Computer Organization Project

Table Of Contents

A high-level data flow diagram for the system

1.Intially the abm instructions containing file is fed into the program as an argument

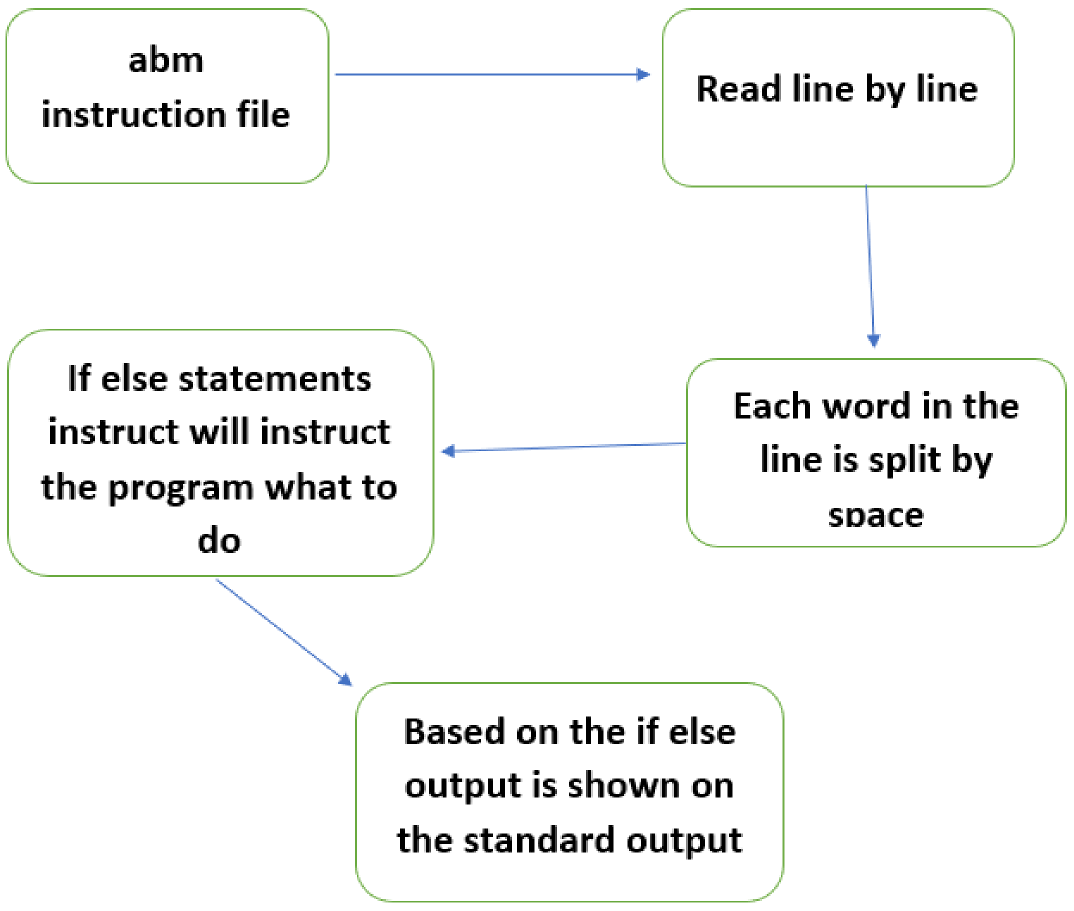
Ex: ./abmsimulator operatorsTest.abm

2.Then the simulator program of mine will read line by line from the abm instruction file.

3.Then each word in the file is split from the spaces and put into a 2d array.

4.Then each of the first words of one specific line is checked by the program. I have used the if statements to instruct what needs to be done incase of a specific instruction.

5.Based on the instructions inside the if else statements the outputs are shown on the standard output to the user.


abm instruction file
Read line by line
Each word in the line is split by space
If else statements instruct will instruct the program what to do
Based on the if else output is shown on the standard output

List Of Routines

A list of routines and their brief descriptions

1.read:

The read() function reads previously written data from a file. If any part of a regular file that appears before the end of the file has not been written, read() will return bytes with a value of 0.

2.fopen:

The fopen() method is used to open a file and execute different operations such as reading, writing, and so on, as well as to switch between modes. If successful, fopen() returns a pointer to the object handling the connected stream. If it fails, it returns a Null pointer

3.fclose:

To close a file, use the fclose() method. If the stream is successfully closed, fclose() returns 0, else it returns EOF.

4.fgets :

The C library function char *fgets(char *str, int n, FILE *stream) reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

5.push :

push() function is used to insert or 'push' an element at the top of the stack.The element is added to the stack container and the size of the stack is increased by 1.The value of the element to be inserted is passed as the parameter.Adds an element of value the same as that of the parameter passed at the top of the stack.

6.pop :

The pop() function is used to remove or ‘pop’ an element from the top of the stack(newest or the topmost element in the stack). The element is removed from the stack container and the size of the stack is decreased by 1.No parameters are passed.Removes the newest element in the stack or basically the top element.

7.peek :

Peek() is one of a stack operation that returns the value of the top most element of the stack without deleting that element from the stack.

