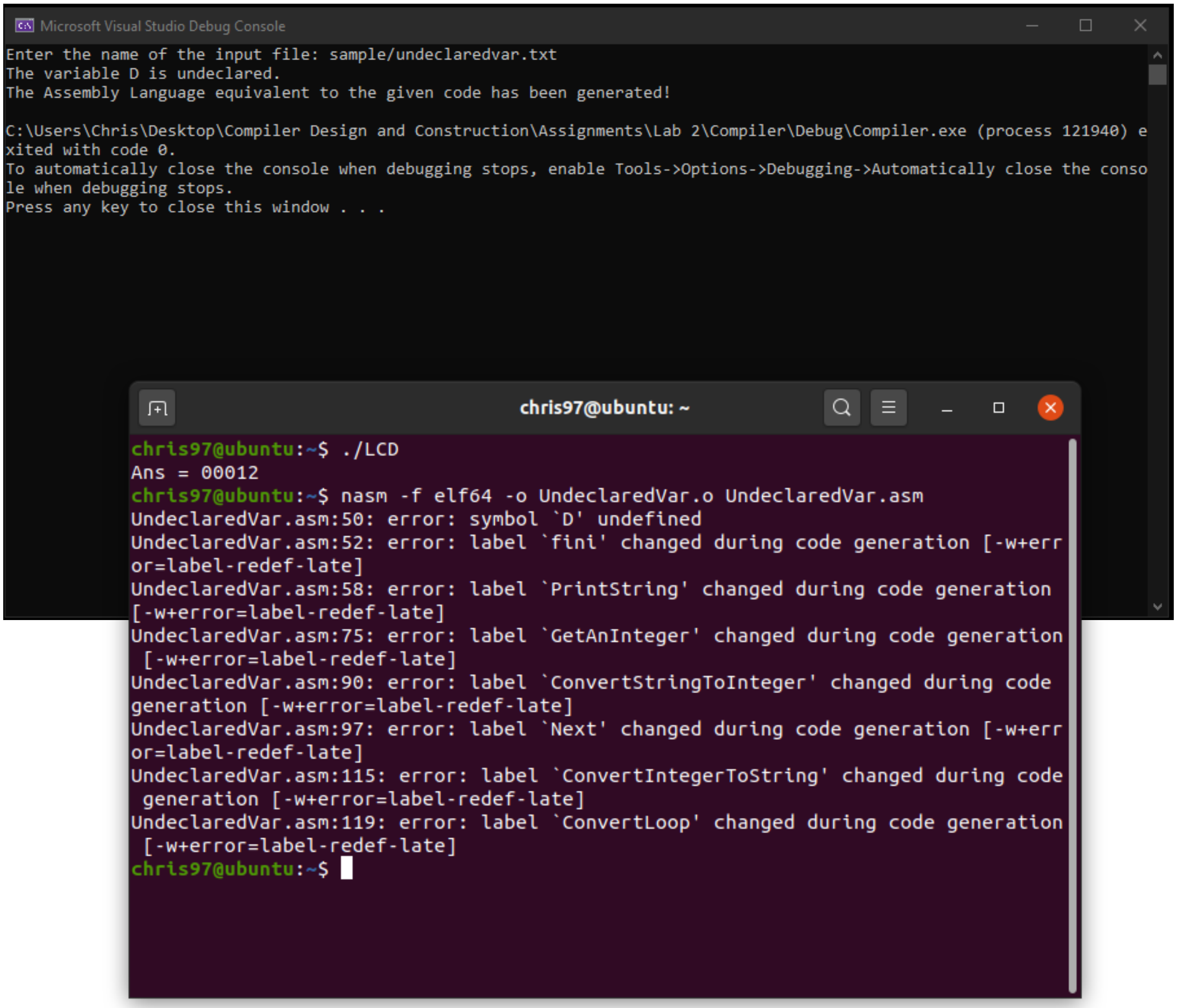
```
sys_exit equu        1
sys_read equ         3
sys_write            equ       4
stdin                equ       0 ; default keyboard
stdout               equ       1 ; default terminal screen
stderr               equ       3

section .data                  ;used to declare constants
          userMsg              db  'Enter an integer(less than 32,765): '
          lenUserMsg           equ       $-userMsg
          displayMsg           db        'You entered: '
          lenDisplayMsg        equ       $-displayMsg
          newline              db        0xA       ; 0xA 0xD is ASCII <LF><CR>

          Ten       DW      10  ;Used converting to base ten.
          Result         db      'Ans = '
          ResultValue                   db        'aaaaa'
                                    db  0xA                  ;return
          ResultEnd       equ           $-Result   ; $=> here - address Result = length to print
          num                           times 6    db 'ABCDEF' ;cheat NASM
          numEnd               equ       $-num

section   .bss        ;used to declare uninitialized variables
          TempChar        RESB    1 ;1 byte temp space for use by GetNextChar
          testchar        RESB    1
          ;Temporary storage GetAnInteger.
          ReadInt         RESW    1 ;4 bytes
          ;Used in converting to base ten.
          tempint         RESW          1
          negflag         RESB    1        ;P=positive, N=negative
          A         RESW 1
          B         RESW 1
          C         RESW 1
          T1        RESW 1
          T2        RESW 1
          T3        RESW 1
```

```
            T4      RESW 1
            T5      RESW 1
            T6      RESW 1
            T7      RESW 1
            T8      RESW 1

section .txt        ;Start of the main program------------------------------------------.

  global _start
_start:     mov word [A], 1
            mov word [B], 2
            mov ax, [A]
            add ax, [B]
            mov [D], ax

fini:
            mov eax, sys_exit
            xor ebx, ebx
            int 80h

;PrintString    PROC
PrintString:
            push   ax           ;Save registers;
            push   dx
; subpgm:
            ; prompt user
            mov eax, 4                  ;Linux print device register conventions
            mov ebx, 1                  ; print default output device
            mov ecx, userMsg            ; pointer to string
            mov edx, lenUserMsg         ; arg1, where to write, screen
            int     80h                 ; interrupt 80 hex, call kernel
            pop    dx           ;Restore registers.
            pop    ax
            ret
;PrintString    ENDP

;GetAnInteger   PROC

GetAnInteger:   ;Get an integer as a string
            ;get response
            mov eax,3       ;read
            mov ebx,2       ;device
            mov ecx,num     ;buffer address
            mov edx,6       ;max characters
            int 0x80

            ;print number   ;works
            mov edx,eax     ; eax contains the number of character read including <lf>
            mov eax, 4
            mov ebx, 1
            mov ecx, num
            int 80h

ConvertStringToInteger:
            mov ax,0        ;hold integer
            mov [ReadInt],ax ;initialize 16 bit number to zero
            mov ecx,num     ;pt - 1st or next digit of number as a string
                                    ;terminated by <lf>.
            mov bx,0
            mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0';convert character to number
            mov ax,[ReadInt]
            mov dx,10
            mul dx          ;eax = eax * 10
            add ax,bx
            mov [ReadInt], ax

            mov bx,0
            add ecx,1       ;pt = pt + 1
            mov bl, byte[ecx]

            cmp bl,0xA      ;is it a <lf>
            jne Next ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&7
            ret
;       ENDP GetAnInteger

;ConvertIntegerToString PROC

ConvertIntegerToString:
            mov ebx, ResultValue + 4   ;Store the integer as a five
                                       ; digit char string at Result for printing

ConvertLoop:
            sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
            mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
            div cx
            add dl,'0' ; Add '0' to dl to convert from binary to character.
            mov [ebx], dl
            dec ebx
            cmp ebx,ResultValue
            jge ConvertLoop

            ret

;ConvertIntegerToString  ENDP
```

--------------------A8(2) Symbol Table--------------------

TOKEN	CLASS	VALUE	ADDRESS	SEGMENT
UndeclaredVar	<ProgramName>		0	CS
A	<var>	?	0	DS
B	<var>	?	2	DS
C	<var>	?	4	DS
lit1	<integer>	1	6	DS
lit2	<integer>	2	8	DS
T1	<temp>	?	10	DS
T2	<temp>	?	12	DS
T3	<temp>	?	14	DS
T4	<temp>	?	16	DS
T5	<temp>	?	18	DS
T6	<temp>	?	20	DS
T7	<temp>	?	22	DS
T8	<temp>	?	24	DS

--------------------A8(2) Screenshots--------------------



```
Enter the name of the input file: sample/undeclaredvar.txt
The variable D is undeclared.
The Assembly Language equivalent to the given code has been generated!

C:\Users\Chris\Desktop\Compiler Design and Construction\Assignments\Lab 2\Compiler\Debug\Compiler.exe (process 121940) e
xited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
chris97@ubuntu:~$ ./LCD
Ans = 00012
chris97@ubuntu:~$ nasm -f elf64 -o UndeclaredVar.o UndeclaredVar.asm
UndeclaredVar.asm:50: error: symbol `D' undefined
UndeclaredVar.asm:52: error: label `fini' changed during code generation [-w+err
or=label-redef-late]
UndeclaredVar.asm:58: error: label `PrintString' changed during code generation
[-w+error=label-redef-late]
UndeclaredVar.asm:75: error: label `GetAnInteger' changed during code generation
 [-w+error=label-redef-late]
UndeclaredVar.asm:90: error: label `ConvertStringToInteger' changed during code
generation [-w+error=label-redef-late]
UndeclaredVar.asm:97: error: label `Next' changed during code generation [-w+err
or=label-redef-late]
UndeclaredVar.asm:115: error: label `ConvertIntegerToString' changed during code
 generation [-w+error=label-redef-late]
UndeclaredVar.asm:119: error: label `ConvertLoop' changed during code generation
 [-w+error=label-redef-late]
chris97@ubuntu:~$ 
```

-------------------------------Source Code Header Files-------------------------------

---------------CodeGen.h----------------

```
/********************************************************
Name: CodeGen.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
class that will be utilized by the Code Generation
body file to generate the assembly code.

        DISCLAIMER:
THIS PORTION "CodeGen.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Parser.h", "LL.h", "Quad.h" and
"CodeGen.cpp".
********************************************************/

#ifndef CODEGEN_H
#define CODEGEN_H

/*Include Program header files*/
#include "Parser.h"
#include "LL.h"
#include "Quad.h"

/*Include System header files*/
#include <fstream>
#include <queue>
#include <iostream>
#include <list>

using namespace std;

/*
Class: CodeGen
Parameters: (N/A)
Return: (N/A)
Description: Declares the CodeGen functions
and variables that will be used when
the header file is called.
*/
class CodeGen
{
public:

        /*Function identifier*/
        CodeGen(list <Quad> q, LL* table);

        /*Void function identifier*/
```

```
		void genCode();

private:

		/*File stream variable*/
		ifstream partDS;	//Data Section file.
		ifstream partBS;	//Bss Section file.
		ifstream IOFile;	//IO Routines file.
		ofstream out_stream;		//Output file.

		/*List variable*/
		list <Quad> queue_Quad;

		/*Linked list pointer variable.*/
		LL* SymTable;

		/*Function identifiers*/
		void addDSandBS();
		void addIO();

		/*Integer function identifier*/
		int quadToInt(Quad qu);
};

#endif
```

---------------LL.h----------------

```
/********************************************************
Name: LL.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
classes that will be utilized by the Linked List body
file to create the symbol table used by the program.

		DISCLAIMER:
THIS PORTION "LL.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Token.h", "CodeGen.h", "Scanner.h",
"Parser.h", and "LL.cpp".
********************************************************/

#ifndef LL_H
#define LL_H

/*Include Program header file*/
#include "Token.h"

/*Include System header files*/
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

/*
Class: Node
Parameters: (N/A)
Return: (N/A)
Description: Declares the Node functions
and variables that will be used when
the header file is called.
*/
class Node
{
public:

	/*Pointer variables*/
	Token token;
	Node* next;

	/*
	Constructor: Node
	Parameters: (Token toke, Node* n)
	Description: Gives token the value
```

```
        of 'toke' and next the value of 'n'.
        */
        Node(Token toke, Node* n)
        {
                token = toke;
                next = n;
        }
};

/*
Class: LL
Parameters: (N/A)
Return: (N/A)
Description: Declares the Linked List
functions and variables that will be
used when the header file is called.
*/
class LL
{
public:

        /*Node pointer variables*/
        Node* head;
        Node* last;

        /*
        Constructor: LL
        Parameters: (string outStream)
        Description: Sets the initial values.
        */
        LL(string outStream)
        {
                this->head = NULL;
                this->symStream = outStream;
        }

        /*Boolean function identifier*/
        bool inLL(string TString);

        /*Void function identifiers*/
        void add(Token newToken);
        void printLL();

        /*String function identifier*/
        string getClass(string TString);

private:

        /*String variable*/
        string symStream;

        /*Output filestream variable*/
        ofstream STFile;
};
```

```
#endif
```

---------------Scanner.h----------------

```
/********************************************************
Name: Scanner.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
classes that will be utilized by the Scanner body
file to 'scan' the given program and gather it's
information to generate the token list that will be
used by the program.

        DISCLAIMER:
THIS PORTION "Scanner.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "LL.h", "Parser.h", and "Scanner.cpp".
********************************************************/

#ifndef SCANNER_H
#define SCANNER_H

/*Include Program header file*/
#include "LL.h"

/*Include System header files*/
#include <iostream>
#include <fstream>
#include <string>
#include <cstdio>
#include <cctype>

using namespace std;

/*Global string variable definitions*/
const string CLASS = "$CLASS";
const string VAR = "<$var>";
const string CONST = "$CONST";
const string PROCEDURE = "$PROCEDURE";
const string WHILE = "$WHILE";
const string DO = "$DO";
const string IF = "$IF";
const string THEN = "$THEN";
const string ELSE = "$ELSE";
const string CALL = "$CALL";
const string READ = "$READ";
const string WRITE = "$WRITE";
const string ODD = "$ODD";

const string PROGRAMNAME = "<ProgramName>";
const string PROCEDURENAME = "<PROCEDURE>";
```

```
/*Global operator definitions*/
const string VARNAME = "<var>";
const string LBRACK = "$LB";                    // '{'
const string ASSIGN = "<assign>";          // '='
const string INTEGER = "<integer>";              // 1 ... infinite
const string COMMA = "<comma>";                            // ','
const string SEMICOLON = "<semi>";               // ';'
const string RBRACK = "$RB";                     // '}'
const string RELOP = "<relop>";                  // '<', '<=', '>', '>=', '==', '!='
const string MOP = "<mop>";                                  // '*', '/'
const string ADDOP = "<addop>";                  // '+', '-'
const string LPAREN = "$LP";                     // '('
const string RPAREN = "$RP";                     // ')'
const string END_OF_FILE = "EOF";                // '\0'
const string NOT = "<NOT>";                                  // '!'
const string TEMP = "<temp>";

const string PROC_LBRACK = "PROC_LBRACK";
const string PROC_RBRACK = "PROC_RBRACK";

const string CS = "CS"; //Code segment
const string DS = "DS"; //Data segment

/*Token array of reserved words*/
const Token reserved[] = {{"CONST", CONST},
                                                 {"IF", IF},
                                                 {"VAR", VAR},
                                                 {"CLASS", CLASS},
                                                 {"THEN", THEN},
                                                 {"ELSE", ELSE},
                                                 {"WHILE", WHILE},
                                                 {"DO", DO},
                                                 {"CALL", CALL},
                                                 {"ODD", ODD},
                                                 {"PROCEDURE", PROCEDURE},
                                                 {"READ", READ},
                                                 {"WRITE", WRITE}
                                         };

/*
Class: Scanner
Parameters: (N/A)
Return: (N/A)
Description: Declares the Scanner functions
and variables that will be used when
the header file is called.
*/
class Scanner
{
public:

        /*LinkedList pointer*/
        LL* SymTable;   //Table for the symbol table.
        /*Function identifier*/
        Scanner(string SFileName);

        /*Token function identifier*/
        Token buildToken();

        /*Void function identifier*/
        void buildSTable();

private:

        /*Integer variables*/
        int rows, columns;
        int reservedKnt = 13;
        int** ScTable;  //Table for the Scanner.
        int** FPTable;  //Table for the first pass DFSA.

        /*Character variable*/
        char LChar = ' ';

        /*Filestream variables*/
        ifstream SFile;         //Source code input file.
        ifstream DTFile;  //Decision table files.
        ofstream TFile;         //Token list output file.
        ofstream STFile;  //Symbol table output file.

        /*Integer function identifiers.*/
        int isReserved(Token token);
        int charToInt(char ch);
        int tokenToInt(Token token);

        /*Character function identifier*/
        char nextChar();
};

#endif
```

