B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institute, Affiliated to VTU, Belagavi)
Post Box No.: 1908, Bull Temple Road, Bengaluru – 560 019

DEPARTMENT OF MACHINE LEARNING

Academic Year: 2022-2023 (Session: October 2022 - February 2023)

DATA STRUCTURES
(22AM3PCDST)
ALTERNATIVE ASSESSMENT TOOL (AAT)

Solitaire using Stacks

Submitted by

Student Name:	**Anushka Rajgiri**	**Chetna Mundra**	**Ilaa S Chengeri**
USN:	**1BM21AI024**	**1BM21AI036**	**1BM21AI053**
Date:	**31st January 2023**		
Semester & Section:	**3A**		
Total Pages:	**35**		
Student Signature:			

Valuation Report (to be filled by the faculty)

Score:	
Comments:	
Faculty In-charge:	**Dr. Monika Puttaramaiah**

1. About the Application

Solitaire is a single-player card game that can be implemented using the C programming language. A C program for solitaire would typically consist of the following components:

- Data structures to represent cards, decks, and piles.
- Functions to shuffle, deal, and move cards within the game.
- A user interface to display the game state and accept player inputs.
- Game logic to determine valid moves, keep track of the score, and handle win/lose conditions.
- The program would also need to make use of various algorithms, such as random number generation for shuffling the deck.
- Data structures such as **stacks and queues** to represent piles of cards.

To create a solitaire program in C, the programmer would need to have a good understanding of the game rules, as well as experience with C programming and data structures. The program can be developed for a variety of platforms, including desktop computers, mobile devices, and web browsers.

Working of Data Structures:
At every move, the program pops the card from one stack and pushes it to the stack mentioned if the move is valid and the game continues until all the cards are pushed into the main deck.

Rules for solitaire:

THE SET UP

In Solitaire, there are four types of piles: the tableau, the stock, the waste, and the foundations.

The Tableau and the Stock

The **tableau** consists of seven piles. The first pile has one card, the second pile has two cards, the third pile has three cards, and so on until there are seven piles. Only the top card in each pile is face up.The cards remaining after building the tableau are placed face down and are called the **stock.**

The Waste and the Foundation

The **waste** is where three cards will be dealt face up from the stock as the game proceeds, with only the top card visible.

The **foundations** consist of four stacks of cards (one for each suit) in ascending order (Ace to King). At the beginning of the game, the foundations are empty. The goal is to play all the cards to these four foundations.

HOW TO PLAY

The object of Solitaire is to move all cards to the foundations in ascending order, beginning with the Ace and ending with the King.

- The first thing to do when starting a new game is to move any available Aces from the tableau to the foundations.

Example: The Ace of Diamonds and Ace of Hearts can be moved to the foundations.

- Next, you'll want to see if there are any Twos that can be moved to the foundations on top of the Aces already played there. In general, it is a good idea to move low-numbered cards to the foundation whenever possible.

Example: The Two of Diamonds can be moved to the foundation, on top of the Ace of Diamonds.

- When no more cards can be moved to the foundation, take a look at the tableau. Cards can be moved from one pile in the tableau to another pile, as long as the card being moved is one number lower and the opposite colour (red vs black) to the card it is placed on.

Example: The 10 of Diamonds (red) can be placed on the Jack of Clubs (black).

- Multiple cards can be moved together, as long as the cards in the sequence being moved are in descending order and are in alternating colours (red/black, etc).

Example: The Jack of Clubs can be moved onto the Queen of Diamonds, and the 10 of Diamonds is moved together with it.

- When there are no more available moves on the tableau, you can draw 3 cards from the stock.

Example: The three cards drawn are the 6 of Clubs, Jack of Clubs, and King of Hearts.

- Check to see if the top card on the waste can be moved to the tableau or the foundations.

Example: The top card is a King of Hearts. This is fortunate because there is an empty pile, and a King is the only card that can be moved to an empty pile. Now the card below it, the Jack of Clubs, becomes available.

- Keep moving cards from the tableau to the foundations if you can, or placing them on other piles within the tableau, in order to make more face down cards available. When you can't make any moves from the tableau, flip up another three cards from the stock to the waste, and continue playing.

You continue this process, flipping three cards from the stock to the waste whenever you get stuck. The goal is to keep playing cards to the tableau and to the foundations. If you successfully manage to move all the cards to the foundations, you have won the game.

