Din Bostandzic
CSE 13S

Assignment 3 - Tower of Hanoi Program Design

The program is the implementation of the Tower of Hanoi game, first done recursively then iteratively using stack data structure to obtain the same outputs in end. The game consists of 3 pegs, onto which disks must be placed, the number of disks is specified by user and they are in increasing size, initially stacked onto the first peg, A. The rules are that all of the disks from that peg must be transferred over to second peg B, using only one disk at a time, and that no disk larger than the disk currently on the peg may be stacked. The third peg, C serves as a temporary peg to allow for this process to occur. The program completes the recursion naturally, if specified by the user then outputs each move including what disk came from what peg and what peg it was then placed onto; there is then display of total number of moves taken to complete the game. As for the iterative approach it is necessary to know if an odd or even amount of disks is being used, to then swap the peg B with peg C, since it seen that when disks are odd, the first peg that the disk gets placed onto is B as a rule, as for even numbers it is first peg C, so destination peg must be swapped with temporary in order for game to properly play out.

The following functional decomposition was used to implement the program:
3.0 Tower of Hanoi program
- 3.1 recursive(in disk_num as integer, in A as character, in B as character, in C as character)
- 3.2 move_counter(in n as integer)
- 3.3 stack_manipulator(in moves as integer, in disks as integer)
- 3.3 stack_caller(in StackA as Stack structure, in StackX as Stack structure, in StackY as Stack structure, in moves as integer, in disks as integer)

Data Design

Define OPTIONS with string constant of “srn:”
Declare A, B, C as characters initialized to ‘A’, ‘B’, ‘C’
Declare next_input as string initialized to NULL
Declare default_disk as integer initialized to 0
Declare s, r, n as bool initialized to false
Declare c as integer initialized to 0

Main Module Design

```
Begin Main (pass in argc as integer, in argv as string)
    Begin While
    While (c = getopt(pass in argc as integer, in argv as string, in OPTIONS as
        string)) does not equal -1
        Begin switch (c)
```

```
                Case ‘s’
                        Assign value of true to s
                        Break statement
                Case ‘r’
                        Assign value of true to r
                        Break statement
                Case ‘n’
                        Assign value of true to n
                        Assign value of  optarg to next_input
                        Assign value of next_input converted to integer, to default_disk
                        Break statement
                Default Case
                        Display “Character not defined in the string”
                        Return with exit status fail
        End switch
End While

Begin if
If (argc == 1)
        Display “Error: no arguments supplied!”
        Return with exit status fail
End If
Begin if
If(s == true)
        Begin If
        if(n == false)
                Assign to default_disk value of 5
        End if

        Display "==================================”
        Display "----------   STACKS   ----------”
        Display "==================================”
        Assign to num_moves value from call move_counter(in default_disk as integer)
        Call stack_manipulator(in num_moves as integer, in default_disk as integer)
        Display "Number of moves: num_moves”
End if
Begin if
If(r == true)
        Begin If
        if(n == false)
```

```
                Assign to default_disk value of 5
            End if

            Display "================================”
            Display "--------   RECURSION   ---------”
            Display "================================”
            Call recursive(in default_disk as integer, in A as character, in B as character, in C
                    as character)
            Assign to num_moves value from call move_counter(in default_disk as integer)
            Display "Number of moves: num_moves”
        End if
End Main
```

recursive Module Design

```
Begin recursive
        Begin if
        If (disks == 0)
            return;                 // base case
        End if
        Begin else
        Else

            // cited, used general idea from AlgoData see README.md
            Call recursive(pass in disks-1 as integer, in A as character, in C as character, in B
                    as character)
            Display “Move disk disks from peg A to peg B”
            Call recursive(pass in disks-1 as integer, in C as character, in B as character, in A
                    as character)
        End else
End recursive
```

move_counter Module Design

```
Begin recursive
        Begin If
        If (n == 0)
            Return 0
        End if
        Begin else
        Else
```

```
            Return 2 * move_counter(n-1)+1
        End else
End recursive
```

stack_manipulator Module Data Design

```
Declare StackA of structure type Stack initialized by calling module stack_create(pass in disks
                                                                    as integer, in ‘A' as character);
Declare StackB of structure type Stack initialized by calling module stack_create(pass in disks as
                                                                    integer, in 'B' as character);
Declare StackC of structure type Stack initialized by calling module stack_create(pass in disks as
                                                                    integer, in 'C' as character);
```

stack_manipulator Module Design

```
Begin stack_manipulator
        Begin If
        If (disks % 2 == 0 )
            Call stack_caller(in StackA as Stack, in StackC as Stack, in StackB as Stack, in
                                    moves as integer, in disks as integer)
        End if
        Begin else if
        Else if (disks % 2 != 0)
            Call stack_caller(in StackA as Stack, in StackB as Stack, in StackC as Stack, in
                                    moves as integer, in disks as integer)
        End else if
End stack_manipulator
```