8.isFull :

Checks whether the stack is full.Stack is full when top is equal to the last index

9.isEmpty :

Checks whether the stack is empty or not.Stack is empty when top is equal to -1

Implementation Details

- First I created a struct data structure as the stack.
- Then I implemented functions to push , pop and peek data to and from the stack.
- I have initialized a 2d array to store words.
- And then using fopen() command the prog reads the file line by line when the file name is given as an argument.
- From this file each line is read inside a while loop using the fgets command.
- All the words in this line are split from the space and stored in the 2d array.
- Using a loop the first word of each instruction line is chosen.
- Based on the instruction type if else statements are used inorder to instruct the program what needs to be done.
- Instructions for the arithmetic operators were handled by simply peeking the last two values on the top of the stack and then after performing the arithmetic operator the result was pushed into the top of the stack.
- Instructions for the logical and relational operators were also handled by simply peeking the last two values on the top of the stack and then after performing the logical and relational operation the result was pushed into the top of the stack.
- In handling the stack manipulation commands such as lvalue , i declared a global variable. The address of this global variable was pushed into the stack. And incase of rvalue the content in the global variable that was set by lvalue was pushed into the top of the stack.
- In Case of := instruction , the both the top values of the stack are first popped out. Then the second value assigned to the top value and then this top value is once again pushed onto the stack.

- When the if condition detects a goto , gofalse , gotrue and call instruction. The current iteration number is saved in a variable called the program counter. And then using a loop a new file pointer traverses the input file to detect any instruction called label. If it detects a label instruction i, then that specific line number is noted in a variable start and the end of the label is detected by the instruction return. This line number is stored in variable end. In case of a jump instruction the program saves the current instruction line in the program counter and jumps to the start instruction line which is the start of the label and executes the instructions one by one until the end instruction and returns back to the initial place by using the instruction line number in the PC.

- In case of begin and end instructions the instruction executes in the normal way but when rvalue and lvalue are used inside the begin then rather than using the global variables i used temporary variables in which i handled the scope of the variables initialized and called inside the begin-end and labels with the variables initialized and called outside.

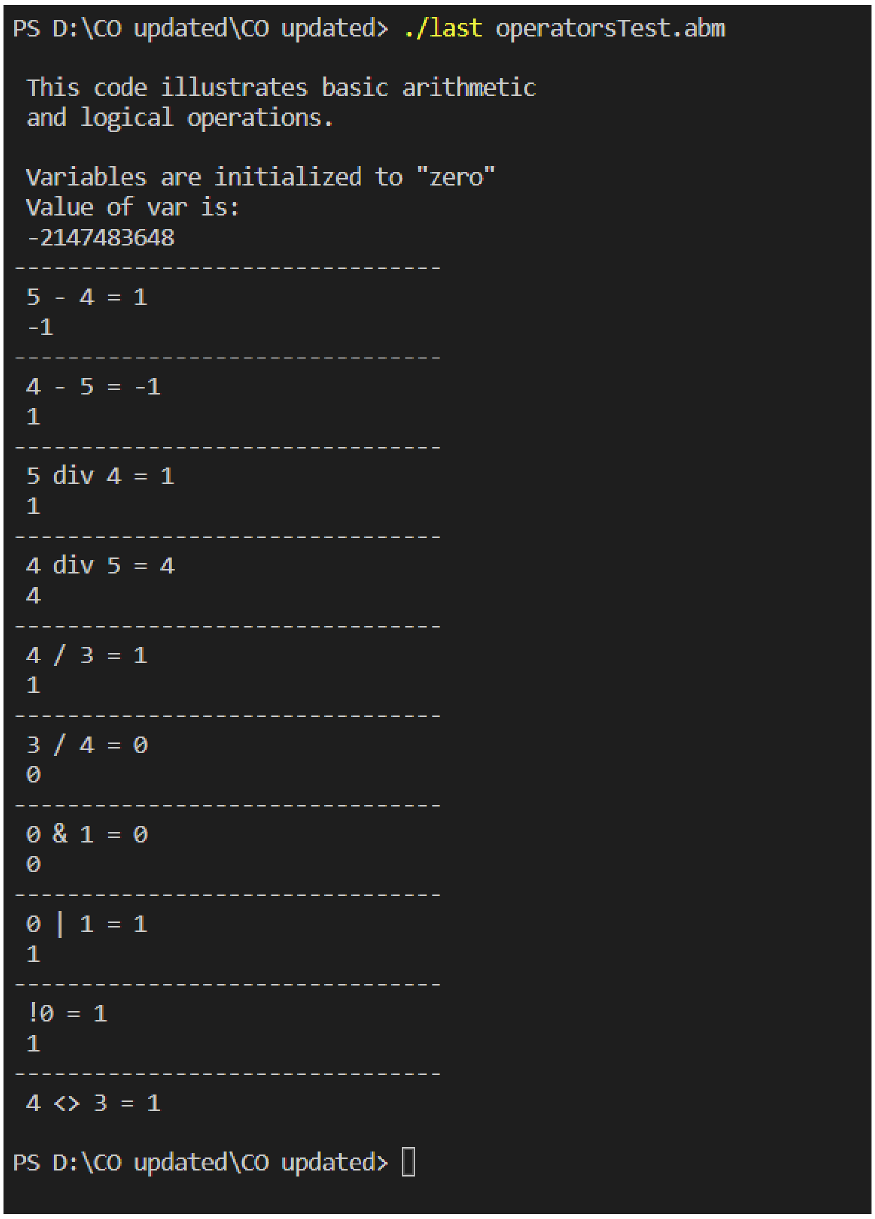
Testing

Test documentation
- Data used for all tests done.

Inorder to test the program the files that were given as reference was used.