*2.***Working Code**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wchar.h>
#include <locale.h>
#include <time.h>
#include <conio.h>

int temp_move = 0, move = 0, var = 0;

void delete(int arr[], int j, int c)
{
   int i;
   for (i = j; i < c - 1; i++)
   {
      arr[i] = arr[i + 1];
   }
}

int fill(int s[], int temp[], int c)
{
   int i, j;
   srand(time(0));

   for (i = 2; i <= s[0] + 2; i++)
   {
      if (c == 0)
         return 0;
```

```
        j = rand() % c;
        s[i] = temp[j];
        delete (temp, j, c);
        c--;
    }
    return c;
}

int max(int s1[], int s2[], int s3[], int s4[], int s5[], int s6[], int s7[])
{
    int max1, max2, max3;
    max1 = (s1[1] > s2[1]) ? (s1[1] > s3[1] ? s1[1] : s3[1]) : (s2[1] > s3[1] ? s2[1] : s3[1]);
    max2 = (s5[1] > s6[1]) ? (s5[1] > s7[1] ? s5[1] : s7[1]) : (s6[1] > s7[1] ? s6[1] : s7[1]);
    max3 = (max1 > max2) ? (max1 > s4[1] ? max1 : s4[1]) : (max2 > s4[1] ? max2 : s4[1]);
    return max3;
}

void cards_display(int card)
{
    wchar_t clubs = 0x2605;
    wchar_t diamonds = 0x2604;
    wchar_t spades = 0x2606;
    wchar_t hearts = 0x2603;

    if (card > 100 && card <= 113)
    {
        wprintf(L"%c", hearts);
        if (card % 100 == 13)
            printf("%c \t", 'K');
```

```
    else if (card % 100 == 12)
        printf("%c \t", 'Q');

    else if (card % 100 == 11)
        printf("%c \t", 'J');

    else if (card % 100 == 1)
        printf("%c  ", 'A');
    else
        printf("%d \t", card % 100);
}

if (card > 200 && card <= 213)
{
    wprintf(L"%c", diamonds);
    if (card % 100 == 13)
        printf("%c \t", 'K');

    else if (card % 100 == 12)
        printf("%c \t", 'Q');

    else if (card % 100 == 11)
        printf("%c \t", 'J');

    else if (card % 100 == 1)
        printf("%c  ", 'A');
    else
        printf("%d \t", card % 100);
}
```

```
if (card > 300 && card <= 313)
{
   wprintf(L"%c", spades);
   if (card % 100 == 13)
      printf("%c \t", 'K');

   else if (card % 100 == 12)
      printf("%c \t", 'Q');

   else if (card % 100 == 11)
      printf("%c \t", 'J');

   else if (card % 100 == 1)
      printf("%c  ", 'A');
   else
      printf("%d \t", card % 100);
}
if (card > 400 && card <= 413)
{
   wprintf(L"%c", clubs);
   if (card % 100 == 13)
      printf("%c \t", 'K');

   else if (card % 100 == 12)
      printf("%c \t", 'Q');

   else if (card % 100 == 11)
      printf("%c \t", 'J');

   else if (card % 100 == 1)
```

```
            printf("%c  ", 'A');
        else
            printf("%d \t", card % 100);
    }
}

void display_decks(int cd[], int od[], int d1[], int d2[], int d3[], int d4[])
{
    printf("\n\n");
    printf("[   ]");
    printf("\t");
    cards_display(od[od[1] + 1]);
    printf("\t");
    cards_display(d1[d1[0]]);
    cards_display(d2[d2[0]]);
    cards_display(d3[d3[0]]);
    cards_display(d4[d4[0]]);
    printf("\n\n\n\n");
}

void display_solitaire(int s1[], int s2[], int s3[], int s4[], int s5[], int s6[], int s7[])
{
    int matrix[7][19], i, j, k;

    for (i = 0; i < 7; i++)
    {
        for (j = 0; j < 19; j++)
            matrix[i][j] = 0;
    }
    i = 0;
```

```
k = 0;
for (j = 2; j <= (s1[1] + 1); j++)
{
   matrix[i][k] = s1[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s2[1] + 1; j++)
{
   matrix[i][k] = s2[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s3[1] + 1; j++)
{
   matrix[i][k] = s3[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s4[1] + 1; j++)
{
   matrix[i][k] = s4[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s5[1] + 1; j++)
```