---------------Token.h----------------

```
/*******************************************************
Name: Token.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
structure that will be utilized to generate the token
list, linked list, and the Assembly code that will be
generated.

		DISCLAIMER:
THIS PORTION "Token.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "LL.h".
*******************************************************/

#ifndef TOKEN_H
#define TOKEN_H

/*Include System header file*/
#include <string>

using namespace std;

/*
Struct: Token
Parameters: (N/A)
Return: (N/A)
Description: Declares the Token
variables that will be used when
the header file is called.
*/
struct Token
{
	/*String variables*/
	string TString;
	string TClass;
	string value;
	string segment;


	/*Integer variable*/
	int address;
};

#endif
```

---------------Parser.h----------------

```
/********************************************************
Name: Parser.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
class that will be utilized in the parsing of the user
provided source code. This also contains class for the
operation of a stack.

		DISCLAIMER:
THIS PORTION "Parser.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Scanner.h", "CodeGen.h", "Quad.h",
"LL.h", "Parser.cpp", and "Driver.cpp".
********************************************************/

#ifndef PARSER_H
#define PARSER_H

/*Include Program header files*/
#include "Scanner.h"
#include "CodeGen.h"
#include "Quad.h"
#include "LL.h"

/*Include System header files*/
#include <queue>
#include <stack>
#include <list>
#include <iostream>
#include <string>

using namespace std;

/*
Class: Stack
Parameters: (N/A)
Return: (N/A)
Description: Declares the Stack functions
and variables that will be used when
the header file is called.
*/
class Stack
{
	/*Function identifier*/
	Stack();

	/*Node pointer variable*/
	Node* top;
```

```
    /*Void function identifier*/
    void push(Token token);

    /*Token function identifiers*/
    Token pop();
    Token peek();
};

/*
Class: Parser
Parameters: (N/A)
Return: (N/A)
Description: Declares the Parser functions
and variables that will be used when
the header file is called.
*/
class Parser
{
public:

    /*Function identifier*/
    Parser(string SFileName);

    /*Void function identifier*/
    void sourceParse();

private:

    /*Integer variables*/
    int operatorKnt;
    int labelKnt = 1;
    int whileKnt = 1;
    int tempKnt = 1;

    /*Boolean variables*/
    bool errors = false;
    bool isQuad = false;

    /*Character variable*/
    char** PTable;

    /*String variables*/
    string label;
    string whileLabel;

    /*File stream variable*/
    ofstream QFile;
    ifstream PTFile;

    /*Scanner pointer variable*/
    Scanner* scanner;

    /*Stack variables*/
```

```
		stack <Token> t;
		stack <string> startWhile;
		stack <string> fixUp;
		stack <string> Stack_Brack;
		stack <string> Stack_Paren;

		/*List variables*/
		list <Quad> mainQ;
		list <Quad> procedureQ;

		/*Integer function identifier*/
		int classToInt(string str);

		/*Void function identifiers*/
		void isError(string oper, Quad& quad, queue <Token>& q);
		void Quad_add(Quad quad);

		/*Boolean function identifiers*/
		bool varCheck(list <Quad> quads, LL* table);
		bool assignCheck(list <Quad> quads, LL* table);

		/*String function identifier*/
		string intToClass(int num);

		/*Token function identifier*/
		Token errorRecov(Token current, int& topCol, int& choice);

		/*Used to determine syntax of variable statement*/
		int varState[4][3] =
		{
				1, -1, -1,
				-1, 2, 3,
				1, -1, -1,
				-1, -1, -1
		};

		/*Used to determine syntax of constant statement*/
		int constState[6][5] =
		{
				1, -1, -1, -1, -1,
				-1, 2, -1, -1, -1,
				-1, -1, 3, -1, -1,
				-1, -1, -1, 4, 5,
				1, -1, -1, -1, -1,
				-1, -1, -1, -1, -1
		};
};

#endif
```

---------------Quad.h----------------

```
/*******************************************************
Name: Quad.h
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a header file that creates the
structure that will be utilized in the creation of
the Quad.txt file, generation of the Assembly code,
and optimizing the completed code.

		DISCLAIMER:
THIS PORTION "Quad.h" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "CodeGen.h", and "Parser.h".
*******************************************************/

#ifndef QUAD_H
#define QUAD_H

/*Include System header files*/
#include <list>
#include <string>
#include <iostream>

using namespace std;

/*
Struct: Quad
Parameters: (N/A)
Return: (N/A)
Description: Declares the Quad function
and variables that will be used when
the header file is called.
*/
struct Quad
{
	string operation;
	string operand_1;
	string operand_2;
	string operand_3;

	string Quad_print()
	{
		string str = operation + ", " + operand_1 + ", " + operand_2 + ", " + operand_3 + "\n";
		return str;
	}
};

/*
Class: optimizeQuads
```

```
Parameters: (N/A)
Return: (N/A)
Description: Declares the optimizeQuads
functions and variables that will be
used when the header file is called.
*/
class optimizeQuads
{
public:

        /*Function identifier*/
        optimizeQuads(list<Quad> q);

        /*Function identifier*/
        list<Quad> optimAssign();

private:

        /*List variable*/
        list <Quad> quads;
};

#endif
```

--------------------------------Source Code Body Files--------------------------------

---------------CodeGen.cpp---------------

```
/*********************************************************
Name: CodeGen.cpp
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a body file that generates the .asm
file, reads portions from the IO1Nasm32Linux.asm and uses
 them as inputs to help in the generation of the code.

            DISCLAIMER:
THIS PORTION "CodeGen.cpp" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "CodeGen.h".
*********************************************************/

/*Include Program header files*/
#include "CodeGen.h"

/*
Constructor: CodeGen
Parameters: (list <Quad> q, LL* table)
Description: Creates the Assembly file
that will be generated.
*/
CodeGen::CodeGen(list <Quad> q, LL* table)
{
        /*Call variables and assign new value*/
        queue_Quad = q;
        SymTable = table;

        /*String variable*/
        string name = "AssemblyCode/";

        //Names the Assembly program from the <ProgramName> in the Symbol Table.
        if (SymTable->head->token.TClass == "<ProgramName>")
        {
                name += SymTable->head->token.TString;
        }

        name += ".asm";  //Adds '.asm' to the end of the file name

        out_stream.open(name);    //Open the file.
        if (out_stream.fail())        //Checks if opening was successful.
        {
                cout << "Error: The file could not be opened properly: " << name;
                cout << endl;
                exit(EXIT_FAILURE);     //End program.
        }
```

```
}

/*
Function: genCode
Parameters: (N/A)
Description: Uses the quads generated by the Parser
to create the Assembly language equivalent of the
source program. For the literal values, optimization
will occur.
*/
void CodeGen::genCode()
{
	/*Integer variable*/
	int choice;

	/*String variable*/
	string lit = "lit";

	/*Call this function*/
	addDSandBS();	//Adds these sections to start of Assembly file.

	//Put 'section .txt' into Assembly file.
	out_stream << "\nsection .txt\t ;Start of the main program------------------------------------------------.\n\n";

	while (queue_Quad.size() != 0)
	{
		choice = quadToInt(queue_Quad.front());

		switch (choice)
		{
		case 0:			//'+'
		{
			if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
			{
				out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
			}
			else
			{
				out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
			}

			if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
			{
				out_stream << "\tadd ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
			}
			else
			{
				out_stream << "\tadd ax, [" << queue_Quad.front().operand_2 << "]\n";
			}

			out_stream << "\tmov [" << queue_Quad.front().operand_3 << "], ax\n\n";

			break;
		}
```

```
        case 1:         //'-'
        {
            if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
            }

            if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
            {
                out_stream << "\tsub ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tsub ax, [" << queue_Quad.front().operand_2 << "]\n";
            }

            out_stream << "\tmov [" << queue_Quad.front().operand_3 << "], ax\n\n";

            break;
        }
        case 2:         //'*'
        {
            if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
            }

            if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov bx, " << queue_Quad.front().operand_2.substr(3) << "\n"
<< "\tmul word bx\n";
            }
            else
            {
                out_stream << "\tmul word [" << queue_Quad.front().operand_2 << "]\n";
            }

            out_stream << "\tmov [" << queue_Quad.front().operand_3 << "], ax\n\n";

            break;
        }
        case 3:         //'/'
        {
            out_stream << "\tmov dx, 0\n";

            if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
            }

            if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov bx, " << queue_Quad.front().operand_2.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tmov bx, [" << queue_Quad.front().operand_2 << "]\n";
            }

            out_stream << "\tdiv bx\n" << "\tmov [" << queue_Quad.front().operand_3 << "],
ax\n\n";

            break;
        }
        case 4:         //'='
        {
            if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
            {
                out_stream << "\tmov word [" << queue_Quad.front().operand_1 << "], "
                    << queue_Quad.front().operand_2.substr(3) << "\n";
            }
            else
            {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_2 << "]\n" <<
"\tmov ["
                    << queue_Quad.front().operand_1 << "], ax\n\n";
            }

            break;
        }
        case 5:         //'IF'
        {
            break;
        }
        case 6:         //'THEN'
        {
            out_stream << queue_Quad.front().operand_1 << "\n\n";

            break;
        }
        case 7:         //'L#' labels
        {
            out_stream << queue_Quad.front().operation << ":\tnop\n";

            break;
        }
```

```
case 8:         //'WHILE'
{
        out_stream << queue_Quad.front().operand_1 << ":\tnop\n";

        break;
}
case 9:         //'DO'
{
        out_stream << queue_Quad.front().operand_1 << "\n";

        break;
}
case 10: //'W#' labels
{
        out_stream << "\tjmp " << queue_Quad.front().operation << "\n";

        break;
}
case 11: //'CLASS'
{
        out_stream << "   global _start\n_start:";
        break;
}
case 12: //'PROCEDURE'
{
        out_stream << queue_Quad.front().operand_1 << ":\tnop\n";
        break;
}
case 13: //'>'
{
        if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
        {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
        {
                out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
        }

        out_stream << "\tjle ";

        break;
}
case 14: //'<'
{
```

```
                if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
                {
                        out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
                }
                else
                {
                        out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
                }

                if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
                {
                        out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
                }
                else
                {
                        out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
                }

                out_stream << "\tjge ";

                break;
        }
        case 15: //'>='
        {
                if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
                {
                        out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
                }
                else
                {
                        out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
                }

                if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
                {
                        out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
                }
                else
                {
                        out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
                }

                out_stream << "\tjl ";

                break;
        }
        case 16: //'<='
        {
                if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
                {
                        out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
                }
                else
                {
```

```
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
        {
                out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
        }

        out_stream << "\tjg ";

        break;
}
case 17: //'!='
{
        if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
        {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
        {
                out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
        }

        out_stream << "\tje ";

        break;
}
case 18: //'=='
{
        if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
        {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        if (queue_Quad.front().operand_2.compare(0, 3, lit) == 0)
        {
                out_stream << "\tcmp ax, " << queue_Quad.front().operand_2.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tcmp ax, [" << queue_Quad.front().operand_2 << "]\n";
        }

        out_stream << "\tjne ";

        break;
}
case 19: //'CALL'
{
        out_stream << "\tcall " << queue_Quad.front().operand_1 << "\n";

        break;
}
case 20: //'ODD'
{
        if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
        {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        out_stream << "\ttest al, 1\n" << "\tjz ";        //Is an even number, jump zero.

        break;
}
case 21: //'READ'
{
        out_stream << "\tcall PrintString\n"
                << "\tcall GetAnInteger\n"
                << "\tmov ax, [ReadInt]\n"
                << "\tmov [" << queue_Quad.front().operand_1 << "], ax\n\n";

        break;
}
case 22: //'WRITE'
{
        if (queue_Quad.front().operand_1.compare(0, 3, lit) == 0)
        {
                out_stream << "\tmov ax, " << queue_Quad.front().operand_1.substr(3) << "\n";
        }
        else
        {
                out_stream << "\tmov ax, [" << queue_Quad.front().operand_1 << "]\n";
        }

        out_stream << "\tcall ConvertIntegerToString\n\n"
                << "\tmov eax, 4\n"
                << "\tmov ebx, 1\n"
```

```
                                << "\tmov ecx, Result\n"
                                << "\tmov edx, ResultEnd\n"
                                << "\tint 80h\n\n";

                        break;
                }
                case 23: //'PROC_LBRACK'
                {
                        break;
                }
                case 24: //'PROC_RBRACK'
                {
                        out_stream << "\tret\n";

                        break;
                }
                case 25: //'EOF'
                {
                        out_stream << "fini:\n"
                                << "\tmov eax, sys_exit\n"
                                << "\txor ebx, ebx\n"
                                << "\tint 80h\n\n";

                        break;
                }
                }

                queue_Quad.pop_front();
        }

        addIO();

        cout << "The Assembly Language equivalent to the given code has been generated!";
        cout << endl;
}

/*
Function: addDSandBS
Parameters: (N/A)
Description: Using the PartialDS and PartialBS files derived
from "IO1NasmLinux32.asm", this function prints the .data
and .bss sections at the beginning Assembly program and prints
the constant, variable, and temporary values using the Symbol Table.
*/
void CodeGen::addDSandBS()
{
        /*String variable*/
        string line;

        /*Node pointer*/
        Node* node = SymTable->head;

        partDS.open("InputFiles/PartialDS.txt");        //Open the file.
        if (partDS.fail())  //Checks if opening was successful.
```

```
	{
		cout << "Error: The file, 'PartialDS.txt', could not be opened properly.";
		cout << endl;
		exit(EXIT_FAILURE);	//End program.
	}

	partBS.open("InputFiles/PartialBS.txt");	//Open the file.
	if (partBS.fail())	//Checks if opening was successful.
	{
		cout << "Error: The file, 'PartialBS.txt', could not be opened properly.";
		cout << endl;
		exit(EXIT_FAILURE);	//End program.
	}

	//Reads each line of the file and prints it to the new one.
	while (getline(partDS, line))
	{
		out_stream << line;
		out_stream << endl;
	}
	out_stream << endl;		//Puts a space between the two sections.

	//Reads each line of the file and prints it to the new one.
	while (getline(partBS, line))
	{
		out_stream << line;
		out_stream << endl;
	}

	//While a node.
	while (node)
	{
		//If the node is a '$CONST'
		if (node->token.TClass == "$CONST")
		{
			//Print 'DW' and its value.
			out_stream << "\t" << node->token.TString << "\tDW" << node->token.value;
			cout << endl;
		}

		//If the node is a '<var>' or a '<temp>'
		if (node->token.TClass == "<var>" || node->token.TClass == "<temp>")
		{
			//Print 'RESW 1'
			out_stream << "\t" << node->token.TString << "\tRESW 1";
			out_stream << endl;
		}

		node = node->next;
	}
}

/*
Function: addIO
```

```
Parameters: (N/A)
Description: Using the IORoutines file derived from
"IO1NasmLinux32.asm", this function prints the I/O
procedures section at the end of the Assembly program.
*/
void CodeGen::addIO()
{
        /*String variable*/
        string line;

        IOFile.open("InputFiles/IORoutines.txt");    //Open the file.
        if (IOFile.fail())   //Checks if opening was successful.
        {
                cout << "Error: The file, IORoutines.txt, could not be opened properly.";
                cout << endl;
                exit(EXIT_FAILURE);     //End program.
        }

        //Reads each line of the file and prints it to the new one.
        while (getline(IOFile, line))
        {
                out_stream << line;
                out_stream << endl;
        }
}

/*
Function: quadToInt
Parameters: (Quad qu)
Description: Produces a numeric value associated
with the operators supported by Java 0. Helps
locate the appropriate case in the genCode function.
*/
int CodeGen::quadToInt(Quad qu)
{
        if (qu.operation == "+")
        {
                return 0;
        }
        else if (qu.operation == "-")
        {
                return 1;
        }
        else if (qu.operation == "*")
        {
                return 2;
        }
        else if (qu.operation == "/")
        {
                return 3;
        }
        else if (qu.operation == "=")
        {
                return 4;
        }
        else if (qu.operation == "IF")
        {
                return 5;
        }
        else if (qu.operation == "THEN")
        {
                return 6;
        }
        else if (qu.operation.at(0) == 'L' && isdigit(qu.operation.at(1)))
        {
                return 7;
        }
        else if (qu.operation == "WHILE")
        {
                return 8;
        }
        else if (qu.operation == "DO")
        {
                return 9;
        }
        else if (qu.operation.at(0) == 'W' && isdigit(qu.operation.at(1)))
        {
                return 10;
        }
        else if (qu.operation == "CLASS")
        {
                return 11;
        }
        else if (qu.operation == "PROCEDURE")
        {
                return 12;
        }
        else if (qu.operation == ">")
        {
                return 13;
        }
        else if (qu.operation == "<")
        {
                return 14;
        }
        else if (qu.operation == ">=")
        {
                return 15;
        }
        else if (qu.operation == "<=")
        {
                return 16;
        }
        else if (qu.operation == "!=")
        {
                return 17;
        }
        else if (qu.operation == "==")
```

```
    {
        return 18;
    }
    else if (qu.operation == "CALL")
    {
        return 19;
    }
    else if (qu.operation == "ODD")
    {
        return 20;
    }
    else if (qu.operation == "READ")
    {
        return 21;
    }
    else if (qu.operation == "WRITE")
    {
        return 22;
    }
    else if (qu.operation == "PROC_LBRACK")
    {
        return 23;
    }
    else if (qu.operation == "PROC_RBRACK")
    {
        return 24;
    }
    else if (qu.operation == "EOF")
    {
        return 25;
    }
    else
    {
        return -1;
    }
}
```