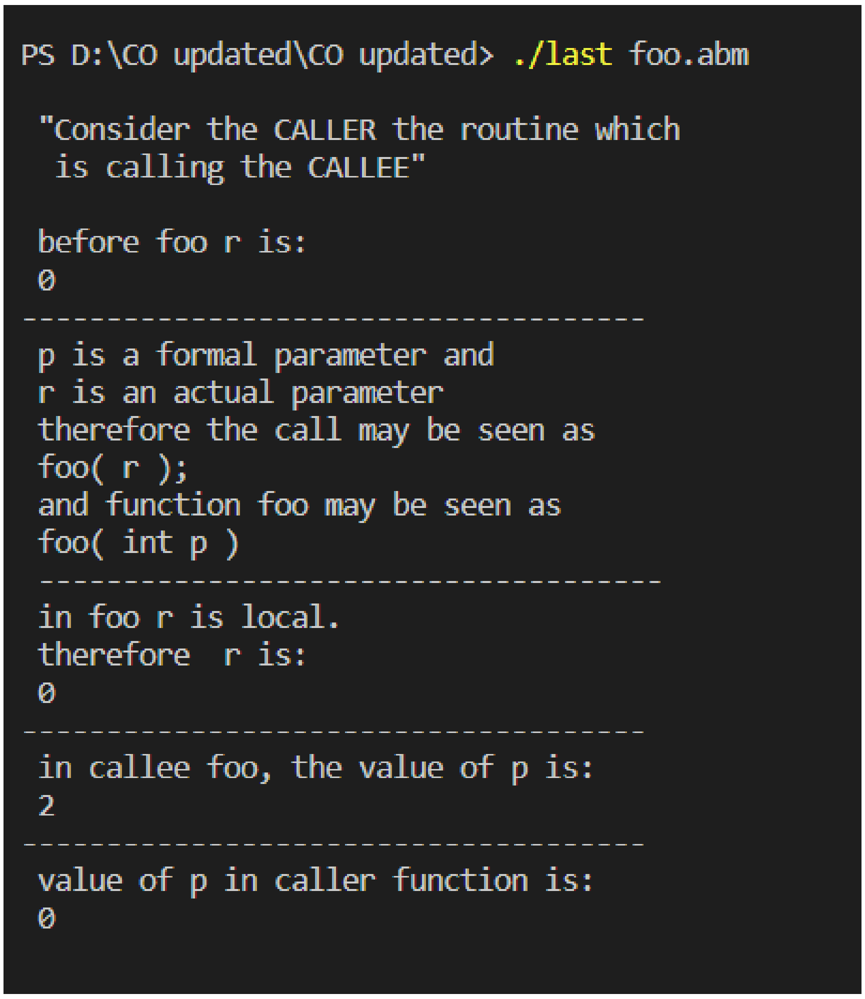
- operatorsTest.abm
- foo.abm
- factProc.abm
- recursiveFactorial.abm
- demo.abm

Test 01

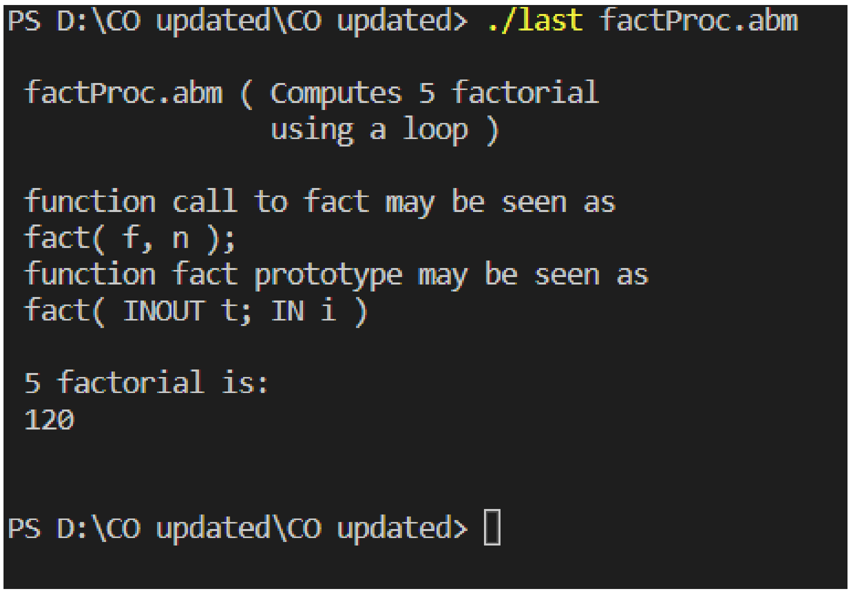


```
PS D:\CO updated\CO updated> ./last operatorsTest.abm

 This code illustrates basic arithmetic
 and logical operations.

 Variables are initialized to "zero"
 Value of var is:
 -2147483648
--------------------------------
 5 - 4 = 1
 -1
--------------------------------
 4 - 5 = -1
 1
--------------------------------
 5 div 4 = 1
 1
--------------------------------
 4 div 5 = 4
 4
--------------------------------
 4 / 3 = 1
 1
--------------------------------
 3 / 4 = 0
 0
--------------------------------
 0 & 1 = 0
 0
--------------------------------
 0 | 1 = 1
 1
--------------------------------
 !0 = 1
 1
--------------------------------
 4 <> 3 = 1

PS D:\CO updated\CO updated> 
```