```
{
   matrix[i][k] = s5[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s6[1] + 1; j++)
{
   matrix[i][k] = s6[j];
   k++;
}
i++;
k = 0;
for (j = 2; j <= s7[1] + 1; j++)
{
   matrix[i][k] = s7[j];
   k++;
}
i = 0;
j = 0;
k = max(s1, s2, s3, s4, s5, s6, s7);
while (j < k)
{
   if (s1[1] >= 0)
   {
      if (matrix[i][j] > 0)
      {
         cards_display(matrix[i][j]);
      }
      else
```

```
        printf("\t");
    i++;
}

if (s2[1] >= 0)
{
    if (matrix[i][j] > 0)
    {
        cards_display(matrix[i][j]);
    }
    else
        printf("\t");
    i++;
}

if (s3[1] >= 0)
{
    if (matrix[i][j] > 0)
    {
        cards_display(matrix[i][j]);
    }
    else
        printf("\t");
    i++;
}

if (s4[1] >= 0)
{
    if (matrix[i][j] > 0)
    {
```

```
        cards_display(matrix[i][j]);
    }
    else
        printf("\t");
    i++;
}

if (s5[1] >= 0)
{
    if (matrix[i][j] > 0)
    {
        cards_display(matrix[i][j]);
    }
    else
        printf("\t");
    i++;
}

if (s6[1] >= 0)
{
    if (matrix[i][j] > 0)
    {
        cards_display(matrix[i][j]);
    }
    else
        printf("\t");
    i++;
}

if (s7[1] >= 0)
```

```
      {
         if (matrix[i][j] > 0)
         {
            cards_display(matrix[i][j]);
         }
         else
            printf("\t");
         i++;
      }

      printf("\n");
      if (j == k - 1)
         break;
      j++;
      i = 0;
   }
}

void create(int s1[], int s2[], int s3[], int s4[], int s5[], int s6[], int s7[], int cd[])
{
   int temp[52];
   int col[4] = {100, 200, 300, 400};
   int num[13] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
   int i, j, c = 0;

   for (i = 0; i < 4; i++)
   {
      for (j = 0; j < 13; j++)
      {
         temp[c++] = col[i] + num[j];
```

```
        }
    }
    i = 0;
    s1[0] = i++;
    s2[0] = i++;
    s3[0] = i++;
    s4[0] = i++;
    s5[0] = i++;
    s6[0] = i++;
    s7[0] = i;
    cd[0] = 22;
    i = 1;
    s1[1] = i++;
    s2[1] = i++;
    s3[1] = i++;
    s4[1] = i++;
    s5[1] = i++;
    s6[1] = i++;
    s7[1] = i;
    cd[1] = 23;

    c = fill(s1, temp, c);
    c = fill(s2, temp, c);
    c = fill(s3, temp, c);
    c = fill(s4, temp, c);
    c = fill(s5, temp, c);
    c = fill(s6, temp, c);
    c = fill(s7, temp, c);
    c = fill(cd, temp, c);
}
```

```
void move_cards(int i, int s1[], int s2[])
{
   int j, k, temp[13], rear = -1;

   j = s1[1] + 1;
   int count = 0;

   for (j = s1[1] + 1; j >= i; j--)
   {
      temp[++rear] = s1[j];
      s1[1]--;
      count++;
      if (count == i)
         break;
   }

   printf("%d", temp[0]);

   k = s2[1] + 1;

   while (rear != -1)
   {
      s2[++k] = temp[rear--];
      s2[1]++;
   }

   move++;
}
```

```
int is_king(int s1[], int s2[])
{
   int i;
   i = s1[s1[1] + 1];
   if ((i % 100 == 13 || i % 200 == 13 || i % 300 == 13 || i % 400 == 13) && s2[1] == 0)
   {
      s2[++s2[1] + 1] = s1[i];
      s1[1]--;
      move++;
      return 1;
   }
   else
      return 0;
}

int valid_move_stacks(int s1[], int s2[])
{
   int i, j;

   i = s1[1] + 1;
   j = s2[1] + 1;

   if (is_king(s1, s2) == 1)
      return 1;

   int k = 1, flag = 0;

   do{
      if (s1[i] > 100 && s1[i] < 200 && (s1[i] - s2[j] == -201 || s1[i] - s2[j] == -301))
      {
```