---------------LL.cpp----------------

```
/*******************************************************
Name: LL.cpp
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a body file that generates and controls
the linked list that is used for the code generation.

		DISCLAIMER:
THIS PORTION "LL.cpp" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "LL.h".
*******************************************************/

/*Include Program header files*/
#include "LL.h"

/*
* DATA STRUCTURES HYMNAL PAGE 49.
*/

/*
Function: inLL
Parameters: (string TString)
Description: Checks to make sure
the node is in the Linked List.
*/
bool LL::inLL(string TString)
{
	/*Node pointer variable assignment*/
	Node* temp = this->head;

	while (temp)
	{
		if (temp->token.TString == TString)
		{
			return true;
		}

		temp = temp->next;
	}

	return false;
}

/*
Function: add
Parameters: (Token newToken)
Description: Adds nodes onto the
Linked List.
```

```
*/
void LL::add(Token newToken)
{
        /*Node pointer variable*/
        Node* temp = this->head;

        if (this->head == NULL)   //New first node or only node
        {
                this->head = new Node(newToken, NULL); //Head is a new node.
                this->last = this->head;                        //and the last node is the head.
        }
        else    //Interior node or new last node.
        {
                while (temp->next != NULL)      //While temp->next is not NULL.
                {
                        temp = temp->next;
                }

                temp->next = new Node(newToken, NULL);          //Next is a new node.
                this->last = temp->next;                        //and the last node is next.
        }
}

/*
Function: printLL
Parameters: (N/A)
Description: Prints the Symbol table
for the given source code.
*/
void LL::printLL()
{
        /*Node pointer variable assignment*/
        Node* temp = this->head;

        if (this->head == NULL)
        {
                return;
        }

        STFile.open(symStream);  //Open symbol table file.
        if (STFile.is_open())    //If symbol table file is open.
        {
                //Symbol Table output design.
                STFile << left << setw(30) << "TOKEN";          //Set field width to 30.
                STFile << left << setw(15) << "CLASS";          //Set field width to 15.
                STFile << right << setw(10) << "VALUE";         //Set field width to 10.
                STFile << right << setw(10) << "ADDRESS";       //Set field width to 10.
                STFile << right << setw(10) << "SEGMENT";       //Set field width to 10.
                STFile << endl;

                do
                {
                        //Outputs token values into the Token List.
                        STFile << left << setw(30) << temp->token.TString;      //Set field width to 30.
```

```
            STFile << left << setw(15) << temp->token.TClass;           //Set field width to 15.
            STFile << right << setw(6) << temp->token.value;            //Set field width to 6.
            STFile << right << setw(8) << temp->token.address;          //Set field width to 8.
            STFile << right << setw(11) << temp->token.segment;                 //Set field width
to 11.

            STFile << endl;
            temp = temp->next;
        } while (temp);
    }
    else
    {
        cout << "Error opening the output file" << symStream;
        cout << endl;
        exit(EXIT_FAILURE);
    }

    STFile.close();    //Close symbol table file.
}

/*
Function: getClass
Parameters: (string TString)
Description: Gets the token class
from the Linked List.
*/
string LL::getClass(string TString)
{
    /*Node pointer variable assignment*/
    Node* temp = this->head;

    while (temp)
    {
        if (temp->token.TString == TString)
        {
            return temp->token.TClass;
        }

        temp = temp->next;
    }
}
```