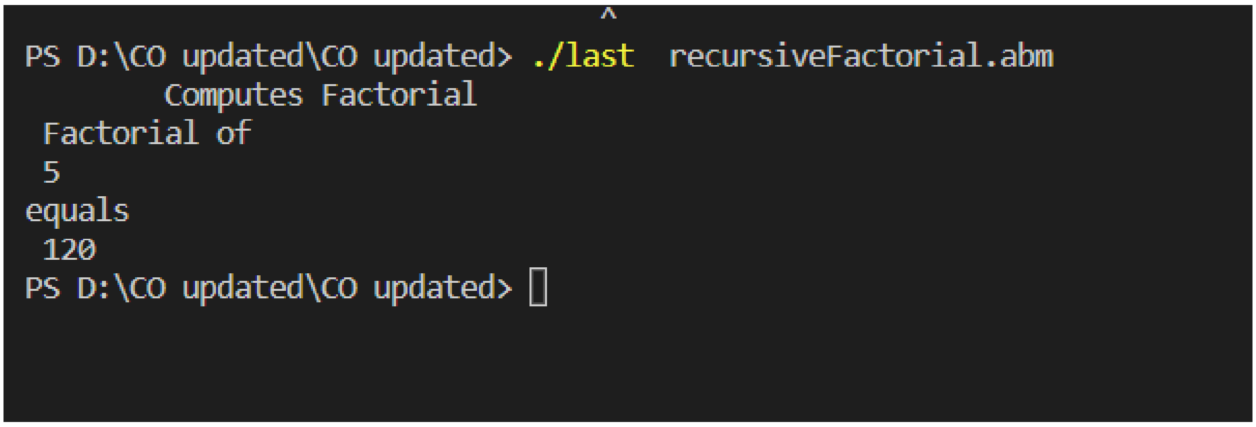
Test 02


```
PS D:\CO updated\CO updated> ./last foo.abm

 "Consider the CALLER the routine which
  is calling the CALLEE"

 before foo r is:
 0
-------------------------------------
 p is a formal parameter and
 r is an actual parameter
 therefore the call may be seen as
 foo( r );
 and function foo may be seen as
 foo( int p )
 -------------------------------------
 in foo r is local.
 therefore  r is:
 0
-------------------------------------
 in callee foo, the value of p is:
 2
-------------------------------------
 value of p in caller function is:
 0
```

Test 03

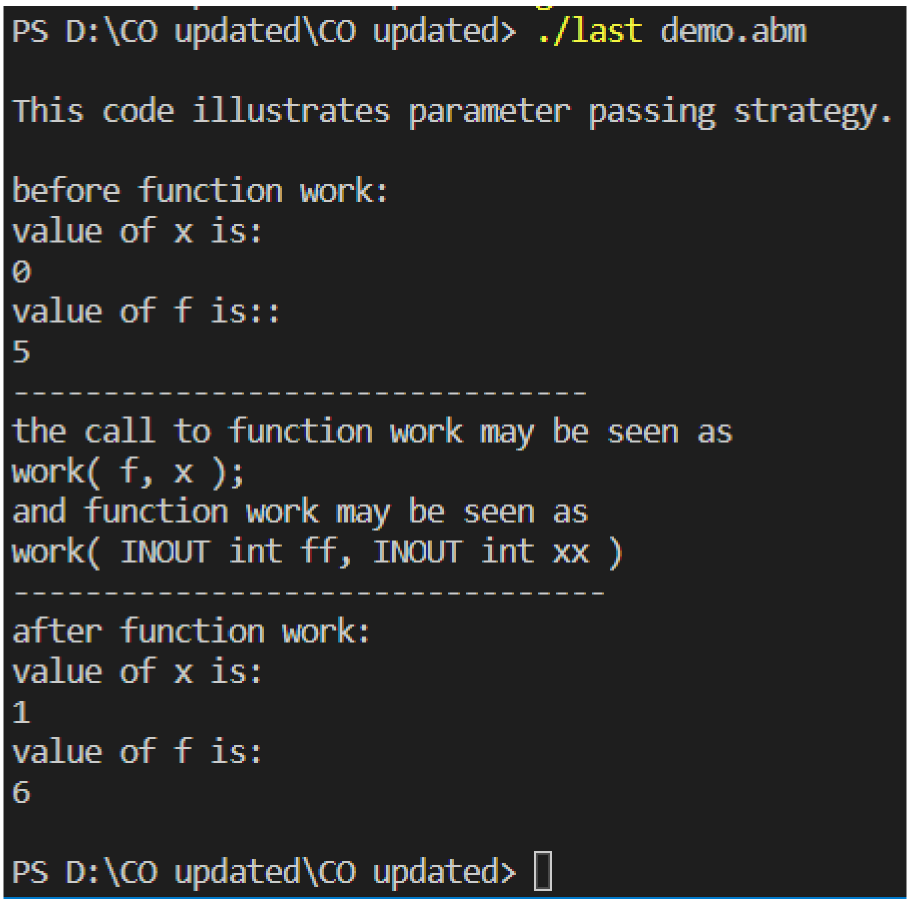

```
PS D:\CO updated\CO updated> ./last factProc.abm

 factProc.abm ( Computes 5 factorial
               using a loop )

 function call to fact may be seen as
 fact( f, n );
 function fact prototype may be seen as
 fact( INOUT t; IN i )

 5 factorial is:
 120


PS D:\CO updated\CO updated> 
```

Test 04



```
PS D:\CO updated\CO updated> ./last  recursiveFactorial.abm
        Computes Factorial
 Factorial of
 5
equals
 120
PS D:\CO updated\CO updated> 
```

Test 05



```
PS D:\CO updated\CO updated> ./last demo.abm

This code illustrates parameter passing strategy.

before function work:
value of x is:
0
value of f is::
5
--------------------------------
the call to function work may be seen as
work( f, x );
and function work may be seen as
work( INOUT int ff, INOUT int xx )
---------------------------------
after function work:
value of x is:
1
value of f is:
6

PS D:\CO updated\CO updated> 
```

i. List of program bugs and your plans to address them.