```
        move_cards(k, s1, s2);
        flag = 1;
        if (is_king(s1, s2) == 1)
            return 1;
        return 1;
    }
    else if (s1[i] > 200 && s1[i] < 300 && (s1[i] - s2[j] == -201 || s1[i] - s2[j] == -101))
    {
        move_cards(k, s1, s2);
        flag = 1;
        if (is_king(s1, s2) == 1)
            return 1;
        return 1;
    }
    else if (s1[i] > 300 && s1[i] < 400 && (s1[i] - s2[j] == 99 || s1[i] - s2[j] == 199))
    {
        move_cards(k, s1, s2);
        flag = 1;
        if (is_king(s1, s2) == 1)
            return 1;
        return 1;
    }
    else if (s1[i] > 400 && s1[i] < 500 && (s1[i] - s2[j] == 199 || s1[i] - s2[j] == 299))
    {
        move_cards(k, s1, s2);
        flag = 1;
        if (is_king(s1, s2) == 1)
            return 1;
        return 1;
    }
```

```
        k++;
        i--;
    } while (i > 1);
    if (flag == 0)
        return 0;
}

int valid_move_deck(int s1[], int c[])
{
    int i, j;
    i = c[0];
    j = s1[1] + 1;

    c[++i] = s1[j--];
    c[0]++;
    s1[1]--;

    move++;
    return 1;
}

int move_decks(int cd[], int od[])
{
    int i, j;
    i = od[1] + 1;
    j = cd[1] + 1;

    od[++i] = cd[j--];
    od[1]++;
    cd[1]--;
```

```
    temp_move++;


    return 1;
}


int any_name(int a, int b, int s1[], int s2[], int s3[], int s4[], int s5[], int s6[], int s7[], int od[],
int cd[], int d1[], int d2[], int d3[], int d4[])
{
    switch (a)
    {
    case 1:
        switch (b)
        {
        case 2:
            return valid_move_stacks(s1, s2);
        case 3:
            return valid_move_stacks(s1, s3);
        case 4:
            return valid_move_stacks(s1, s4);
        case 5:
            return valid_move_stacks(s1, s5);
        case 6:
            return valid_move_stacks(s1, s6);
        case 7:
            return valid_move_stacks(s1, s7);
        case 8:
            return valid_move_deck(s1, d1);
        case 9:
            return valid_move_deck(s1, d2);
        case 10:
```

```
        return valid_move_deck(s1, d3);
    case 11:
        return valid_move_deck(s1, d4);
    default:
        break;
    };
case 2:
    switch (b)
    {
    case 1:
        return valid_move_stacks(s2, s1);
    case 3:
        return valid_move_stacks(s2, s3);
    case 4:
        return valid_move_stacks(s2, s4);
    case 5:
        return valid_move_stacks(s2, s5);
    case 6:
        return valid_move_stacks(s2, s6);
    case 7:
        return valid_move_stacks(s2, s7);
    case 8:
        return valid_move_deck(s2, d1);
    case 9:
        return valid_move_deck(s2, d2);
    case 10:
        return valid_move_deck(s2, d3);
    case 11:
        return valid_move_deck(s2, d4);
    default:
```

```
        break;
      };
    case 3:
      switch (b)
      {
      case 1:
        return valid_move_stacks(s3, s1);
      case 2:
        return valid_move_stacks(s3, s2);
      case 4:
        return valid_move_stacks(s3, s4);
      case 5:
        return valid_move_stacks(s3, s5);
      case 6:
        return valid_move_stacks(s3, s6);
      case 7:
        return valid_move_stacks(s3, s7);
      case 8:
        return valid_move_deck(s3, d1);
      case 9:
        return valid_move_deck(s3, d2);
      case 10:
        return valid_move_deck(s3, d3);
      case 11:
        return valid_move_deck(s3, d4);
      default:
        break;
      };
    case 4:
      switch (b)
```