stack_caller Module Design

```
Begin stack_caller
        Begin for
        For (i declared as integer assign to it value of disks, i>=i, i--)
            Call stack_push(in StackA as Stack, in i as integer)
        End for
        Begin for
        For(i declared as integer assign to it value of 1, i<=moves, i++)
            Begin if
            // cited, used general idea of modding pegs and checking for even/odd moves
              from StackExchange see README.md
            If (i%3 == 1)
```

```
        Begin if
        If (call Stack_empty(in StackX as Stack))
                Call stack_push(in StackX as Stack, in value call
                                stack_pop(in StackA as Stack) as integer)
                Display "Move disk call stack_peek(in StackX as Stack) from
                                StackA->name to StackX->name”
        End if
        Begin else if
        Else if (call stack_empty(in StackA as Stack))
                Call stack_push(in StackA as Stack, in stack_pop(StackX as Stack)
                                as integer)
                 Display "Move disk call stack_peek(in StackA as Stack) from
                                        StackY->name to StackA->name”
        End else if
        Begin else if
        Else if (call stack_peek(in StackA as Stack) < call stack_peek(in StackX as Stack))
                Call stack_push(in StackX as Stack, call stack_pop(StackA) as
                                integer)
                Display “Move disk stack_peek(StackX) from StackA->name to
                        StackX->name”
        End else if
        Begin else if
        Else if (call stack_peek(StackX) < call stack_peek(StackA)
                Call stack_push(in StackA, call stack_pop(StackX) as integer)

                Display "Move disk stack_peek(StackA) from StackX->name to
                                StackA->name”
        End else if
End if
Begin if
If (i % 3 == 2)
        Begin if
        If (call stack_empty(StackY))
                Call stack_push(in StackY, in stack_pop(StackA) as integer)
                Display "Move disk stack_peek(StackY) from StackA->name to
                   StackY->name"
        End if
        Begin else if
        Else if (call Stack_empty(in StackA))
                Call stack_push(in StackA, in value call
                  stack_pop(in StackY) as integer)
```

```
                    Display "Move disk call stack_peek(in StackA) from
                      StackY->name to StackA->name”
                End else if
                Begin else if
                Else if (call stack_peek(in StackA) < call stack_peek(in StackY))
                    Call stack_push(in StackY, call stack_pop(StackA) as
                    integer)
                   Display “Move disk stack_peek(StackY) from StackY->name to
                    StackA->name”
                End else if
                Begin else if
                Else if (call stack_peek(in StackY) < call stack_peek(in StackA))
                    Call stack_push(in StackA, call stack_pop(StackY) as
                    integer)
                   Display “Move disk stack_peek(StackA) from StackY->name to
                    StackA->name”
                End else if
            End if
            Begin if
            If (i % 3 == 0)
                Begin if
                If ((stack_peek(StackX) !=0) && call stack_peek(in StackX) < call
                    stack_peek(in StackY))
                    Call stack_push(in StackY, call stack_pop(StackX) as
                    integer)
                   Display “Move disk stack_peek(StackY) from StackX->name to
                    StackY->name”
                End if
                Begin else if
                Else if (call Stack_empty(in StackA))
                    Call stack_push(in StackA, in value call
                      stack_pop(in StackY) as integer)
                    Display "Move disk call stack_peek(in StackA) from
                      StackY->name to StackA->name”
                End else if
                Begin else
                Else (call stack_push(in StackX, in stack_pop(StackY) as integer)
                      Display "Move disk stack_peek(StackX) from StackY->name to
                StackX->name”
                End else
            End if
        End For
End stack_caller
```

Stack.c Design

stack_create Module Design

```
Begin stack_create
        Dynamically allocate s of type Stack structure
        Begin if
        if (s == NULL)
                return 0
        End if
        Begin if
        if (capacity < 1)
                Assign to capacity value of 1
        End if
        Set s->capacity = capacity
        Set s->name = name
        Set s->top = 0
        Dynamically create array items of type integer given size capacity
        Begin if
        If (s->items == NULL)
                return 0
        End if
  return s
End stack_create
```

stack_delete Module Design

```
Begin stack_delete
        Call free(s->items)
        Call free(s)
        return
End stack_delete
```

stack_pop Module Design

```
Begin stack_pop
        Begin if
        if (s == NULL)
                Display "The stack not in memory"
                return -1
        End if
        Begin if
        if (s->top > 0)
```

```
                s->top -= 1
                return s->items[s->top]
        End if
        Begin else
        Else return -1
        End else
End stack_pop
```

stack_push Module Design

```
Begin stack_push
        Begin if
        If (s == NULL)
                return
        End if
        Begin if
        if (s->top == s->capacity)
                s->capacity *= 2
                 s->items = (int *)realloc(s->items, s->capacity * sizeof(int)) // reallocate memory
                                                                                    into bigger size of capacity
        End if
        Begin if
        if (s->items)
           s->items[s->top] = item
           s->top += 1
        End if
   Return
End stack_push
```

stack_empty Module Design

```
Begin stack_empty
          return s->top == 0
End stack_empty
```

stack_peek Module Design

```
Begin stack_peek
           return s->items[s->top - 1]
End stack_peek
```