---------------Scanner.cpp----------------

```
/*******************************************************
Name: Scanner.cpp
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a body file that performs a 'Scan'
of the source code and passes the appropriate outputs
into .txt files.

		DISCLAIMER:
THIS PORTION "Scanner.cpp" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Scanner.h".
*******************************************************/

/*Include Program header files*/
#include "Scanner.h"

/*
Constructor: Scanner
Parameters: (string SFileName)
Description: Open and closes the files that
will be used in the creation of the Scanner
and First pass arrays. It then populates
said arrays.
*/
Scanner::Scanner(string SFileName)
{
	/*Integer variables*/
	int i, j;

	//Get input file.

	SFile.open(SFileName.c_str(), fstream::in);  //Open and read the file.
	if (SFile.fail())     //Checks if opening was successful.
	{
		cout << "Error opening source code input file " << SFileName;
		cout << endl;
		exit(EXIT_FAILURE);     //End program.
	}

	TFile.open("Token List.txt");          //Open token list.
	if (TFile.fail())     //Checks if opening was successful.
	{
		cout << "Error opening Token List.txt!";
		cout << endl;
		exit(EXIT_FAILURE);     //End program
	}

	DTFile.open("InputFiles/ScannerDFSA.txt");            //Scanner decision table.
```

```
        if (DTFile.fail())  //Checks if opening was successful.
        {
                cout << "Error opening ScannerDFSA.txt!";
                cout << endl;
                exit(EXIT_FAILURE);     //End program
        }

        DTFile >> rows;
        DTFile >> columns;

        ScTable = new int* [rows];              //Make scanner table a 2D array.
        for (i = 0; i < rows; i++)
        {
                ScTable[i] = new int[columns];

                for (j = 0; j < columns; j++)           //Populate the table.
                {
                        DTFile >> ScTable[i][j];
                }
        }
        DTFile.close();     //Close scanner decision table file.

        DTFile.open("InputFiles/FirstPassDFSA.txt");            //First pass decision table.
        if (DTFile.fail())  //Checks if opening was successful.
        {
                cout << "Error opening FirstPassDFSA.txt!";
                cout << endl;
                exit(EXIT_FAILURE);     //End program
        }

        DTFile >> rows;
        DTFile >> columns;

        FPTable = new int* [rows];              //Make first pass table a 2D array.
        for (i = 0; i < rows; i++)
        {
                FPTable[i] = new int[columns];

                for (j = 0; j < columns; j++)           //Populate the table.
                {
                        DTFile >> FPTable[i][j];
                }
        }
        DTFile.close();     //Close scanner decision table file.
}

/*
Function: buildToken
Parameters: (N/A)
Description: Uses the scanner DFSA to build
tokens one character at a time. Returns EOF
token when finished.
*/
Token Scanner::buildToken()
```

```
{
	if (SFile.tellg() == 0)		//If current position of the file pointer is at the start of the file, LChar is set to 0.
	{
		LChar = ' ';		//LChar = a whitespace.
	}

	/*Structure variable*/
	Token newToken;

	/*Initizalize new token string*/
	newToken.TString = "";

	/*Integer variables*/
	int state = 0;		//Set intial state to 0.
	int choice;

	/*Character variable*/
	char cha = LChar;

	/*Boolean variable*/
	bool finish = false;			//Set finish to a state of false.

	while (!finish)
	{
		switch (state)
		{
		default:
		{
			cout << "Error: Unkown state in the Scanner DFSA.";
			cout << endl;
			exit(EXIT_FAILURE);

			break;
		}
		case 0:			//Get next character
		{
			choice = charToInt(cha);
			state = ScTable[0][choice];

			if (state == 0)
			{
				cha = nextChar();
			}

			break;
		}
		case 1:			//ERROR
		{
			cout << "Error, there is an illegal character in the input! : " << cha;
			cout << endl;
			exit(EXIT_FAILURE);

			break;
		}
```

```
        case 2:         // Asterisk
        {
            newToken.TString += cha;

            cha = nextChar();
            choice = charToInt(cha);
            state = ScTable[2][choice];

            break;
        }
        case 3:         //Digit
        {
            if (isdigit(cha))
            {
                newToken.TString += cha;
            }

            cha = nextChar();
            choice = charToInt(cha);
            state = ScTable[3][choice];

            break;
        }
        case 4:         // <integer>,     final state
        {
            newToken.TClass = INTEGER;
            finish = true;

            break;
        }
        case 5:         //LetterDigit
        {
            if (isalnum(cha))  //Checks if 'character' is either a letter or number.
            {
                newToken.TString += cha;
            }

            cha = nextChar();
            choice = charToInt(cha);
            state = ScTable[5][choice];

            break;
        }
        case 6:         // <var>, final state
        {
            int index = isReserved(newToken); //Give index the value returned from the function
isReserved.
            if (index != -1)
            {
                newToken = reserved[index];
            }
            else if (index == -1)
            {
                newToken.TClass = VARNAME;
        }

        finish = true;

        break;
}
case 7:             //Slash
{
        newToken.TString += cha;

        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[7][choice];

        break;
}
case 8:             //Left comment (/*)
{
        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[8][choice];

        break;
}
case 9:             //Right comment (*/)
{
        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[9][choice];

        if (state == 0)
        {
                newToken.TString = "";
                cha = nextChar();
        }

        break;
}
case 10: // '/' <mop>,        final state
{
        newToken.TClass = MOP;
        finish = true;

        break;
}
case 11: //Equal
{
        newToken.TString += cha;

        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[11][choice];

        break;
```

```
}
case 12: // '=' <assign>,      final state
{
            newToken.TClass = ASSIGN;
            finish = true;

            break;
}
case 13: // '==' <relop>,     final state
{
            newToken.TClass = RELOP;
            newToken.TString += cha;
            cha = nextChar();
            finish = true;
            break;
}
case 14: //Less than
{
            newToken.TString += cha;
            cha = nextChar();
            choice = charToInt(cha);
            state = ScTable[14][choice];

            break;
}
case 15: // '<' <relop>,       final state
{
            newToken.TClass = RELOP;
            finish = true;

            break;
}
case 16: // '<=' <relop>,     final state
{
            newToken.TClass = RELOP;
            newToken.TString += cha;
            cha = nextChar();
            finish = true;

            break;
}
case 17: // '{' $LB,           final state
{
            newToken.TClass = LBRACK;
            newToken.TString += cha;
            cha = nextChar();
            finish = true;

            break;
}
case 18: // '}' $RB,           final state
{
            newToken.TClass = RBRACK;
            newToken.TString += cha;
```

```
        cha = nextChar();
        finish = true;

        break;
}
case 19: //Add
{
        newToken.TString += cha;
        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[19][choice];

        break;
}
case 20: // '+' <addop>,     final state
{
        newToken.TClass = ADDOP;
        finish = true;

        break;
}
case 21: //Subtract
{
        newToken.TString += cha;
        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[21][choice];

        break;
}
case 22: // '-' <addop>,     final state
{
        newToken.TClass = ADDOP;
        finish = true;

        break;
}
case 23: // ',' <comma>,     final state
{
        newToken.TClass = COMMA;
        newToken.TString += cha;
        cha = nextChar();
        finish = true;

        break;
}
case 24: // ';' <semi>,      final state
{
        newToken.TClass = SEMICOLON;
        newToken.TString += cha;
        cha = nextChar();
        finish = true;

        break;
```

```
}
case 25: // '\0' EOF,           final state
{
        newToken.TClass = END_OF_FILE;
        finish = true;

        break;
}
case 26: // '*' <mop>,          final state
{
        newToken.TClass = MOP;
        finish = true;

        break;
}
case 27: //Greater than
{
        newToken.TString += cha;
        cha = nextChar();
        choice = charToInt(cha);
        state = ScTable[27][choice];

        break;
}
case 28: // '>' <relop>,        final state
{
        newToken.TClass = RELOP;
        finish = true;

        break;
}
case 29: // '>=' <relop>,       final state
{
        newToken.TClass = RELOP;
        newToken.TString += cha;
        cha = nextChar();
        finish = true;

        break;
}
case 30: // '(' $LP,            final state
{
        newToken.TClass = LPAREN;
        newToken.TString += cha;
        cha = nextChar();
        finish = true;

        break;
}
case 31: // ')' $RP,            final state
{
        newToken.TClass = RPAREN;
        newToken.TString += cha;
        cha = nextChar();
```

```
                        finish = true;

                        break;
                    }
                    case 32: //Exclamation mark
                    {
                        newToken.TString += cha;
                        cha = nextChar();
                        choice = charToInt(cha);
                        state = ScTable[32][choice];

                        break;
                    }
                    case 33: // '!' <NOT>,        final state
                    {
                        newToken.TClass = NOT;
                        finish = true;

                        break;
                    }
                    case 34: // '!=' <relop>,     final state
                    {
                        newToken.TClass = RELOP;
                        newToken.TString += cha;
                        cha = nextChar();
                        choice = charToInt(cha);
                        finish = true;

                        break;
                    }
                }
            }

            LChar = cha;

            return newToken;
}

/*
Function: buildSTable
Parameters: (N/A)
Description: Creates the Symbol Table
from the Linked List. Also, creates
the token list.
*/
void Scanner::buildSTable()
{
            /*Resets the file pointer to the start of the file.*/
            SFile.clear();        //Remove flags and allow for further operations to be attempted on file stream.
            SFile.seekg(0, SFile.beg);  //Sets the pointer to the beginning of the file stream.

            /*SymTable variable*/
            SymTable = new LL("Symbol Table.txt");
```

```
/*Structure variable*/
Token newToken;

/*Integer variables*/
int choice, i;
int DSA = 0;
int CSA = 0;
int state = 0;

/*Boolean variables*/
bool finish = false;

while (!finish)
{
        switch (state)
        {
        default:
        {
                cout << "Error: Token class is unknown.";
                cout << endl;
                exit(EXIT_FAILURE);
                break;
        }
        case 0:	//Start
        {
                newToken = buildToken();
                choice = tokenToInt(newToken);
                state = FPTable[0][choice];

                break;
        }
        case 1:	//Class
        {
                newToken = buildToken();
                choice = tokenToInt(newToken);
                state = FPTable[1][choice];

                break;
        }
        case 2:	//Add '<ProgramName>' to the Symbol Table.		<var>
        {
                newToken.TClass = PROGRAMNAME;
                newToken.address = CSA;
                newToken.segment = CS;
                CSA += 2;
                SymTable->add(newToken);

                newToken = buildToken();
                choice = tokenToInt(newToken);
                state = FPTable[2][choice];

                break;
        }
        case 3:	// '{', ';'
```

```
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[3][choice];

        break;
}
case 4: //CONST, ','
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[4][choice];

        break;
}
case 5: //Add '$CONST' to the Symbol Table.         <var>
{
        newToken.TClass = CONST;
        newToken.address = DSA;
        newToken.segment = DS;
        CSA += 2;
        SymTable->add(newToken);

        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[5][choice];

        break;
}
case 6: // '='
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[6][choice];

        break;
}
case 7: //Assign the last value to the 'CONST' variable.        <int>
{
        SymTable->last->token.value = newToken.TString;
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[7][choice];

        break;
}
case 8: //VAR, ','
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[8][choice];

        break;
}
```

```
case 9:     //Add variable to the Symbol Table <var>
{
            if (!SymTable->inLL(newToken.TString))
            {
                        newToken.address = DSA;
                        newToken.segment = DS;
                        newToken.value = '?';
                        DSA += 2;
                        SymTable->add(newToken);
            }

            newToken = buildToken();
            choice = tokenToInt(newToken);
            state = FPTable[9][choice];

            break;
}
case 10: //Any, reserved words <var>
{
            newToken = buildToken();
            choice = tokenToInt(newToken);
            state = FPTable[10][choice];

            break;
}
case 11: //Add integer to the Symbol Table.  <int>
{
            newToken.TString = "lit" + newToken.TString;

            if (!SymTable->inLL(newToken.TString))
            {
                        newToken.address = DSA;
                        newToken.segment = DS;
                        newToken.value = newToken.TString.substr(3);        //Returns 'lit' with its value initialized to a copy of a sub-string.
                        DSA += 2;
                        SymTable->add(newToken);
            }

            newToken = buildToken();
            choice = tokenToInt(newToken);
            state = FPTable[11][choice];

            break;
}
case 12: //Reached 'EOF' (End of pass 1).
{
            finish = true;

            break;
}
case 13:
{
            newToken = buildToken();
```

```
        choice = tokenToInt(newToken);
        state = FPTable[13][choice];

        break;
}
case 14: //Add '<PROCEDURE>' to the Symbol Table.
{
        if (!SymTable->inLL(newToken.TString))
        {
                newToken.TClass = PROCEDURENAME;
                newToken.address = CSA;
                newToken.segment = CS;
                CSA += 2;
                SymTable->add(newToken);
        }

        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[14][choice];

        break;
}
case 15:
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[15][choice];

        break;
}
case 16: //Add '<PROCEDURE>' parameter(s) to the Symbol Table.
{
        newToken.address = CSA;
        newToken.segment = CS;
        CSA += 2;
        SymTable->add(newToken);

        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[16][choice];

        break;
}
case 17:
{
        newToken = buildToken();
        choice = tokenToInt(newToken);
        state = FPTable[17][choice];
        break;
}
}

if (newToken.TClass != END_OF_FILE)
{
```

```
            TFile << newToken.TString << "\t" << newToken.TClass;
            TFile << endl;
        }
    }

    TFile.close();      //Close token list file.

    //Add '<temp>' variables to the Symbol Table.
    newToken.TClass = "";
    newToken.value = "?";       //Prints '?' in value position.
    newToken.segment = DS;

    for (i = 0; i < 8; i++)
    {
        newToken.TString = "T" + to_string(i + 1);  //Returns string with value of i + 1.
        newToken.TClass = TEMP;
        newToken.address = DSA;
        DSA += 2;
        SymTable->add(newToken);
    }

    SymTable->printLL();
}

/*
Function: isReserved
Parameters: (Token token)
Description: Checks to see if the
string value is one of the reserved
words.
*/
int Scanner::isReserved(Token token)
{
    /*Integer variables*/
    int i;

    for (i = 0; i < reservedKnt; i++)
    {
        if (reserved[i].TString == token.TString)
        {
            return i;
        }
    }

    return -1;
}

/*
Function: charToInt
Parameters: (char ch)
Description: Converts a specified character
to a numeric value that can be passed to
another function. Unrecognized characters
are passed to a function that displays
```

```
an error message.
*/
int Scanner::charToInt(char ch)
{
	if (isalpha(ch))		//Letter
	{
		return 0;
	}
	else if (isdigit(ch))		//Digit
	{
		return 1;
	}
	else if (ch == '*')		// '*', Multiplication
	{
		return 2;
	}
	else if (ch == '/')		// '/', Division
	{
		return 3;
	}
	else if (ch == '=')		// '=', Assignment
	{
		return 4;
	}
	else if (ch == '<')		// '<', Less than
	{
		return 5;
	}
	else if (isspace(ch))		// ' ' Whitespace
	{
		return 6;
	}
	else if (ch == '{')		// '{', Left bracket
	{
		return 7;
	}
	else if (ch == '}')		// '}', Right bracket
	{
		return 8;
	}
	else if (ch == '+')		// '+', Addition
	{
		return 9;
	}
	else if (ch == '-')		// '-', Subtract
	{
		return 10;
	}
	else if (ch == ',')		// ',', Comma
	{
		return 11;
	}
	else if (ch == ';')		// ';', Semicolon
	{
```

```
                return 12;
        }
        else if (ch == '\0') //End of the file.
        {
                return 13;
        }
        else if (ch == '>')     // '>', Greate than
        {
                return 14;
        }
        else if (ch == '(')     // '(', Left parenthesis
        {
                return 15;
        }
        else if (ch == ')')     // ')', Right parenthesis
        {
                return 16;
        }
        else if (ch == '!')     // '!', Exclamation point
        {
                return 17;
        }
        else                                    //Other
        {
                return 18;
        }
}

/*
Function: tokenToInt
Parameters: (Token token)
Description: Converts a recognized token's class
to a numeric value.
*/
int Scanner::tokenToInt(Token token)
{
        if (token.TClass == CLASS)                              // $CLASS
        {
                return 0;
        }
        else if (token.TClass == VARNAME)               // <var>
        {
                return 1;
        }
        else if (token.TClass == LBRACK)        // '{'   $LB
        {
                return 2;
        }
        else if (token.TClass == RBRACK)        // '}'   $RB
        {
                return 3;
        }
        else if (token.TClass == SEMICOLON)             // ';'     <semi>
        {
```

```
        return 4;
    }
    else if (token.TClass == CONST)                 // $CONST
    {
        return 5;
    }
    else if (token.TClass == VAR)                   // <$var>
    {
        return 6;
    }
    else if (token.TClass == ASSIGN)        // '='     <assign>
    {
        return 7;
    }
    else if (token.TClass == INTEGER)               // 1 ... infinite    <integer>
    {
        return 8;
    }
    else if (token.TClass == COMMA)                 // ','      <comma>
    {
        return 9;
    }
    else if (token.TClass == END_OF_FILE)   // '\0'    EOF
    {
        return 10;
    }
    else if (isReserved(token) != -1 && token.TClass != PROCEDURE)    //Reserved words that are not
$PROCEDURE.
    {
        return 11;
    }
    else if (token.TClass == ADDOP || token.TClass == MOP)                 // '+', '-', '*', '/'     <addop>
and <mop>
    {
        return 12;
    }
    else if (token.TClass == PROCEDURE)             // $PROCEDURE
    {
        return 13;
    }
    else if (token.TClass == LPAREN)        // '('     $LP
    {
        return 14;
    }
    else if (token.TClass == RPAREN)        // ')'     $RP
    {
        return 15;
    }
    else if (token.TClass == RELOP)                 // '<', '<=', '>', '>=', '==', '!=', <relop>
    {
        return 16;
    }
}
```

```
/*
Function: nextChar
Parameters: (N/A)
Description: Gets the next available character
from the Source Code file. If it's not EOF reads
the next character else returns '\0'.
*/
char Scanner::nextChar()
{
        /*Char variable*/
        char cha;

        if (SFile.peek() != EOF)    //If not End of File
        {
                SFile >> noskipws >> cha;        //Read next char skipping whitespaces.
        }
        else        //End of File
        {
                cha = '\0';
        }

        return cha;
}
```