```
    {
    case 1:
      return valid_move_stacks(s4, s1);
    case 2:
      return valid_move_stacks(s4, s2);
    case 3:
      return valid_move_stacks(s4, s3);
    case 5:
      return valid_move_stacks(s4, s5);
    case 6:
      return valid_move_stacks(s4, s6);
    case 7:
      return valid_move_stacks(s4, s7);
    case 8:
      return valid_move_deck(s4, d1);
    case 9:
      return valid_move_deck(s4, d2);
    case 10:
      return valid_move_deck(s4, d3);
    case 11:
      return valid_move_deck(s4, d4);
    default:
      break;
    };
  case 5:
    switch (b)
    {
    case 1:
      return valid_move_stacks(s5, s1);
    case 3:
```

```
        return valid_move_stacks(s5, s3);
      case 4:
        return valid_move_stacks(s5, s4);
      case 2:
        return valid_move_stacks(s5, s2);
      case 6:
        return valid_move_stacks(s5, s6);
      case 7:
        return valid_move_stacks(s5, s7);
      case 8:
        return valid_move_deck(s5, d1);
      case 9:
        return valid_move_deck(s5, d2);
      case 10:
        return valid_move_deck(s5, d3);
      case 11:
        return valid_move_deck(s5, d4);
      default:
        break;
      };
    case 6:
      switch (b)
      {
      case 1:
        return valid_move_stacks(s6, s1);
      case 3:
        return valid_move_stacks(s6, s3);
      case 4:
        return valid_move_stacks(s6, s4);
      case 5:
```

```
        return valid_move_stacks(s6, s5);
    case 2:
        return valid_move_stacks(s6, s2);
    case 7:
        return valid_move_stacks(s6, s7);
    case 8:
        return valid_move_deck(s6, d1);
    case 9:
        return valid_move_deck(s6, d2);
    case 10:
        return valid_move_deck(s6, d3);
    case 11:
        return valid_move_deck(s6, d4);
    default:
        break;
    };
case 7:
    switch (b)
    {
    case 1:
        return valid_move_stacks(s7, s1);
    case 3:
        return valid_move_stacks(s7, s3);
    case 4:
        return valid_move_stacks(s7, s4);
    case 5:
        return valid_move_stacks(s7, s5);
    case 6:
        return valid_move_stacks(s7, s6);
    case 2:
```

```
        return valid_move_stacks(s7, s2);
    case 8:
        return valid_move_deck(s7, d1);
    case 9:
        return valid_move_deck(s7, d2);
    case 10:
        return valid_move_deck(s7, d3);
    case 11:
        return valid_move_deck(s7, d4);
    default:
        break;
    };
case 12:
    return move_decks(cd, od);

case 13:
    switch (b)
    {
    case 1:
        return valid_move_stacks(od, s1);
    case 3:
        return valid_move_stacks(od, s3);
    case 2:
        return valid_move_stacks(od, s2);
    case 4:
        return valid_move_stacks(od, s4);
    case 5:
        return valid_move_stacks(od, s5);
    case 6:
        return valid_move_stacks(od, s6);
```

```
    case 7:
      return valid_move_stacks(od, s7);
    case 8:
      return valid_move_deck(od, d1);
    case 9:
      return valid_move_deck(od, d2);
    case 10:
      return valid_move_deck(od, d3);
    case 11:
      return valid_move_deck(od, d4);
    default:
      break;
    }
  case 8:
    switch (b)
    {
    case 1:
      return valid_move_stacks(d1, s1);
    case 3:
      return valid_move_stacks(d1, s3);
    case 4:
      return valid_move_stacks(d1, s4);
    case 5:
      return valid_move_stacks(d1, s5);
    case 6:
      return valid_move_stacks(d1, s6);
    case 7:
      return valid_move_stacks(d1, s7);
    case 2:
      return valid_move_deck(d1, s2);
```

```
    default:
      break;
    };
  case 9:
    switch (b)
    {
    case 1:
      return valid_move_stacks(d2, s1);
    case 3:
      return valid_move_stacks(d2, s3);
    case 4:
      return valid_move_stacks(d2, s4);
    case 5:
      return valid_move_stacks(d2, s5);
    case 6:
      return valid_move_stacks(d2, s6);
    case 7:
      return valid_move_stacks(d2, s7);
    case 2:
      return valid_move_deck(d2, s2);
    default:
      break;
    };
  case 10:
    switch (b)
    {
    case 1:
      return valid_move_stacks(d3, s1);
    case 3:
      return valid_move_stacks(d3, s3);
```