Here the main problem that I encountered was, in linux when implementing the =: instruction the address of a variable must be popped from the stack and to that address some data must be stored and the address must be written back to the stack. Here my stack was only capable of storing integers. But an address is a pointer variable. I had to typecast this pointer variable to an integer and then store it.
This showed a warning when compiling in Linux. Therefore I had to compile the program in windows using a gcc compiler. Then it allowed me to compile it. Then my program worked. Instructions to compile and run are provided under the user documentation section.

iii. Difficulties and solutions involved in creating the program.

It was difficult to handle when the program had multiple labels, and Begin and End instructions. As a solution for that, we have used a program counter to track the execution order of the program. So when the program reaches a jump instruction it goes to the specific label instruction and executes all the instruction lines that belong to the label and finally, it returns from the subroutine to the place where the jump instruction is called. After that, the rest of the code lines are executed.

Algorithms and data structures.

1.Algorithms.

Does your system use any complex algorithm?

No. We used a simple and procedural approach to code the solution.

ii.Data structures.

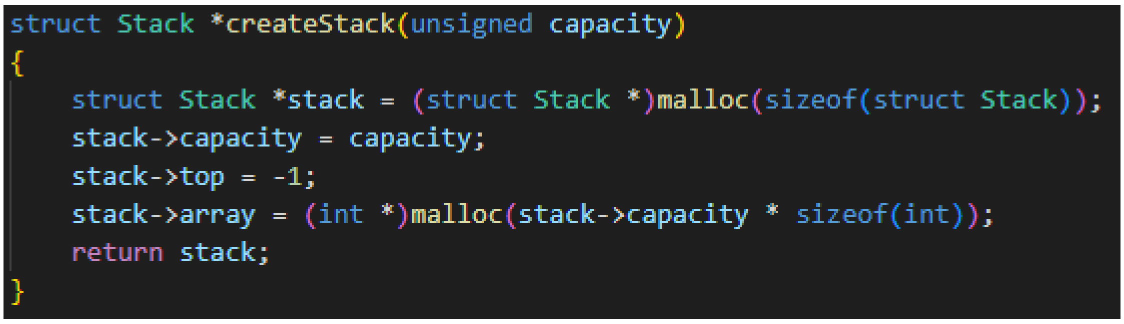
Does your system use any complex data structures such as arrays, linked lists, stacks, queues, graphs, hash tables, or trees?

Yes we used few complex data structures as follows,

Here the struct data structure is used in implementing the stack.

We chose the struct data structure because when implementing the stack multiple parameters were required. That is a variable to store the capacity , a variable to note the index and array to store the content. All these were belonging to one entity. Therefore a struct data structure was chosen.

When allocating space for the stack iv used a dynamic memory allocation using the malloc function.



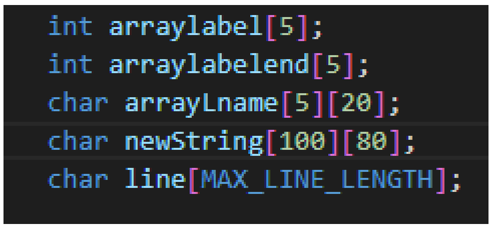
```
struct Stack *createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int *)malloc(stack->capacity * sizeof(int));
    return stack;
}
```

And functions such as pop , push , peek , isFull and isEmpty are implemented to manipulate the stack.

Also in order to store the words retrieved from the instruction file a 2d array is implemented.

The main reason for using a 2d array was we had to deal with lines and the words of a specific line. Lines were considered as the rows and the words can be considered as columns of a specific row.In Terms of performance and flexibility the usage of arrays were far more simple and easier and consumed less memory space when compared with a linked list implementation.

Also to store the label names and for some other purposes some additional arrays are used.



```
int arraylabel[5];
int arraylabelend[5];
char arrayLname[5][20];
char newString[100][80];
char line[MAX_LINE_LENGTH];
```

User Documentation

i. What operating system was used?

I used Windows 11 inorder to code the program. I had already installed the gcc compiler on my PC using MinGW. Therefore I used my Windows operating system itself to code rather than going for the Virtual machine since it was too heavy for my PC.

ii. How to compile your program?

Note : Sometimes the program might show warnings when compiling in linux operating system.
This is because I have handled the memory addresses in a different way.Therefore please be kind enough to compile the program in windows using a gcc compiler.

If your PC does not already contain a gcc compiler, then it must be installed to windows.(using MinGW)
(Reference : https://www.scaler.com/topics/c/c-compiler-for-windows/)

Next compile the program using

```
gcc -o abmsimulator abmsimulator.c
```

iii. What other applications (tools) does your system require?