---------------Parser.cpp----------------

```
/*********************************************************
Name: Parser.cpp
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is a body file that parses the source
code, generates and optimizes the associated quads, and
generates and operates the stack.

        DISCLAIMER:
THIS PORTION "Parser.cpp" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Parser.h".
*********************************************************/

/*Include Program header files*/
#include "Parser.h"

/*DATA STRUCTURES HYMNAL PAGE 30*/

/*
Constructor: Stack
Parameters: (N/A)
Description: Initializes stack to NULL.
*/
Stack::Stack()
{
        top = NULL;             //Indicates empty stack/list.
}

/*
Function: push
Parameters: (Token token)
Description: Pushes a new element onto
the stack incrementing it by one.
*/
void Stack::push(Token token)
{
        Node* newNode = NULL;

        if (nullptr != newNode)
        {
                newNode->token = token; //Pt.Info <- Y.
                newNode->next = top;    //Pt.Link <- Top.
                top = newNode;                  //Top <- Pt.
        }
        else    //Overflow
        {
                cout << "\nStack overflow.";
                exit(EXIT_FAILURE);
```

```
		}
}

/*
Function: pop
Parameters: (N/A)
Description: Pops the top element from
the stack decrementing it by one.
*/
Token Stack::pop()
{
		Token token;

		if (top == NULL)
		{
				cout << "\nStack Underflow.";
				return token;		//Underflow, empty stack.
		}
		else
		{
				Node* temp = top;		//Pt <- Top.
				token = top->token;		//Y <- Top.Info.
				top = top->next;	//Top <- Top.Link.

				delete temp;		//Pt => Avail, Avoid memory hemorraging.
				return token;
		}
}

/*
Function: peek
Parameters: (N/A)
Description: Returns top element of the
stack without modifying the stack.
*/
Token Stack::peek()
{
		Token token;
		if (top == NULL)//If stack is empty.
		{
				return token;
		}
		else		//Else stack isn't empty.
		{
				token = top->token;
				return token;
		}
}

/*
Constructor: optimizeQuads
Parameters: (list<Quad> q)
Description: Gives quads the value
of 'q'.
```

```
*/
optimizeQuads::optimizeQuads(list<Quad> q)
{
    quads = q;

}

/*
Function: optimAssign
Parameters: (N/A)
Description: Optimizes the quads by
removing some redundant code.
*/
list<Quad> optimizeQuads::optimAssign()
{
    list<Quad> Quads_new;

    while (!quads.empty())      //While quads list is not empty.
    {
        if (quads.front().operation == "=")   //If the front of the quads list is '='.
        {
            //Returns the operation reference to the last element in the list container.
            if (Quads_new.back().operation == "+" || Quads_new.back().operation == "-" ||
                Quads_new.back().operation == "*" || Quads_new.back().operation == "/")
            {
                //If operand_3 on the back of the list is equal to operand_2 on the front of the
list.
                if (Quads_new.back().operand_3 == quads.front().operand_2)
                {
                    //Assign operand_1 at the front of the list to operand_3 at the back.
                    Quads_new.back().operand_3 = quads.front().operand_1;
                }
                else    //Add the front of the quad to the end of the Quads_new container.
                {
                    Quads_new.push_back(quads.front());
                }
            }
            else    //Add the front of the quad to the end of the Quads_new container.
            {
                Quads_new.push_back(quads.front());
            }
        }
        else    //Add the front of the quad to the end of the Quads_new container.
        {
            Quads_new.push_back(quads.front());
        }

        quads.pop_front();          //Pops the front of the quads list.
    }

    return Quads_new;
}

/*
```

```
Constructor: Parser
Parameters: (string SFileName)
Description: Open and closes the file that
will be used in the creation of the Precedence
array. It then populates said array. It also
creates the Quad file here.
*/
Parser::Parser(string SFileName)
{
        /*Integer variables*/
        int i, j;

        scanner = new Scanner(SFileName);

        QFile.open("Quad File.txt");            //Open file
        if (QFile.fail())    //Check if opening was successful.
        {
                cout << "Error opening Quad File.txt";
                cout << endl;
                exit(EXIT_FAILURE);     //End program.
        }

        PTFile.open("InputFiles/precedenceTable.txt");          //Open file
        if (PTFile.fail())   //Check if opening was successful.
        {
                cout << "Error opening precedenceTable.txt";
                cout << endl;
                exit(EXIT_FAILURE);     //End program.
        }

        PTFile >> operatorKnt;

        PTable = new char* [operatorKnt]; //Make Precedence Table a 2D array.
        for (i = 0; i < operatorKnt; i++)
        {
                PTable[i] = new char[operatorKnt];

                for (j = 0; j < operatorKnt; j++)       //Populate the table.
                {
                        PTFile >> PTable[i][j];
                }
        }
        PTFile.close();    //Close precedence table file.
}

/*
Function: sourceParse
Parameters: (N/A)
Description: Retrieves one token at a time
from the Scanner and either pushes onto the
stack or pops from the stack in order to
generate the quads. Also, determines the
precedence of an operator as it comes in.
*/
```

```
void Parser::sourceParse()
{
    /*Token structure variables*/
    Token token;
    Token semiT;
    Token temp;

    /*Quad structure variables*/
    Quad LBracket;
    Quad RBracket;
    Quad quad;
    Quad Quad_IF;
    Quad Quad_WHILE;
    Quad Quad_ThenDo;
    Quad Quad_EOF;

    /*Boolean variable*/
    bool finish;

    /*Integer variables*/
    int choice;
    int topCol = 0;

    /*Character variable*/
    char prec;

    semiT.TString = ";";
    semiT.TClass = SEMICOLON;
    t.push(semiT);    //Pushes the initial semicolon onto the stack.

    //Get the next token.
    token = scanner->buildToken();
    choice = classToInt(token.TClass); //Pass the return value to the variable.

    while (token.TClass != END_OF_FILE)     //While not the End of the File.
    {
        finish = false;

        while (!finish)     //While not finished.
        {
            if (token.TClass == END_OF_FILE)        //If End of File, exit.
            {
                break;
            }
            else if (choice == -1)      //Else push it onto the stack.
            {
                if (token.TClass == INTEGER)
                {
                    token.TString = "lit" + token.TString;        //Gives an INTEGER the
form of "lit#".
                }
                t.push(token);      //Pushes the token onto the stack.

                //Get the next token.
                token = scanner->buildToken();
                choice = classToInt(token.TClass);  //Pass the return value to the variable.
            }
            else if (t.top().TClass == SEMICOLON && token.TClass == SEMICOLON)      //Else, if
it's a SEMICOLON, ';', finish.
            {
                //Get the next token.
                token = scanner->buildToken();
                choice = classToInt(token.TClass);  //Pass the return value to the variable.
                finish = true;
            }
            else        //Else it's an operator.
            {
                prec = PTable[topCol][choice];

                if (prec == '<' || prec == '=')             //Top of the stack yields to the incoming
operator.
                {
                    if (token.TClass == LBRACK || token.TClass == PROC_LBRACK)
    //Left bracket.
                    {
                        Stack_Brack.push(token.TClass);     //Pushes the token class
onto the bracket stack.
                        t.push(token);       //Pushes the token onto the stack.
                        topCol = classToInt(token.TClass); //Pass the return value to
the variable.

                        LBracket = { token.TString, "?", "?", "?" };   //Left bracket
quad.

                        QFile << LBracket.Quad_print();     //Print left bracket quad
into Quad file.
                        Quad_add(LBracket);        //Send LBracket to Quad_add
function.

                        //Get the next token.
                        token = scanner->buildToken();
                        choice = classToInt(token.TClass);  //Pass the return value to
the variable.
                    }
                    else if (token.TClass == RBRACK || token.TClass ==
PROC_RBRACK)            //Right bracket.
                    {
                        if (Stack_Brack.empty())    //Error since right bracket is
missing matching left bracket.
                        {
                            cout << "Error: missing a matching pair of brackets";
                            cout << endl;

                            errorRecov(token, topCol, choice);
                        }
                        else        //No error since right bracket has matching left
bracket.
                        {
                            if (Stack_Brack.top() != PROC_LBRACK)  //Create
right bracket quad.
                            {
                                RBracket = { token.TString, "?", "?", "?" };
    //Right bracket quad.

                                mainQ.push_back(RBracket);         //Adds
RBracket to the end of the mainQ container.
                            }
                            else    //Create PROC_RBRACK quad.
                            {
                                RBracket = { PROC_RBRACK, "?", "?",
"?" };
                                procedureQ.push_back(RBracket);  //Adds
RBracket to the end of the procedureQ container.

                                isQuad = false;     //Not a quad.
                                RBracket.operation = "}";
                            }

                            Stack_Brack.pop();          //Pop the bracket stack.
                            t.pop();   //Pops the token from the stack.

                            QFile << RBracket.Quad_print();    //Print right
bracket quad into Quad file.

                            topCol = classToInt(t.top().TClass);//Pass the return
value to the variable.

                            token = semiT;
                            choice = 0;
                        }
                    }
                    else if (token.TClass == LPAREN) //Left parenthesis.
                    {
                        Stack_Paren.push(token.TClass);    //Pushes the token class
onto the parenthesis stack.

                        topCol = choice;
                        t.push(token);      //Pushes the token onto the stack.

                        //Get the next token.
                        token = scanner->buildToken();
                        choice = classToInt(token.TClass);  //Pass the return value to
the variable.
                    }
                    else if (token.TClass == RPAREN) //Right parenthesis.
                    {
                        if (Stack_Paren.size() > 0)
                        {
                            Stack_Paren.pop();          //Pop parenthesis stack.
                            topCol = choice;
                            t.push(token);      //Pushes the token onto the stack.

                            //Get the next token.
                            token = scanner->buildToken();
                            choice = classToInt(token.TClass);  //Pass the return
value to the variable.
                        }
                        else        //An error.
                        {
                            cout << "Error: missing a matching pair of
parenthesis";

                            cout << endl;

                            errorRecov(token, topCol, choice);
                        }
                    }
                    else        //Neither a bracket, '{' '}', nor a parenthesis, '(' ')'.
                    {
                        t.push(token);      //Pushes the token onto the stack.

                        if (token.TClass == IF)      // If an 'IF' statement.
                        {
                            Quad_IF = { token.TString, "?", "?", "?" };    //IF
statement quad.


                            QFile << Quad_IF.Quad_print();     //Print IF
statement quad into Quad file.

                            Quad_add(Quad_IF);        //Send Quad_IF to
Quad_add function.

                        }
                        else if (token.TClass == THEN || token.TClass == DO)
    //Else, if a 'THEN' or 'DO' statement.

                        {
                            label = "L" + to_string(labelKnt);
                            fixUp.push(label);            //Pushed the label onto the
fixUp stack.

                            labelKnt++;

                            Quad_ThenDo = { token.TString, label, "?", "?" };
    //THEN and DO statement quad.


                            QFile << Quad_ThenDo.Quad_print();         //Print
ThenDo statement quad into Quad file.

                            Quad_add(Quad_ThenDo);            //Send
Quad_ThenDo to Quad_add function.

                        }
                        else if (token.TClass == WHILE)    // Else, if a 'WHILE'
statement.

                        {
                            whileLabel = "W" + to_string(whileKnt);
                            whileKnt++;
                            startWhile.push(whileLabel);          //Pushes the
WHILE label onto the startWhile stack.


                            Quad_WHILE = { token.TString, whileLabel, "?",
"?" };    //WHILE statement quad.
```

```
                            QFile << Quad_WHILE.Quad_print();        //Print
WHILE statement quad into Quad file.
                            Quad_add(Quad_WHILE);        //Send
Quad_WHILE to Quad_add function.
                        }

                        topCol = choice;

                        //Get the next token.
                        token = scanner->buildToken();
                        choice = classToInt(token.TClass);  //Pass the return value to
the variable.
                    }
                }
                else if (prec == '>')        //Top of the stack takes precedence over the
incoming operator.
                {
                    /*Queue variable*/
                    queue <Token> q;

                    /*Integer variables*/
                    int operatorN;
                    int popped = -1;

                    while (prec == '>' || prec == '=' || prec == 'X' || prec == 'x')
                    {
                        operatorN = classToInt(t.top().TClass);        //Pass the return
value to the variable.

                        if (operatorN != -1)
                        {
                            if (popped != -1)
                            {
                                prec = PTable[operatorN][popped];

                                if (prec == '>' || prec == '=' || prec == 'X' ||
prec == 'x')
                                {
                                    popped = operatorN;
                                    q.push(t.top());    //Pushes the top
token onto the quad stack.
                                    t.pop();
                                }
                            }
                            else
                            {
                                popped = operatorN;
                                q.push(t.top());    //Pushes the top token
onto the quad stack.
                                t.pop();  //Pops the token from the stack.

                            }
                        }
                        else        //operatorN == -1.
                        {
                            if (t.top().TClass == TEMP)
                            {
                                //Generates a sub string then converts it to
an integer and allows the reuse
                                //of the temporaries that have been popped
from the stack.
                                tempKnt = stoi(t.top().TString.substr(1));
                            }

                            q.push(t.top());    //Pushes the top token onto the
quad stack.
                            t.pop();  //Pops the token from the stack.
                        }
                    }

                    switch (popped)
                    {
                    case 0:    //Semicolon, ';'.
                    {
                        break;
                    }
                    case 1:    //Assignment, '='.
                    {
                        isError("=", quad, q);
                        quad.operand_3 = "?";      //Prints '?' in operand position.

                        QFile << quad.Quad_print();        //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 2:    //Add operator, '+' and '-'.
                    {
                        isError("ADDOP", quad, q);
                        temp.TString = "T" + to_string(tempKnt);    //Convert a
numeric value to string.
                        tempKnt++;

                        quad.operand_3 = temp.TString;
                        temp.TClass = TEMP;
                        t.push(temp);      //Pushes the temporary onto the stack.

                        QFile << quad.Quad_print();        //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 3:    //Left parenthesis, '('.
                    {
                        //If front of quad equals a right parenthesis.
                        if (q.front().TClass == RPAREN)
                        {
                            q.pop();  //Pop the quad.
                        }

                        while (q.size() != 0)
                        {
                            if (classToInt(q.front().TClass) == -1)
                            {
                                t.push(q.front());  //Pushes the front quad
onto the stack.
                            }
                            q.pop();  //Pops the quad from the stack.
                        }

                        break;
                    }
                    case 4:   //Right parenthesis, ')'
                    {
                        break;
                    }
                    case 5:   //Multiplication operator, '*' and '/'
                    {
                        isError("MOP", quad, q);

                        temp.TString = "T" + to_string(tempKnt);     //Convert a
numeric value to a string.

                        temp.TClass = TEMP;

                        quad.operand_3 = temp.TString;
                        t.push(temp);       //Pushes the temporary onto the stack.

                        tempKnt++;

                        QFile << quad.Quad_print();          //Print the quad into Quad
file.

                        Quad_add(quad);

                        break;
                    }
                    case 6:   //IF
                    {
                        quad.operation = fixUp.top();
                        fixUp.pop();        //Pops the quad from the stack.

                        quad.operand_1 = "?";      //Prints '?' in operand position.
                        quad.operand_2 = "?";      //Prints '?' in operand position.
                        quad.operand_3 = "?";      //Prints '?' in operand position.

                        QFile << quad.Quad_print();          //Print the quad into Quad
file.

                        Quad_add(quad);

                        break;
                    }
                    case 7:   // THEN
                    {
                              break;
                    }
                    case 8:   //ODD
                    {
                              if (q.front().TClass != VARNAME && q.front().TClass != 
INTEGER)          //Error
                              {
                                        cout << "Error: Missing an operand after 'ODD'.";
                                        cout << endl;

                                        quad.operand_1 = "ERROR";          //Prints 
"ERROR" in operand position.

                                        errors = true;        //Is error.
                              }
                              else          //Good
                              {
                                        quad.operand_1 = q.front().TString;
                                        q.pop();  //Pops the quad from the stack.
                              }

                              quad.operation = q.front().TString;
                              q.pop();  //Pops the quad from the stack.

                              quad.operand_2 = "?";       //Prints '?' in operand position.
                              quad.operand_3 = "?";       //Prints '?' in operand position.

                              QFile << quad.Quad_print();          //Print the quad into Quad 
file.

                              Quad_add(quad);

                              break;
                    }
                    case 9:   //Relational Operator, '<', '<=', '>', '>=', '==', and '!='.
                    {
                              isError("RELOP", quad, q);
                              quad.operand_3 = "?";       //Prints '?' in operand position.

                              QFile << quad.Quad_print();          //Print the quad into Quad 
file.

                              Quad_add(quad);

                              break;
                    }
                    case 10: //Left bracket, '{'.
                    {
                              break;
                    }
                    case 11: //Right bracket, '}'
                    {
                              break;
                    }
                    case 12: //CALL
                    {
                        if (q.front().TClass == RPAREN)
                        {
                            while (q.front().TClass != LPAREN)
                            {
                                q.pop();  //Pops the quad from the stack.
                            }
                            q.pop();  //Pops the quad from the stack.
                        }

                        if (q.front().TClass == VARNAME)
                        {
                            quad.operand_1 = q.front().TString;
                            q.pop();  //Pops the quad from the stack.
                        }
                        else if (q.front().TClass != VARNAME)
                        {
                            cout << "Error: Missing the procedure identifier after
the keyword 'CALL'.";

                            cout << endl;

                            quad.operand_1 = "ERROR";         //Prints
"ERROR" in operand position.


                            errors = true;       //Is error.
                        }

                        quad.operation = q.front().TString;
                        q.pop();  //Pops the quad from the stack.

                        quad.operand_2 = "?";       //Prints '?' in operand position.
                        quad.operand_3 = "?";       //Prints '?' in operand position.

                        QFile << quad.Quad_print();          //Print the quad into Quad
file.

                        Quad_add(quad);

                        break;
                    }
                    case 13: //WHILE
                    {
                        quad.operation = startWhile.top();
                        startWhile.pop(); //Pops the WHILE from the stack.

                        quad.operand_1 = "?";       //Prints '?' in operand position.
                        quad.operand_2 = "?";       //Prints '?' in operand position.
                        quad.operand_3 = "?";       //Prints '?' in operand position.

                        QFile << quad.Quad_print();          //Print the quad into Quad
file.

                        Quad_add(quad);

                        quad.operation = fixUp.top();
                        fixUp.pop();        //Pops the quad from the stack.

                        QFile << quad.Quad_print();        //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 14: //DO
                    {
                        break;
                    }
                    case 15: //COMMA, ','
                    {
                        break;
                    }
                    case 16: //CLASS
                    {
                        if (q.front().TClass == VARNAME)
                        {
                            quad.operand_1 = q.front().TString;
                            q.pop();  //Pops the quad from the stack.
                        }
                        else if (q.front().TClass != VARNAME)
                        {
                            cout << "Error: Missing the class identifier.";
                            cout << endl;

                            quad.operand_1 = "ERROR";        //Prints
"ERROR" in operand position.