```
    case 4:
      return valid_move_stacks(d3, s4);
    case 5:
      return valid_move_stacks(d3, s5);
    case 6:
      return valid_move_stacks(d3, s6);
    case 7:
      return valid_move_stacks(d3, s7);
    case 2:
      return valid_move_deck(d3, s2);
    default:
      break;
    };
  case 11:
    switch (b)
    {
    case 1:
      return valid_move_stacks(d4, s1);
    case 3:
      return valid_move_stacks(d4, s3);
    case 4:
      return valid_move_stacks(d4, s4);
    case 5:
      return valid_move_stacks(d4, s5);
    case 6:
      return valid_move_stacks(d4, s6);
    case 7:
      return valid_move_stacks(d4, s7);
    case 2:
      return valid_move_deck(d4, s2);
```

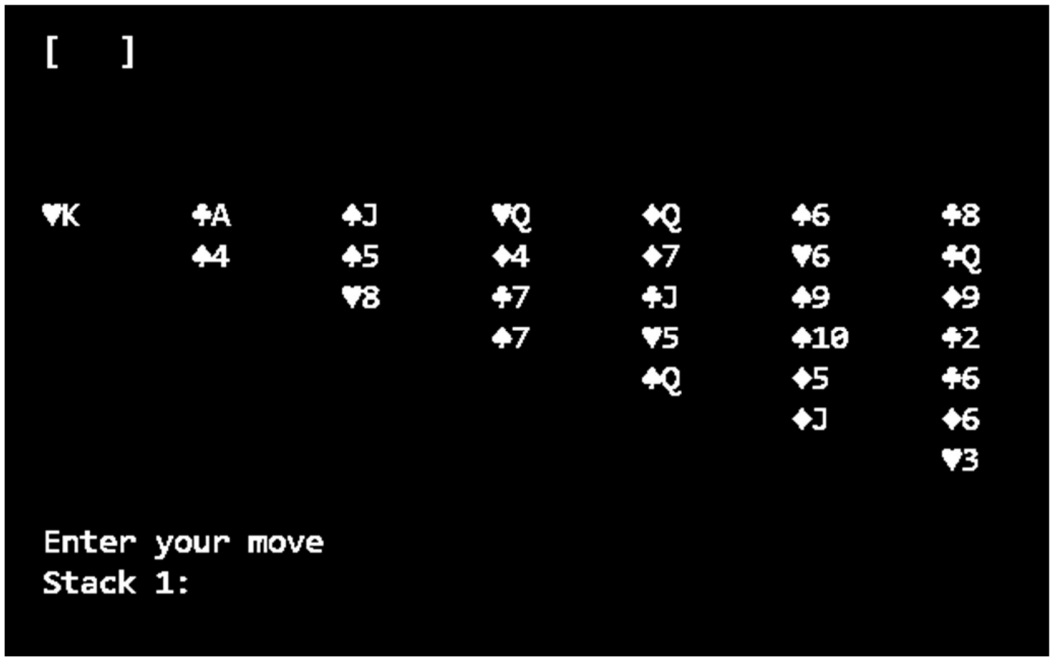
```
      default:
         break;
      };
   default:
      return 0;
   };
}

int lose(int cd[], int od[])
{
   if (cd[1] == 0)
   {
      if (var == move)
      {
         return 1;
      }
      var = move;
      temp_move = 0;
      int k;
      k = cd[1] + 1;
      for (int i = od[1] + 1; i > 1; i--)
      {
         cd[++k] = od[od[1] + 1];
         od[1]--;
         cd[1]++;
      }
   }
   return 0;
}
```