A gcc compiler installed in Windows (gcc using MinGW)
(Reference : https://www.scaler.com/topics/c/c-compiler-for-windows/)

iv. How to run your program.

```
./abmsimulator filename.abm
```

Source Code

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LINE_LENGTH 10000
#define varr 6

struct Stack
{
   int top;
   unsigned capacity;
   int *array;
};

struct Stack *createStack(unsigned capacity)
{
   struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
   stack->capacity = capacity;
   stack->top = -1;
   stack->array = (int *)malloc(stack->capacity * sizeof(int));
   return stack;
}

int isFull(struct Stack *stack)
{
   return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack *stack)
{
   return stack->top == -1;
}

void push(struct Stack *stack, int item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
}

int pop(struct Stack *stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}
int peek(struct Stack *stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top];
}
int simulator(char newString[100][80], int p, int r, int var, int ctr, struct Stack *stack)
{

    if (newString[0][0] == 'p' && newString[0][1] == 'o' && newString[0][2] == 'p')
    {
        pop(stack);
    }

    else if (newString[0][0] == 's' && newString[0][1] == 'h' && newString[0][2] == 'o' &&
newString[0][3] == 'w')
    {
        for (int i = 1; i < ctr; i++)
        {
            printf("%s ", newString[i]);
        }
    }
    else if (newString[0][0] == 'p' && newString[0][1] == 'u' && newString[0][2] == 's' &&
newString[0][3] == 'h')
    {
        push(stack, atoi(newString[1]));
    }
```

```
  if (newString[0][0] == 'p' && newString[0][1] == 'r' && newString[0][2] == 'i' &&
newString[0][3] == 'n' && newString[0][4] == 't')
  {
    int val = peek(stack);
    printf("%d\n", val);
  }

  else if (newString[0][0] == 'd' && newString[0][1] == 'i' && newString[0][2] == 'v')
  {
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);

    int division = value2 % value1;
    push(stack, division);
  }
  else if (newString[0][0] == '-')
  {
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);

    int substraction = value1 - value2;
    push(stack, substraction);
  }
  else if (newString[0][0] == '+')
  {
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    int addition = value1 + value2;
    push(stack, addition);
  }
  else if (newString[0][0] == '/')
  {
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);

    int div = value2 / value1;
    push(stack, div);
}
else if (newString[0][0] == '<' && newString[0][1] == '>')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value1 == value2)
    {
        push(stack, 0);
    }
    else
    {
        push(stack, 1);
    }
}
else if (newString[0][0] == '<' && newString[0][1] == '=')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value2 <= value1)
    {
        push(stack, 1);
    }
    else
    {
        push(stack, 0);
    }
}
```

```
else if (newString[0][0] == '>' && newString[0][1] == '=')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value2 >= value1)
    {
        push(stack, 1);
    }
    else
    {
        push(stack, 0);
    }
}
else if (newString[0][0] == '<')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value2 < value1)
    {
        push(stack, 1);
    }
    else
    {
        push(stack, 0);
    }
}
else if (newString[0][0] == '>')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value2 > value1)
    {
        push(stack, 1);
    }
    else
    {
        push(stack, 0);
    }
}

else if (newString[0][0] == '=')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    if (value2 == value1)
    {
        push(stack, 1);
    }
    else
    {
        push(stack, 0);
    }
}

else if (newString[0][0] == '&')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
    push(stack, value2);
    int value3 = value2 && value1;
    push(stack, value3);
}
else if (newString[0][0] == '|')
{
    int value1 = pop(stack);
    int value2 = pop(stack);
    push(stack, value1);
```

```
        push(stack, value2);
        int value3 = value2 || value1;
        push(stack, value3);
    }
    else if (newString[0][0] == '!')
    {
        int value1 = pop(stack);
        push(stack, value1);
        push(stack, !value1);
    }
    else if (newString[0][0] == 'h' && newString[0][1] == 'a' && newString[0][2] == 'l' &&
newString[0][3] == 't')
    {
        return 0;
    }

    else if (newString[0][0] == 'c' && newString[0][1] == 'o' && newString[0][2] == 'p' &&
newString[0][3] == 'y')
    {
        int copy = peek(stack);
        push(stack, copy);
    }
}
int countspaces(char line[MAX_LINE_LENGTH])
{
    int spaces = 0;
    for (int i = 0; i <= (strlen(line)); i++)
    {
        if (line[0] == ' ')
        {
            spaces = 1;
        }
        if (line[0] == ' ' && line[1] == ' ')
        {
            spaces = 2;
        }
        if (line[0] == ' ' && line[1] == ' ' && line[2] == ' ')
        {
            spaces = 3;
        }
        if (line[0] == ' ' && line[1] == ' ' && line[2] == ' ' && line[3] == ' ')
        {
            spaces = 4;
        }
    }
    return spaces;
}
int main(int argc, char *argv[])
{
    FILE *filePointer;
    FILE *filePointer2;
    FILE *filePointer3;
    FILE *filePointer4;

    int globalcount = 0;
    int tempP = 0;
    int tempR = 0;
    int tempVar = 0;

    int j, ctr, count, i, d = 0, z = 5;
    int p = 0, r = 0, var = 0;
    int arraylabel[varr];
    int arraylabelend[varr];
    char arrayLname[varr][20];

    char newString[100][80];

    char line[MAX_LINE_LENGTH];

    struct Stack *stack = createStack(100);

    filePointer2 = fopen(argv[1], "r");

    if (NULL == filePointer2)
    {
        printf("file can't be opened \n");
        return 0;
    }
    if (argv[1][0] == 'd')
    {
        d = 1;
    }
    filePointer = fopen(argv[1], "r");

    if (NULL == filePointer)
    {
        printf("file can't be opened \n");
        return 0;
    }
    count = 0;
    int start, end, pCounter = 0, bstart, bend, bpCounter;
    while (fgets(line, MAX_LINE_LENGTH, filePointer2))
    {

        int spaces = countspaces(line);
        int order = 0;
        int t = 0;

        if (line[spaces] == 'l' && line[spaces + 1] == 'a' && line[spaces + 2] == 'b' &&
line[spaces + 3] == 'e' && line[spaces + 4] == 'l')
        {
            arraylabel[order] = count + 1;
            for (int s = 0; line[spaces + 6 + s] != '\0'; s++)
            {
                arrayLname[order][s] = line[spaces + 6 + s];
            }
            order++;