                            errors = true;        //Is error.
                        }

                        quad.operation = q.front().TString;
                        q.pop();  //Pops the quad from the stack.

                        quad.operand_2 = "?";        //Prints '?' in operand position.
                        quad.operand_3 = "?";        //Prints '?' in operand position.

                        QFile << quad.Quad_print();        //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 17: //VAR
                    {
                        int state = 0;        //Initial switch state.
                        int index;
                        bool varCheck = true;        //There is a variable.

                        while (varCheck)
                        {
                            if (q.size() > 0)
                            {
                                if (q.front().TClass == VARNAME)
                                {
                                    index = 0;
                                }
                                else if (q.front().TClass == COMMA)
                                {
                                    index = 1;
                                }
                                else if (q.front().TClass == VAR)
                                {
                                    index = 2;
                                }
                                else    //Error
                                {
                                    cout << "Error: The token " <<
q.front().TString << ", within the 'VAR' statement, is unexpected.";
                                    cout << endl;

                                    errors = true;      //Is error.

                                    break;
                                }
                            }

                            switch (state)
                            {
                            case 0:
                            {
                                if (index == 0)     //No error
                                {
                                    state = varState[0][index];
                                }
                                else    //Error
                                {
                                    cout << "Error: Missing a variable
within the 'VAR' statement.";
                                    cout << endl;

                                    varCheck = false;//No variable.
                                    errors = true;      //Is error.
                                }

                                break;
                            }
                            case 1:
                            {
                                if (index == 1 || index == 2)           //No
error
                                {
                                    state = varState[1][index];
                                }
                                else    //Error
```

```
                        {
                            cout << "Error: The token " <<
q.front().TString << ", within the 'VAR' statement, is missing a ";

                            cout << "'" << COMMA << "'";
                            cout << endl;

                            varCheck = false;   //No variable.
                            errors = true;      //Is error.
                        }

                        break;
                    }
                    case 2:
                    {
                        if (index == 0)     //No error
                        {
                            state = varState[2][index];
                        }
                        else        //Error
                        {
                            cout << "Error: Missing a variable
within the 'VAR' statement.";

                            cout << endl;

                            varCheck = false;   //No variable.
                            errors = true;      //Is error.
                        }
                    }
                    case 3:  //Error
                    {
                        varCheck = false;   //No variable.

                        break;
                    }
                    }

                    if (q.size() > 0)
                    {
                        q.pop();
                    }
                }

                break;
            }
            case 18: //CONST
            {
                int state = 0;      //Initial switch state.
                int index;
                bool constCheck = true;     //There is a constant.

                while (constCheck)
                {
                    if (q.size() > 0)
                    {
                        if (q.front().TClass == INTEGER)
                        {
                            index = 0;
                        }
                        else if (q.front().TClass == ASSIGN)
                        {
                            index = 1;
                        }
                        else if (q.front().TClass == VARNAME)
                        {
                            index = 2;
                        }
                        else if (q.front().TClass == COMMA)
                        {
                            index = 3;
                        }
                        else if (q.front().TClass == CONST)
                        {
                            index = 4;
                        }
                        else    //Error
                        {
                            cout << "Error: The token " <<
q.front().TString << ", within the 'CONST' statement, is unexpected.";
                            cout << endl;

                            errors = true;    //Is error.

                            break;
                        }
                    }

                    switch (state)
                    {
                    case 0:
                    {
                        if (index == 0)    //No error
                        {
                            state = constState[0][index];
                        }
                        else    //Error
                        {
                            cout << "Error: Missing an integer
value within the 'CONST' statement.";
                            cout << endl;

                            constCheck = false;        //No
constant.
                            errors = true;    //Is error.
                        }

                        break;
                    }
                    case 1:
```

```
{
        if (index == 1)     //No error
        {
                state = constState[1][index];
        }
        else        //Error
        {
                cout << "Error: Missing a '=' within the 'CONST' statement.";
                cout << endl;
                constCheck = false;         //No constant.
                errors = true;      //Is error.
        }

        break;
}
case 2:
{
        if (index == 2)     //No error
        {
                state = constState[2][index];
        }
        else        //Error
        {
                cout << "Error: Missing a variable value within the 'CONST' statement.";
                cout << endl;
                constCheck = false;         //No constant.
                errors = true;      //Is error.
        }

        break;
}
case 3:
{
        if (index == 3 || index == 4)           //No error
        {
                state = constState[3][index];
        }
        else        //Error
        {
                cout << "Error: The token " << q.front().TString << ", within the 'CONST' statement, is unexpected.";
                cout << endl;
                constCheck = false;         //No constant.
                errors = true;      //Is error.
        }
```

```
                                break;
                            }
                            case 4:
                            {
                                if (index == 0)     //No error
                                {
                                    state = constState[4][index];
                                }
                                else    //Error
                                {
                                    cout << "Error: Missing an integer
value within the 'CONST' statement.";

                                    cout << endl;

                                    constCheck = false;         //No
constant.

                                    errors = true;      //Is error.
                                }

                                break;
                            }
                            case 5:  //Error
                            {
                                constCheck = false;         //No constant.

                                break;
                            }
                            }

                            if (q.size() > 0)
                            {
                                q.pop();  //Pops the quad from the stack.
                            }
                        }

                        break;
                    }
                    case 19: //PROCEDURE
                    {
                        isQuad = true;      //Is a quad.

                        token.TClass = PROC_LBRACK;

                        if (q.front().TClass == RPAREN)
                        {
                            while (q.front().TClass != LPAREN)
                            {
                                q.pop();  //Pops the quad from the stack.
                            }

                            q.pop();  //Pops the quad from the stack.
                        }
                        if (q.front().TClass != VARNAME)//Error
                        {
                            cout << "Error: Missing the procedure identifier.";
                            cout << endl;

                            quad.operand_1 = "ERROR";        //Prints
"ERROR" in operand position.

                            errors = true;     //Is error.
                        }
                        else if (q.front().TClass == VARNAME)     //No error
                        {
                            quad.operand_1 = q.front().TString;
    //<ProcedureName>
                            q.pop();  //Pops the quad from the stack.
                        }

                        quad.operation = q.front().TString;  //"PROCEDURE"
                        q.pop();  //Pops the quad from the stack.

                        quad.operand_2 = "?";     //Prints '?' in operand position.
                        quad.operand_3 = "?";     //Prints '?' in operand position.

                        QFile << quad.Quad_print();        //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 20: //READ
                    {
                        if (q.front().TClass != VARNAME)//Error
                        {
                            cout << "Error: After 'READ', there is a missing
variable operand.";

                            cout << endl;

                            quad.operand_1 = "ERROR";        //Prints
"ERROR" in operand position.

                            errors = true;     //Is error.
                        }
                        else if (q.front().TClass == VARNAME)     //No error
                        {
                            quad.operand_1 = q.front().TString;
                            q.pop();  //Pops the quad from the stack.
                        }

                        quad.operation = q.front().TString;
                        q.pop();  //Pops the quad from the stack.

                        quad.operand_2 = "?";     //Prints '?' in operand position.
                        quad.operand_3 = "?";     //Prints '?' in operand position.
                        QFile << quad.Quad_print();          //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    case 21: //WRITE
                    {
                        if (q.front().TClass != VARNAME && q.front().TClass !=
INTEGER)
                        {
                            cout << "Error: After 'WRITE', there is a missing
variable or integer operand.";
                            cout << endl;

                            quad.operand_1 = "ERROR";        //Prints
"ERROR" in operand position.

                            errors = true;       //Is error.
                        }
                        else
                        {
                            quad.operand_1 = q.front().TString;
                            q.pop();  //Pops the quad from the stack.
                        }

                        quad.operation = q.front().TString;
                        q.pop();  //Pops the quad from the stack.

                        quad.operand_2 = "?";      //Prints '?' in operand position.
                        quad.operand_3 = "?";      //Prints '?' in operand position.

                        QFile << quad.Quad_print();          //Print the quad into Quad
file.
                        Quad_add(quad);

                        break;
                    }
                    }

                    topCol = operatorN;          //New top operator on the stack.

                }
                else        //Error
                {
                    /*String variables*/
                    string opString;
                    string topString = intToClass(topCol);

                    //No precedence relation = 'X' in the table.
                    cout << "Error: Found '" << token.TString << "' after " << topString <<
", but expected ";

                    for (int i = 0; i < operatorKnt; i++)
                    {
                        if (PTable[topCol][i] != 'X' && PTable[topCol][i] != 'x')
                        {
                            opString = intToClass(i);

                            cout << opString << ", ";

                            errors = true;
                        }
                        else if (PTable[topCol][i] == 'X' || PTable[topCol][i] == 'x')
                        {
                            errors = false;
                        }
                    }

                    token = errorRecov(token, topCol, choice);
                }
            }
        }
    }

    //Missing a bracket.
    if (Stack_Brack.size() > 0) //Error
    {
        cout << "Bracket is missing its pair.";
        cout << endl;

        errors = true;      //Is error.
    }
    else    //Good
    {
        errors = false;
    }

    if (errors)         //Errors
    {
        cout << "Error: Parse was completed, but errors were detected.";
        cout << endl;
    }
    else    //No errors
    {
        scanner->buildSTable();
        Quad_EOF = { "EOF", "?", "?", "?" };        //EOF quad.
        mainQ.push_back(Quad_EOF);      //Adds Quad_EOF to the end of the procedureQ container.

        if (procedureQ.size() > 0)
        {
            mainQ.splice(mainQ.end(), procedureQ);
        }

        if (!varCheck(mainQ, scanner->SymTable))
        {
            if (assignCheck(mainQ, scanner->SymTable))
            {
                optimizeQuads optimize(mainQ);
                mainQ = optimize.optimAssign();

                CodeGen gen(mainQ, scanner->SymTable);
                gen.genCode();
            }
            else
            {
                cout << "Error: The reassignment attempt(s) of 'CONST' need(s) to be resolved
before the code can be generated.";
                cout << endl;
            }
        }
        else
        {
            cout << "Error: The undeclared variable(s) need(s) to be resolved before the code can be
generated.";
            cout << endl;
        }
    }

    QFile.close();  //Close Quad file.
}

/*
Function: classToInt
Parameters: (string str)
Description: Converts token classification
to a numeric value.
*/
int Parser::classToInt(string str)
{
    if (str == VARNAME || str == INTEGER || str == END_OF_FILE || str == TEMP)
    {
        return -1;
    }
    else if (str == SEMICOLON)
    {
        return 0;
    }
    else if (str == ASSIGN)
    {
        return 1;
    }
    else if (str == ADDOP)
    {
        return 2;
    }
    else if (str == LPAREN)
    {
        return 3;
    }
    else if (str == RPAREN)
    {
```

```
		return 4;
	}
	else if (str == MOP)
	{
		return 5;
	}
	else if (str == IF)
	{
		return 6;
	}
	else if (str == THEN)
	{
		return 7;
	}
	else if (str == ODD)
	{
		return 8;
	}
	else if (str == RELOP)
	{
		return 9;
	}
	else if (str == LBRACK || str == PROC_LBRACK)
	{
		return 10;
	}
	else if (str == RBRACK || str == PROC_RBRACK)
	{
		return 11;
	}
	else if (str == CALL)
	{
		return 12;
	}
	else if (str == WHILE)
	{
		return 13;
	}
	else if (str == DO)
	{
		return 14;
	}
	else if (str == COMMA)
	{
		return 15;
	}
	else if (str == CLASS)
	{
		return 16;
	}
	else if (str == VAR)
	{
		return 17;
	}
```

```
        else if (str == CONST)
        {
                return 18;
        }
        else if (str == PROCEDURE)
        {
                return 19;
        }
        else if (str == READ)
        {
                return 20;
        }
        else if (str == WRITE)
        {
                return 21;
        }
}

/*
Function: isError
Parameters: (string oper, Quad& quad, queue <Token>& q)
Description: Checks for a missing operand and looks to
see if said operand is of a valid classification.
*/
void Parser::isError(string oper, Quad& quad, queue <Token>& q)
{
        if (q.front().TClass != INTEGER && q.front().TClass != VARNAME && q.front().TClass != TEMP)
        //Error
        {
                cout << "Error: Missing an operand after '" << oper << "'";
                cout << endl;

                quad.operand_2 = "ERROR";        //Prints "ERROR" in operand position.

                errors = true;       //Is error.
        }
        else
        {
                quad.operand_2 = q.front().TString;
                q.pop();
        }

        quad.operation = q.front().TString;
        q.pop();

        if (q.size() > 0)
        {
                if (q.front().TClass != INTEGER && q.front().TClass != VARNAME && q.front().TClass !=
TEMP)
                {
                        cout << "Error: Missing an operand after '" << oper << "'";
                        cout << endl;

                        quad.operand_1 = "ERROR";        //Prints "ERROR" in operand position.
```

```
            errors = true;      //Is error.
        }
        else
        {
            quad.operand_1 = q.front().TString;
            q.pop();
        }
    }
    else if (q.size() <= 0)
    {
        cout << "Error: Missing an operand before '" << oper << "'";
        cout << endl;

        quad.operand_1 = "ERROR";       //Prints "ERROR" in operand position.

        errors = true;      //Is error.
    }
}

/*
Function: Quad_add
Parameters: (Quad quad)
Description: Adds quad to appropriate container.
*/
void Parser::Quad_add(Quad quad)
{
    if (!isQuad)
    {
        mainQ.push_back(quad);  //Adds quad to the end of the mainQ container.
    }
    else if (isQuad)
    {
        procedureQ.push_back(quad);     //Adds quad to the end of the procedureQ container.
    }
}

/*
Function: varCheck
Parameters: (list <Quad> quads, LL* table)
Description: Checks the symbol table to make sure
all of the variables and literals, that are in the
quad, are present.
*/
bool Parser::varCheck(list <Quad> quads, LL* table)
{
    bool undeclaredVars = false;

    while (!quads.empty())
    {
        if (quads.front().operation != "WHILE" && quads.front().operation != "DO" &&
            quads.front().operation.at(0) != 'L' && quads.front().operation != "THEN")
        {
            if (quads.front().operand_1 != "?")
            {
                if (table->inLL(quads.front().operand_1))
                {
                    undeclaredVars = false;
                }
                else
                {
                    undeclaredVars = true;

                    cout << "The variable " << quads.front().operand_1 << " is 
undeclared.";
                    cout << endl;
                }
            }

            if (quads.front().operand_2 != "?")
            {
                if (table->inLL(quads.front().operand_2))
                {
                    undeclaredVars = false;
                }
                else
                {
                    undeclaredVars = true;

                    cout << "The variable " << quads.front().operand_2 << " is 
undeclared.";
                    cout << endl;
                }
            }

            if (quads.front().operand_3 != "?")
            {
                if (table->inLL(quads.front().operand_3))
                {
                    undeclaredVars = false;
                }
                else
                {
                    undeclaredVars = true;

                    cout << "The variable " << quads.front().operand_3 << " is 
undeclared.";
                    cout << endl;
                }
            }
        }

        quads.pop_front();
    }

    return undeclaredVars;
}
```

```
/*
Function: assignCheck
Parameters: (list <Quad> quads, LL* table)
Description: Checks to see if any of the
quads with an assignment operator try to
illegaly assign a value to a constant.
*/
bool Parser::assignCheck(list <Quad> quads, LL* table)
{
        string oper;
        bool assignCheck = true;

        while (!quads.empty())
        {
                oper = quads.front().operation;

                if (oper == "=")
                {
                        if (table->getClass(quads.front().operand_1) != CONST)
                        {
                                assignCheck = true;
                        }
                        else if (table->getClass(quads.front().operand_1) == CONST)
                        {
                                assignCheck = false;

                                cout << "Error: CONST variable on left side of assignment statement: " <<
quads.front().operand_1;
                                cout << endl;
                        }
                }

                quads.pop_front();
        }

        return assignCheck;
}

/*
Function: intToClass
Parameters: (int num)
Description: Returns the class associated
with the value of the respective token.
Used for displaying error messages.
*/
string Parser::intToClass(int num)
{
        switch (num)
        {
        case 0:                 // <semi>
        {
                return "';'";
                break;
        }
```

```
case 1:                 // <assign>
{
        return "'='";
        break;
}
case 2:                 // <addop>
{
        return "'+' or '-'";
        break;
}
case 3:                 // $LP
{
        return "'('";
        break;
}
case 4:                 // $RP
{
        return "')'";
        break;
}
case 5:                 // <mop>
{
        return "'*', '/'";
        break;
}
case 6:                 // $IF
{
        return "'IF'";
        break;
}
case 7:                 // $THEN
{
        return "'THEN'";
        break;
}
case 8:                 // $ODD
{
        return "'ODD'";
        break;
}
case 9:                 // <relop>
{
        return "'>', '<', '<=', '>=', or '!='";
        break;
}
case 10: // $LB
{
        return "'{'";
        break;
}
case 11: // $RB
{
        return "'}'";
        break;
}
case 12: // $CALL
{
        return "'CALL'";
        break;
}
case 13: // $WHILE
{
        return "'WHILE'";
        break;
}
case 14: //$DO
{
        return "'DO'";
        break;
}
case 15: // <comma>
{
        return "','";
        break;
}
case 16: // $CLASS
{
        return "'CLASS'";
        break;
}
case 17: // <$var>
{
        return "'VAR'";
        break;
}
case 18: // $CONST
{
        return "'CONST'";
        break;
}
case 19: // $PROCEDURE
{
        return "'PROCEDURE'";
        break;
}
case 20: // $READ
{
        return "'READ'";
        break;
}
case 21: // $WRITE
{
        return "'WRITE'";
        break;
}
}
}
```

```
/*
Function: errorRecov
Parameters: (Token current, int& topCol, int& choice)
Description: Skips to the next line of code until a
delimiter or EOF is reached. An error has occured and
code generation has been prvented.
*/
Token Parser::errorRecov(Token current, int& topCol, int& choice)
{
        errors = true;      //Is error.         Code will not generate.

        cout << "Next line...";
        cout << endl;

        while (current.TClass != SEMICOLON && current.TClass != END_OF_FILE && current.TClass !=
RBRACK
                && current.TClass != LBRACK && current.TClass != PROC_LBRACK)
        {
                current = scanner->buildToken();
        }

        while (t.top().TClass != SEMICOLON && t.size() != 1)
        {
                if (t.top().TClass == LBRACK || t.top().TClass == PROC_LBRACK)
                {
                        break;
                }
                t.pop();
        }

        topCol = classToInt(t.top().TClass);//New top of the stack
        choice = classToInt(current.TClass);        //Next token for input.

        return current;
}
```