```
int win(int s1[], int s2[], int s3[], int s4[], int s5[], int s6[], int s7[])
{
   if (s1[1] == 0 && s2[1] == 0 && s3[1] == 0 && s4[1] == 0 && s5[1] == 0 && s6[1] == 0 &&
s7[1] == 0)
      return 1;

   return 0;
}

void main()
{
   int s1[30], s2[30], s3[30], s4[30], s5[30], s6[30], s7[30];
   int d1[13], d2[13], d3[13], d4[13];
   int od[50], cd[50];
   int x;

   od[0] = -1;
   od[1] = 1;

   d1[0] = 0;
   d2[0] = 0;
   d3[0] = 0;
   d4[0] = 0;
   int a, b;
   int i[52], j[52];
   srand(time(0));
   create(s1, s2, s3, s4, s5, s6, s7, cd);
   display_decks(cd, od, d1, d2, d3, d4);
   display_solitaire(s1, s2, s3, s4, s5, s6, s7);
```

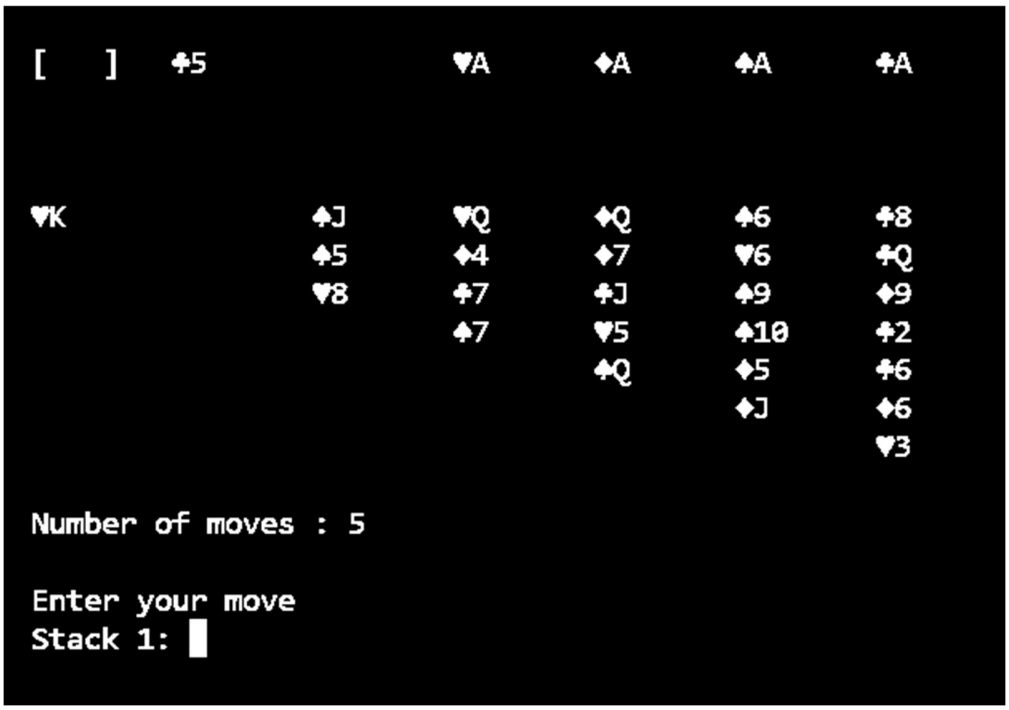
```
for (int k = 0; k < 200; k++)
  {
    printf("\nEnter your move\n");
    printf("Stack 1: ");
    scanf("%d", &a);
    printf("Stack 2: ");