            if (line[spaces + 6] == 'f' && line[spaces + 7] == 'a' && line[spaces + 8] == 'c' &&
line[spaces + 9] == 't')
            {
                break;
            }
            if (line[spaces + 6] == 'b' && line[spaces + 7] == 'e')
            {
                break;
            }
        }

        if (line[spaces] == 'r' && line[spaces + 1] == 'e' && line[spaces + 2] == 't' &&
line[spaces + 3] == 'u' && line[spaces + 4] == 'r' && line[spaces + 5] == 'n')
        {
            arraylabelend[t] = count + 1;
            t++;
        }
        count++;
    }
    if (d == 1)
    {
        arraylabelend[0] = '0';
    }
    fclose(filePointer2);
    while (fgets(line, MAX_LINE_LENGTH, filePointer))
    {
        int begin = 0;
        int spaces = countspaces(line);
        j = 0;
        ctr = 0;
        for (i = spaces; i <= (strlen(line)); i++)
        {
            if ((line[i] == ' ' || line[i] == '\0'))
            {
                newString[ctr][j] = '\0';
                ctr++; // for next word
                j = 0; // for next word, init index to 0
            }
            else
            {
                newString[ctr][j] = line[i];
                j++;
            }
        }
        if (!(globalcount >= arraylabel[0] - 1 && globalcount <= arraylabelend[0]))
        {
            simulator(newString, p, r, var, ctr, stack);
            if (newString[0][0] == 'l' && newString[0][1] == 'v' && newString[0][2] == 'a' &&
newString[0][3] == 'l' && newString[0][4] == 'u' && newString[0][5] == 'e')
            {
                if (begin = 1)
                {
                    if (newString[1][0] == 'p')
                    {
                        int *address = &tempP;
                        int add = (int)address;
                        push(stack, add);
                    }
                    if (newString[1][0] == 'r')
                    {
                        int *address = &tempR;
                        int add = (int)address;
                        push(stack, add);
                    }

                    if (newString[1][0] == 'v' && newString[1][1] == 'a' && newString[1][2] == 'r')
                    {
                        int *address = &tempVar;
                        int add = (int)address;
                        push(stack, add);
                    }
                    else
                    {
                        int *address = &tempVar;
                        int add = (int)address;
                        push(stack, add);
                    }
                }
                else
                {
                    if (newString[1][0] == 'p')
                    {
```

```
                int *address = &p;
                int add = (int)address;
                push(stack, add);
            }

            if (newString[1][0] == 'r')
            {
                int *address = &r;
                int add = (int)address;
                push(stack, add);
            }

            if (newString[1][0] == 'v' && newString[1][1] == 'a' && newString[1][2] == 'r')
            {
                int *address = &var;
                int add = (int)address;
            }
            else
            {
                int *address = &var;
                int add = (int)address;
                //  push(stack, add);
            }
        }
        pCounter++;
    }
    else if (newString[0][0] == 'r' && newString[0][1] == 'v' && newString[0][2] == 'a'
&& newString[0][3] == 'l' && newString[0][4] == 'u' && newString[0][5] == 'e')
    {
        int ret = 'x';
        if (begin = 1)
        {
            if (newString[1][0] == 'p')
            {
                push(stack, 0);
            }

            if (newString[1][0] == 'r')
            {
                push(stack, tempR);
            }

            if (newString[1][0] == 'v' && newString[1][1] == 'a' && newString[1][2] == 'r')
            {
                push(stack, tempVar);
            }
            if (newString[1][0] == 'r' && newString[1][1] == 'e' && newString[1][2] == 't')
            {
                push(stack, ret);
            }
            if (newString[1][0] == 'a')
            {
                push(stack, z);
            }
            if (newString[1][0] == 'f')
            {
                push(stack, ret);
            }
            if (newString[1][0] == 'f' && d == 1)
            {
                push(stack, varr);
            }
            if (newString[1][0] == 'x')
            {
                push(stack, begin);
            }
            else
            {
                // push(stack, tempVar);
            }
        }
        else
        {
            if (newString[1][0] == 'p')
            {
                push(stack, p);
```

```
            }
            if (newString[1][0] == 'r')
            {
                push(stack, r);
            }
            if (newString[1][0] == 'v' && newString[1][1] == 'a' && newString[1][2] == 'r')
            {
                push(stack, var);
            }
            if (newString[1][0] == 'f' && d == 1)
            {
                push(stack, varr);
            }
            if (newString[1][0] == 'x')
            {
                push(stack, begin);
            }

            else
            {
                // push(stack, var);
            }
        }
        pCounter++;
    }
    else if (newString[0][0] == ':' && newString[0][1] == '=')
    {
        int value3 = pop(stack);
        int adrs = pop(stack);
        int *ptr;
        ptr = adrs;
        *ptr = value3;
        push(stack, adrs);
        pCounter++;
    }
    else if ((newString[0][0] == 'b' && newString[0][1] == 'e' && newString[0][2] == 'g'
&& newString[0][3] == 'i' && newString[0][3] == 'n'))
    {
        begin = 1;
        pCounter++;
        continue;
    }
    else if ((newString[0][0] == 'e' && newString[0][1] == 'n' && newString[0][2] ==
'd'))
    {
        begin = 0;
        pCounter++;
        continue;
    }