---------------Driver.cpp----------------

```
/**********************************************
Name: Driver.cpp
Author: Christopher McDaniel
Date Started: 11 February 2020
Date Completed: 23 April 2020
Class: COSC 4316
Version: 1.1
Copyright: 2020
Description: This is the main driving body file that
gets the source code file and passes it to the
Parser body file.

            DISCLAIMER:
THIS PORTION "Driver.cpp" IS REQUIRED TO BE USED IN
CONJUNCTION WITH "Parser.h".
**********************************************/

/*Include Program header files*/
#include "Parser.h"

/*
Function: main
Parameters: (N/A)
Description: Asks the user for source
file input and passes it to the Parser.
*/
int main() {
	/*String variable*/
	string SFile;

	//Ask for user input.
	cout << "Enter the name of the input file: ";
	cin >> SFile;		//Get user input.

	Parser parser(SFile);		//Pass source code file to Parser.
	parser.sourceParse();		//Parse the source code file.
}
```

--------------------------------Additional Information--------------------------------

-----------------Precedence Table----------------

	0: SEMI	1: ASSIGN	2: ADDOP	3: LPAREN	4: RPAREN	5: MOP	6: IF	7: THEN	8: ODD	9: RELOP	10: LBRACK
0: SEMI	X	<	X	X	X	X	<	X	X	X	<
1: ASSIGN	>	X	<	<	X	<	X	X	X	X	X
2: ADDOP	>	X	>	<	>	<	X	>	X	>	X
3: LPAREN	X	X	<	<	=	<	X	X	X	X	X
4: RPAREN	>	X	>	X	>	>	X	>	X	X	>
5: MOP	>	X	>	<	>	>	X	>	X	>	X
6: IF	X	X	<	<	X	<	X	=	<	<	X
7: THEN	>	<	X	X	X	X	<	X	X	X	<
8: ODD	X	X	<	<	X	<	X	>	X	X	X
9: RELOP	X	X	<	<	X	<	X	>	X	X	X
10: LBRACK	<	<	X	X	X	X	<	X	X	X	<
11: RBRACK	X	X	X	X	X	X	X	X	X	X	X
12: CALL	>	X	X	=	X	X	X	X	X	X	X
13: WHILE	X	X	<	<	X	<	X	X	<	<	X
14: DO	>	<	X	X	X	X	<	X	X	X	<
15: COMMA	>	=	X	X	=	X	X	X	X	X	X
16: CLASS	X	X	X	X	X	X	X	X	X	X	>
17: VAR	>	X	X	X	X	X	X	X	X	X	X
18: CONST	X	=	X	X	X	X	X	X	X	X	X
19: PROCEDURE	X	X	X	=	X	X	X	X	X	X	>
20: READ	>	X	X	X	X	X	X	X	X	X	X
21: WRITE	>	X	X	X	X	X	X	X	X	X	X

	11: RBRACK	12: CALL	13: WHILE	14: DO	15: COMMA	16: CLASS	17: VAR	18: CONST	19: PROCEDURE	20: READ	21: WRITE
0: SEMI	>	<	<	X	X	<	<	<	<	<	<
1: ASSIGN	>	X	X	X	=	X	X	X	X	X	X
2: ADDOP	>	X	X	>	X	X	X	X	X	X	X
3: LPAREN	X	X	X	X	=	X	X	X	X	X	X
4: RPAREN	X	X	X	>	X	X	X	X	X	X	X
5: MOP	>	X	X	>	X	X	X	X	X	X	X
6: IF	X	X	X	X	X	X	X	X	X	X	X
7: THEN	X	<	<	X	X	X	X	X	X	<	<
8: ODD	X	X	X	>	X	X	X	X	X	X	X
9: RELOP	X	X	X	>	X	X	X	X	X	X	X
10: LBRACK	=	<	<	X	X	X	<	<	<	<	<
11: RBRACK	X	X	X	X	X	X	X	X	X	X	X
12: CALL	X	X	X	X	X	X	X	X	X	X	X
13: WHILE	X	X	X	=	X	X	X	X	X	X	X
14: DO	X	<	<	X	X	X	X	X	X	<	<
15: COMMA	X	X	X	X	=	X	X	X	X	X	X
16: CLASS	X	X	X	X	X	X	X	X	X	X	X
17: VAR	X	X	X	X	=	X	X	X	X	X	X
18: CONST	X	X	X	X	X	X	X	X	X	X	X
19: PROCEDUR	X	X	X	X	X	X	X	X	X	X	X
20: READ	X	X	X	X	X	X	X	X	X	X	X
21: WRITE	X	X	X	X	X	X	X	X	X	X	X

------------------Scanner DFSA----------------

	0: Letter	1: Digit	2: *	3: /	4: =	5: <	6: Space	7: {	8: }	9: +	10: -	11: ,	12: ;	13: EOF	14: >	15: (	16:)	17: !	18: Other
0: GET NEXT CHARACTER	5	3	2	7	11	14	0	17	18	19	21	23	24	25	27	30	31	32	1
1: ERROR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2: ASTERISK	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26	26
3: DIGIT	4	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
4: <INTEGER>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5: LETTER-DIGIT	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
6: <VAR>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7: SLASH	10	10	8	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
8: LEFT COMMENT (/*)	8	8	9	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9: RIGHT COMMENT (*/)	8	8	8	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
10: <MOP> (/)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11: EQUAL	12	12	12	12	13	12	12	12	12	12	12	12	12	12	12	12	12	12	12
12: <ASSIGN> (=)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13: <RELOP> (==)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14: LESS THAN	15	15	15	15	16	15	15	15	15	15	15	15	15	15	15	15	15	15	15
15: <RELOP> (<)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16: <RELOP> (<=)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17: $LB ({)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18: $RB (})	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19: ADD	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
20: <ADDOP> (+)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21: SUBTRACT	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22
22: <ADDOP> (-)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23: <COMMA> (,)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24: <SEMI> (;)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25: EOF (\0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26: <MOP> (*)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27: GREATER THAN	28	28	28	28	29	28	28	28	28	28	28	28	28	28	28	28	28	28	28
28: <RELOP> (>)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29: <RELOP> (>=)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30: $LP (( )	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31: $RP ( ))	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32: EXCLAMATION MARK	33	33	33	33	34	33	33	33	33	33	33	33	33	33	33	33	33	33	33
33: <NOT> (!)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34: <RELOP> (!=)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

-------------------C1 Quad File-----------------

```
CLASS, C1, ?, ?
{, ?, ?, ?
READ, a, ?, ?
READ, b, ?, ?
READ, c, ?, ?
READ, Bob, ?, ?
READ, Jane, ?, ?
+, Bob, Jane, T1
-, T1, lit10, T1
/, T1, lit2, T1
*, T1, lit4, T1
*, a, T1, T1
+, b, c, T2
/, T1, T2, T1
=, ans, T1, ?
WRITE, ans, ?, ?
}, ?, ?, ?
```

--------------------C1 Token List------------------

```
CLASS  $CLASS
C1     <var>
{      $LB
VAR    <$var>
ans    <var>
,      <comma>
a      <var>
,      <comma>
Bob    <var>
,      <comma>
Jane   <var>
,      <comma>
b      <var>
,      <comma>
c      <var>
,      <comma>
N      <var>
,      <comma>
fact   <var>
;      <semi>
READ   $READ
a      <var>
;      <semi>
READ   $READ
b      <var>
;      <semi>
READ   $READ
c      <var>
;      <semi>
READ   $READ
Bob    <var>
;      <semi>
READ   $READ
Jane   <var>
;      <semi>
ans    <var>
=      <assign>
a      <var>
*      <mop>
(      $LP
(      $LP
Bob    <var>
+      <addop>
Jane   <var>
-      <addop>
10     <integer>
)      $RP
/      <mop>
2      <integer>
*      <mop>
4      <integer>
)      $RP
/       <mop>
(       $LP
b       <var>
+       <addop>
c       <var>
)       $RP
;       <semi>
WRITE   $WRITE
ans     <var>
;       <semi>
}       $RB
```

-------------------C2 Quad File-----------------

```
CLASS, C2, ?, ?
{, ?, ?, ?
READ, a, ?, ?
READ, b, ?, ?
IF, ?, ?, ?
>, a, b, ?
THEN, L1, ?, ?
WRITE, a, ?, ?
L1, ?, ?, ?
IF, ?, ?, ?
>, b, a, ?
THEN, L2, ?, ?
WRITE, b, ?, ?
L2, ?, ?, ?
}, ?, ?, ?
```

-------------------C2 Token List-----------------

```
CLASS   $CLASS
C2      <var>
{       $LB
VAR     <$var>
a       <var>
,       <comma>
b       <var>
;       <semi>
READ    $READ
a       <var>
;       <semi>
READ    $READ
b       <var>
;       <semi>
IF      $IF
a       <var>
>       <relop>
b       <var>
THEN    $THEN
WRITE   $WRITE
a       <var>
;       <semi>
IF      $IF
b       <var>
>       <relop>
a       <var>
THEN    $THEN
WRITE   $WRITE
b       <var>
;       <semi>
}       $RB
```

--------------------C3 Quad File------------------

```
CLASS, C3, ?, ?
{, ?, ?, ?
READ, a, ?, ?
READ, b, ?, ?
READ, c, ?, ?
IF, ?, ?, ?
>, a, b, ?
THEN, L1, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
>, a, c, ?
THEN, L2, ?, ?
{, ?, ?, ?
WRITE, a, ?, ?
}, ?, ?, ?
L2, ?, ?, ?
}, ?, ?, ?
L1, ?, ?, ?
IF, ?, ?, ?
>, b, a, ?
THEN, L3, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
>, b, c, ?
THEN, L4, ?, ?
{, ?, ?, ?
WRITE, b, ?, ?
}, ?, ?, ?
L4, ?, ?, ?
}, ?, ?, ?
L3, ?, ?, ?
IF, ?, ?, ?
>, c, a, ?
THEN, L5, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
>, c, b, ?
THEN, L6, ?, ?
{, ?, ?, ?
WRITE, c, ?, ?
}, ?, ?, ?
L6, ?, ?, ?
}, ?, ?, ?
L5, ?, ?, ?
}, ?, ?, ?
```