    if ((newString[0][0] == 'g' && newString[0][1] == 'o' && newString[0][2] == 't' &&
newString[0][3] == 'o') ||
        (newString[0][0] == 'g' && newString[0][1] == 'o' && newString[0][2] == 'f' &&
newString[0][3] == 'a' && newString[0][4] == 'l' && newString[0][5] == 's' &&
newString[0][6] == 'e') ||
        (newString[0][0] == 'g' && newString[0][1] == 'o' && newString[0][2] == 't' &&
newString[0][3] == 'r' && newString[0][4] == 'u' && newString[0][5] == 'e') ||
        (newString[0][0] == 'c' && newString[0][1] == 'a' && newString[0][2] == 'l' &&
newString[0][3] == 'l'))
    {
        int index = 0;
        for (int x = 0; x < 3; x++)
        {
            if ((arrayLname[x][0] == newString[1][0]) && (arrayLname[x][1] ==
newString[1][1]))
            {
                index = x;
            }
        }
        if (newString[0][0] == 'g' && newString[0][1] == 'o' && newString[0][2] == 'f' &&
newString[0][3] == 'a' && newString[0][4] == 'l' && newString[0][5] == 's' &&
newString[0][6] == 'e')

        {
            int truefalse = pop(stack);
            if (truefalse != 0)
            {
                continue;
            }
        }
        if (newString[0][0] == 'g' && newString[0][1] == 'o' && newString[0][2] == 't' &&
newString[0][3] == 'r' && newString[0][4] == 'u' && newString[0][5] == 'e')

        {
            int truefalse = pop(stack);
            if (truefalse != 1)
            {
                continue;
            }
        }

        char line1[MAX_LINE_LENGTH];
        filePointer3 = fopen(argv[1], "r");

        if (NULL == filePointer3)
        {
            printf("file can't be opened \n");
            return 0;
        }
        int whilecount = 0;
        while (fgets(line1, MAX_LINE_LENGTH, filePointer3))
        {

            int spaces1 = countspaces(line1);

            int j, ctr1, i;
            char newString1[100][80];

            j = 0;
            ctr1 = 0;
            for (i = spaces1; i <= (strlen(line1)); i++)
            {
                if (line1[i] == ' ' || line1[i] == '\0')
                {
```

```
                newString1[ctr1][j] = '\0';
                ctr1++; // for next word
                j = 0;  // for next word, init index to 0
            }
            else
            {
                newString1[ctr1][j] = line1[i];

                j++;
            }
        }
        if (whilecount >= arraylabel[0] && whilecount <= arraylabelend[0])
        {
            int pvar = 2;
            int rval = 0;
            simulator(newString1, p, r, var, ctr1, stack);
            if (newString1[0][0] == 'l' && newString1[0][1] == 'v' && newString1[0][2] ==
'a' && newString1[0][3] == 'l' && newString1[0][4] == 'u' && newString1[0][5] == 'e')
            {
                if (begin = 1)
                {
                    if (newString[1][0] == 'p')
                    {
                        int *address = &tempP;
                        int add = (int)address;
                    }

                    if (newString[1][0] == 'r')
                    {
                        int *address = &tempR;
                        int add = (int)address;
                    }

                    if (newString[1][0] == 'v' && newString[1][1] == 'a' && newString[1][2]
== 'r')

                    {
                        int *address = &tempVar;
                        int add = (int)address;
                    }
                    else
                    {
                        int *address = &tempVar;
                        int add = (int)address;
                    }
                }
                else
                {
                    if (newString1[1][0] == 'p')
                    {
                        int *address = &p;
                        int add = (int)address;
                    }

                    if (newString1[1][0] == 'r')
                    {
                        int *address = &r;
                        int add = (int)address;
                    }

                    if (newString1[1][0] == 'v' && newString1[1][1] == 'a' && newString1[1][2]
== 'r')
                    {
                        int *address = &var;
                        int add = (int)address;
                    }
                    else
                    {
                        int *address = &var;
                        int add = (int)address;
                    }
                }
            }
            else if (newString1[0][0] == 'r' && newString1[0][1] == 'v' &&
newString1[0][2] == 'a' && newString1[0][3] == 'l' && newString1[0][4] == 'u' &&
newString1[0][5] == 'e')
            {
```

```
if (begin = 1)
{

    if (newString1[1][0] == 'p')
    {
        push(stack, pvar);
    }

    if (newString1[1][0] == 'r')
    {
        push(stack, rval);
    }

    if (newString1[1][0] == 'v' && newString1[1][1] == 'a' && newString1[1][2]
== 'r')
    {
        push(stack, tempVar);
    }
    if (newString1[1][0] == 'f')
    {
        push(stack, z);
    }
    if (newString1[1][0] == 'x')
    {
        push(stack, rval);
    }
    else
    {
        // push(stack, 1);
    }
}
else
{
    if (newString1[1][0] == 'p')
    {
        push(stack, p);
    }
```

```
                if (newString1[1][0] == 'r')
                {
                  push(stack, r);
                }
                if (newString1[1][0] == 'v' && newString1[1][1] == 'a' && newString1[1][2]
== 'r')
                {
                  push(stack, var);
                }

                else
                {
                  // push(stack, tempVar);
                }
              }
            }
            else if (newString1[0][0] == ':' && newString1[0][1] == '=')
            {
              int value3 = pop(stack);
              int adrs = pop(stack);
              int *ptr;
              ptr = adrs;
              *ptr = value3;
              push(stack, adrs);
            }
          }
          whilecount++;
        }
      }
      else if (newString[0][0] == 'l' && newString[0][1] == 'a' && newString[0][2] == 'b'
&& newString[0][3] == 'e' && newString[0][4] == 'l')
      {
      }
      globalcount++;
    }
  }
  fclose(filePointer);
```

```
    return 0;
}
```