--------------------C3 Token List------------------

```
CLASS  $CLASS
C3     <var>
{      $LB
VAR    <$var>
a      <var>
,      <comma>
b      <var>
,      <comma>
c      <var>
;      <semi>
READ   $READ
a      <var>
;      <semi>
READ   $READ
b      <var>
;      <semi>
READ   $READ
c      <var>
;      <semi>
IF     $IF
a      <var>
>      <relop>
b      <var>
THEN   $THEN
{      $LB
IF     $IF
a      <var>
>      <relop>
c      <var>
THEN   $THEN
{      $LB
WRITE  $WRITE
a      <var>
;      <semi>
}      $RB
}      $RB
IF     $IF
b      <var>
>      <relop>
a      <var>
THEN   $THEN
{      $LB
IF     $IF
b      <var>
>       <relop>
c       <var>
THEN    $THEN
{       $LB
WRITE   $WRITE
b       <var>
;       <semi>
}       $RB
}       $RB
IF      $IF
c       <var>
>       <relop>
a       <var>
THEN    $THEN
{       $LB
IF      $IF
c       <var>
>       <relop>
b       <var>
THEN    $THEN
{       $LB
WRITE   $WRITE
c       <var>
;       <semi>
}       $RB
}       $RB
}       $RB
```

-------------------C4 Quad File-----------------

```
CLASS, C4, ?, ?
{, ?, ?, ?
=, fact, lit1, ?
READ, N, ?, ?
WHILE, W1, ?, ?
>, N, lit1, ?
DO, L1, ?, ?
{, ?, ?, ?
*, fact, N, T1
=, fact, T1, ?
-, N, lit1, T1
=, N, T1, ?
}, ?, ?, ?
W1, ?, ?, ?
L1, ?, ?, ?
WRITE, fact, ?, ?
}, ?, ?, ?
```

-------------------C4 Token List-----------------

```
CLASS   $CLASS
C4      <var>
{       $LB
VAR     <$var>
N       <var>
,       <comma>
fact    <var>
;       <semi>
fact    <var>
=       <assign>
1       <integer>
;       <semi>
READ    $READ
N       <var>
;       <semi>
WHILE   $WHILE
N       <var>
>       <relop>
1       <integer>
DO      $DO
{       $LB
fact    <var>
=       <assign>
fact    <var>
*       <mop>
N       <var>
;       <semi>
N       <var>
=       <assign>
N       <var>
-       <addop>
1       <integer>
;       <semi>
}       $RB
WRITE   $WRITE
fact    <var>
;       <semi>
}       $RB
```

-------------------B5 Quad File-----------------

```
CLASS, B5, ?, ?
{, ?, ?, ?
=, fact, lit1, ?
=, knt, lit0, ?
READ, M, ?, ?
WHILE, W1, ?, ?
<, knt, M, ?
DO, L1, ?, ?
{, ?, ?, ?
READ, N, ?, ?
WHILE, W2, ?, ?
>, N, lit1, ?
DO, L2, ?, ?
{, ?, ?, ?
*, fact, N, T1
=, fact, T1, ?
-, N, lit1, T1
=, N, T1, ?
}, ?, ?, ?
W2, ?, ?, ?
L2, ?, ?, ?
WRITE, fact, ?, ?
=, fact, lit1, ?
+, knt, lit1, T1
=, knt, T1, ?
}, ?, ?, ?
W1, ?, ?, ?
L1, ?, ?, ?
}, ?, ?, ?
```

-------------------B5 Token List-----------------

```
CLASS  $CLASS
B5     <var>
{      $LB
VAR    <$var>
M      <var>
,      <comma>
fact   <var>
,      <comma>
knt    <var>
,      <comma>
N      <var>
;      <semi>
fact   <var>
=      <assign>
1      <integer>
;      <semi>
knt    <var>
=      <assign>
0      <integer>
;      <semi>
READ   $READ
M      <var>
;      <semi>
WHILE  $WHILE
knt    <var>
<      <relop>
M      <var>
DO     $DO
{      $LB
READ   $READ
N      <var>
;      <semi>
WHILE  $WHILE
N      <var>
>      <relop>
1      <integer>
DO     $DO
{      $LB
fact    <var>
=       <assign>
fact    <var>
*       <mop>
N       <var>
;       <semi>
N       <var>
=       <assign>
N       <var>
-       <addop>
1       <integer>
;       <semi>
}       $RB
WRITE   $WRITE
fact    <var>
;       <semi>
fact    <var>
=       <assign>
1       <integer>
;       <semi>
knt     <var>
=       <assign>
knt     <var>
+       <addop>
1       <integer>
;       <semi>
}       $RB
}       $RB
```

-------------------B6 Quad File-----------------

```
CLASS, B6, ?, ?
{, ?, ?, ?
READ, N, ?, ?
=, fact, lit1, ?
PROCEDURE, factorial, ?, ?
{, ?, ?, ?
WHILE, W1, ?, ?
>, N, lit1, ?
DO, L1, ?, ?
{, ?, ?, ?
*, fact, N, T1
=, fact, T1, ?
-, N, lit1, T1
=, N, T1, ?
}, ?, ?, ?
W1, ?, ?, ?
L1, ?, ?, ?
}, ?, ?, ?
CALL, factorial, ?, ?
WRITE, fact, ?, ?
}, ?, ?, ?
```

-------------------B6 Token List-----------------

```
CLASS     $CLASS
B6        <var>
{         $LB
VAR       <$var>
N         <var>
,         <comma>
fact      <var>
;         <semi>
READ      $READ
N         <var>
;         <semi>
fact      <var>
=         <assign>
1         <integer>
;         <semi>
PROCEDURE     $PROCEDURE
factorial <var>
(         $LP
)         $RP
{         $LB
WHILE     $WHILE
N         <var>
>         <relop>
1         <integer>
DO        $DO
{         $LB
fact      <var>
=         <assign>
fact      <var>
*         <mop>
N         <var>
;         <semi>
N         <var>
=         <assign>
N         <var>
-         <addop>
1         <integer>
;         <semi>
}         $RB
}         $RB
CALL      $CALL
factorial <var>
(         $LP
)         $RP
;         <semi>
WRITE     $WRITE
fact      <var>
;         <semi>
}           $RB
```

--------------------A7 Quad File-----------------

```
CLASS, A7, ?, ?
{, ?, ?, ?
READ, N, ?, ?
=, fact, lit1, ?
PROCEDURE, RecursiveFactorial, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
!=, N, lit1, ?
THEN, L1, ?, ?
{, ?, ?, ?
*, fact, N, T1
=, fact, T1, ?
-, N, lit1, T1
=, N, T1, ?
CALL, RecursiveFactorial, ?, ?
}, ?, ?, ?
L1, ?, ?, ?
}, ?, ?, ?
CALL, RecursiveFactorial, ?, ?
WRITE, fact, ?, ?
}, ?, ?, ?
```

--------------------A7 Token List-----------------

```
CLASS       $CLASS
A7          <var>
{           $LB
VAR         <$var>
N           <var>
,           <comma>
fact        <var>
;           <semi>
READ        $READ
N           <var>
;           <semi>
fact        <var>
=           <assign>
1           <integer>
;           <semi>
PROCEDURE   $PROCEDURE
RecursiveFactorial          <var>
(           $LP
)           $RP
{       $LB
IF      $IF
N       <var>
!=      <relop>
1       <integer>
THEN    $THEN
{       $LB
fact    <var>
=       <assign>
fact    <var>
*       <mop>
N       <var>
;       <semi>
N       <var>
=       <assign>
N       <var>
-       <addop>
1       <integer>
;       <semi>
CALL    $CALL
RecursiveFactorial      <var>
(       $LP
)       $RP
;       <semi>
}       $RB
}       $RB
CALL    $CALL
RecursiveFactorial      <var>
(       $LP
)       $RP
;       <semi>
WRITE   $WRITE
fact    <var>
;       <semi>
}       $RB
```

-------------------A8 Quad File-----------------

```
CLASS, LCD, ?, ?
{, ?, ?, ?
PROCEDURE, Multiply, ?, ?
{, ?, ?, ?
=, A, X, ?
=, B, Y, ?
=, Z, lit0, ?
WHILE, W1, ?, ?
>, B, lit0, ?
DO, L1, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
ODD, B, ?, ?
THEN, L2, ?, ?
+, Z, A, T1
```

```
=, Z, T1, ?
L2, ?, ?, ?
*, lit2, A, T1
=, A, T1, ?
/, B, lit2, T1
=, B, T1, ?
}, ?, ?, ?
W1, ?, ?, ?
L1, ?, ?, ?
}, ?, ?, ?
PROCEDURE, Divide, ?, ?
{, ?, ?, ?
=, R, X, ?
=, Q, lit0, ?
=, W, Y, ?
WHILE, W2, ?, ?
<=, W, R, ?
DO, L3, ?, ?
*, lit2, W, T1
=, W, T1, ?
W2, ?, ?, ?
L3, ?, ?, ?
WHILE, W3, ?, ?
>, W, Y, ?
DO, L4, ?, ?
{, ?, ?, ?
*, lit2, Q, T1
=, Q, T1, ?
/, W, lit2, T1
=, W, T1, ?
IF, ?, ?, ?
<=, W, R, ?
THEN, L5, ?, ?
{, ?, ?, ?
-, R, W, T1
=, R, T1, ?
+, Q, lit1, T1
=, Q, T1, ?
}, ?, ?, ?
L5, ?, ?, ?
}, ?, ?, ?
W3, ?, ?, ?
L4, ?, ?, ?
}, ?, ?, ?
PROCEDURE, GCD, ?, ?
{, ?, ?, ?
=, F, X, ?
=, G, Y, ?
WHILE, W4, ?, ?
!=, F, G, ?
DO, L6, ?, ?
{, ?, ?, ?
IF, ?, ?, ?
<, F, G, ?
```

```
THEN, L7, ?, ?
-, G, F, T1
=, G, T1, ?
L7, ?, ?, ?
IF, ?, ?, ?
<, G, F, ?
THEN, L8, ?, ?
-, F, G, T1
=, F, T1, ?
L8, ?, ?, ?
}, ?, ?, ?
W4, ?, ?, ?
L6, ?, ?, ?
=, Z, F, ?
}, ?, ?, ?
=, X, M, ?
=, Y, N, ?
CALL, Multiply, ?, ?
=, X, lit25, ?
=, Y, lit3, ?
CALL, Divide, ?, ?
=, X, lit84, ?
=, Y, lit36, ?
CALL, GCD, ?, ?
WRITE, Z, ?, ?
}, ?, ?, ?
```

--------------------A8 Token List------------------

```
CLASS  $CLASS
LCD    <var>
{      $LB
CONST  $CONST
M      <var>
=      <assign>
7      <integer>
,      <comma>
N      <var>
=      <assign>
85     <integer>
;      <semi>
VAR    <$var>
X      <var>
,      <comma>
Y      <var>
,      <comma>
Z      <var>
,      <comma>
Q      <var>
,      <comma>
R      <var>
;      <semi>
PROCEDURE     $PROCEDURE
```

Multiply	<var>
(	$LP
)	$RP
{	$LB
VAR	<$var>
A	<var>
,	<comma>
B	<var>
;	<semi>
A	<var>
=	<assign>
X	<var>
;	<semi>
B	<var>
=	<assign>
Y	<var>
;	<semi>
Z	<var>
=	<assign>
0	<integer>
;	<semi>
WHILE	$WHILE
B	<var>
>	<relop>
0	<integer>
DO	$DO
{	$LB
IF	$IF
ODD	$ODD
B	<var>
THEN	$THEN
Z	<var>
=	<assign>
Z	<var>
+	<addop>
A	<var>
;	<semi>
A	<var>
=	<assign>
2	<integer>
*	<mop>
A	<var>
;	<semi>
B	<var>
=	<assign>
B	<var>
/	<mop>
2	<integer>
;	<semi>
}	$RB
}	$RB
PROCEDURE	$PROCEDURE
Divide	<var>
(	$LP

)	$RP
{	$LB
VAR	<$var>
W	<var>
;	<semi>
R	<var>
=	<assign>
X	<var>
;	<semi>
Q	<var>
=	<assign>
0	<integer>
;	<semi>
W	<var>
=	<assign>
Y	<var>
;	<semi>
WHILE	$WHILE
W	<var>
<=	<relop>
R	<var>
DO	$DO
W	<var>
=	<assign>
2	<integer>
*	<mop>
W	<var>
;	<semi>
WHILE	$WHILE
W	<var>
>	<relop>
Y	<var>
DO	$DO
{	$LB
Q	<var>
=	<assign>
2	<integer>
*	<mop>
Q	<var>
;	<semi>
W	<var>
=	<assign>
W	<var>
/	<mop>
2	<integer>
;	<semi>
IF	$IF
W	<var>
<=	<relop>
R	<var>
THEN	$THEN
{	$LB
R	<var>
=	<assign>

```
R          <var>
-          <addop>
W          <var>
;          <semi>
Q          <var>
=          <assign>
Q          <var>
+          <addop>
1          <integer>
;          <semi>
}          $RB
}          $RB
}          $RB
PROCEDURE    $PROCEDURE
GCD        <var>
(          $LP
)          $RP
{          $LB
VAR        <$var>
F          <var>
,          <comma>
G          <var>
;          <semi>
F          <var>
=          <assign>
X          <var>
;          <semi>
G          <var>
=          <assign>
Y          <var>
;          <semi>
WHILE      $WHILE
F          <var>
!=         <relop>
G          <var>
DO         $DO
{          $LB
IF         $IF
F          <var>
<          <relop>
G          <var>
THEN       $THEN
G          <var>
=          <assign>
G          <var>
-          <addop>
F          <var>
;          <semi>
IF         $IF
G          <var>
<          <relop>
F          <var>
THEN       $THEN
F          <var>
```

=	<assign>
F	<var>
-	<addop>
G	<var>
;	<semi>
}	$RB
Z	<var>
=	<assign>
F	<var>
;	<semi>
}	$RB
X	<var>
=	<assign>
M	<var>
;	<semi>
Y	<var>
=	<assign>
N	<var>
;	<semi>
CALL	$CALL
Multiply	<var>
(	$LP
)	$RP
;	<semi>
X	<var>
=	<assign>
25	<integer>
;	<semi>
Y	<var>
=	<assign>
3	<integer>
;	<semi>
CALL	$CALL
Divide	<var>
(	$LP
)	$RP
;	<semi>
X	<var>
=	<assign>
84	<integer>
;	<semi>
Y	<var>
=	<assign>
36	<integer>
;	<semi>
CALL	$CALL
GCD	<var>
(	$LP
)	$RP
;	<semi>
WRITE	$WRITE
Z	<var>
;	<semi>
}	$RB

-------------------A8(2) Quad File-----------------

```
CLASS, UndeclaredVar, ?, ?
{, ?, ?, ?
=, A, lit1, ?
=, B, lit2, ?
+, A, B, T1
=, D, T1, ?
}, ?, ?, ?
```

-------------------A8(2) Token List-----------------

```
CLASS  $CLASS
UndeclaredVar    <var>
{        $LB
VAR      <$var>
A        <var>
,        <comma>
B        <var>
,        <comma>
C        <var>
;        <semi>
A        <var>
=        <assign>
1        <integer>
;        <semi>
B        <var>
=        <assign>
2        <integer>
;        <semi>
D        <var>
=        <assign>
A        <var>
+        <addop>
B        <var>
;        <semi>
}        $RB
```