    scanf("%d", &b);
    x = any_name(a, b, s1, s2, s3, s4, s5, s6, s7, od, cd, d1, d2, d3, d4);
    if (x == 1)
    {
      system("cls");
      display_decks(cd, od, d1, d2, d3, d4);
      display_solitaire(s1, s2, s3, s4, s5, s6, s7);
      printf("\nNumber of moves : %d \n", move);
    }
    else
      printf("Invalid move");
    if (lose(cd, od) == 1)
    {
      printf("You lost");
      break;
    }
    if (win(s1, s2, s3, s4, s5, s6, s7) == 1)
    {
      printf("You won!!!");
      break;
    }
  }
}
```

2. Screen shots

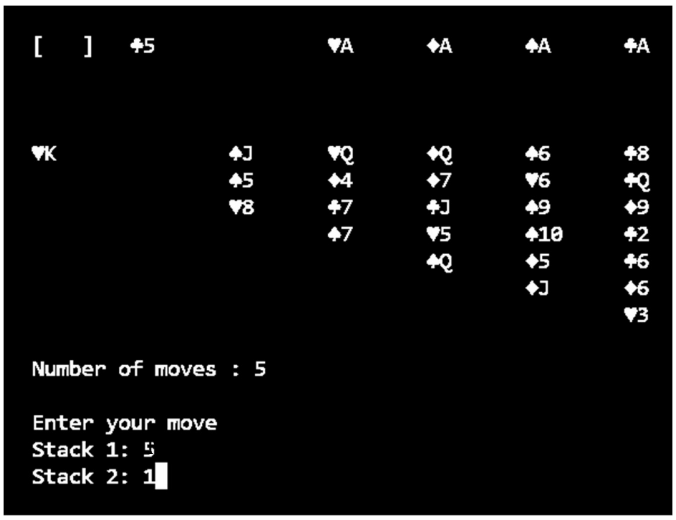
Game start:



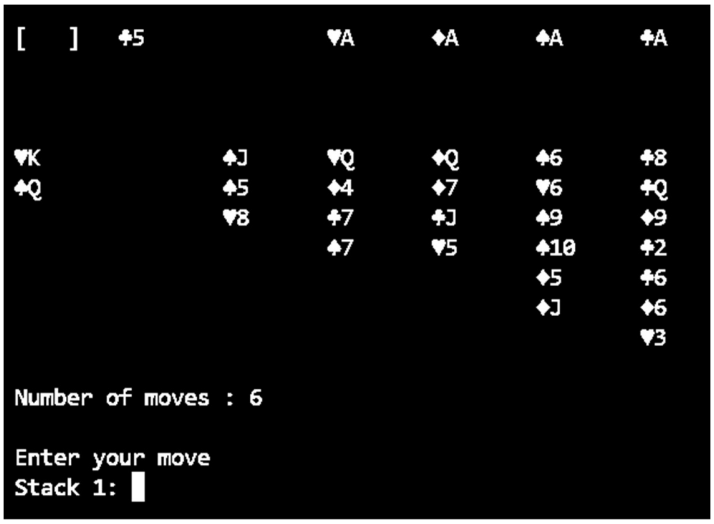
After a Few moves:



Move from Stack 1 to Stack 2:

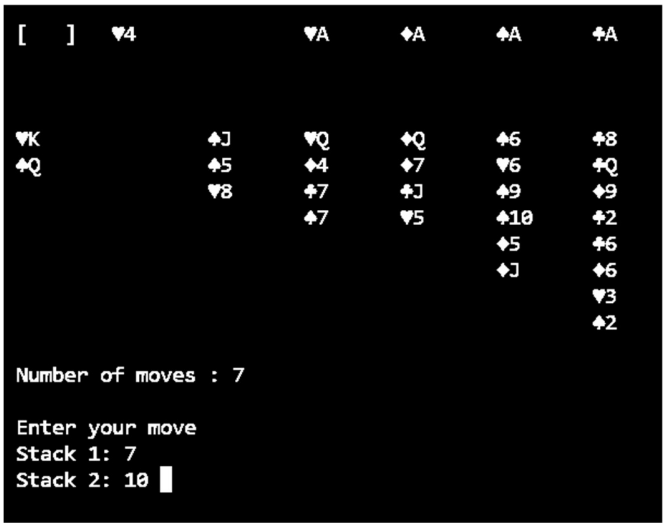


input

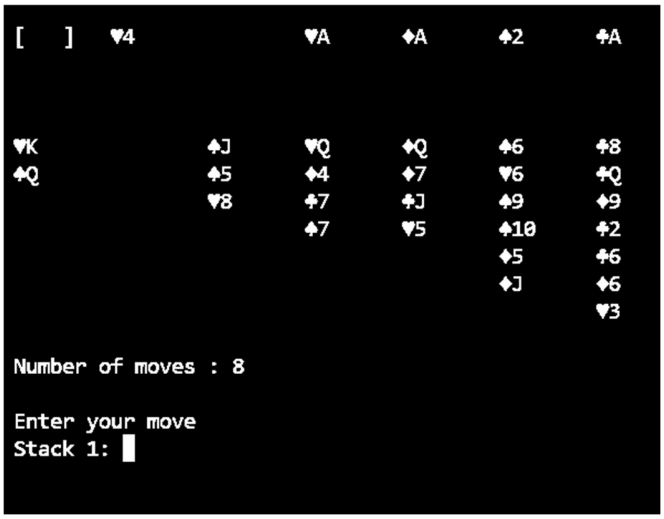


output

Move from Stacks to Central deck:

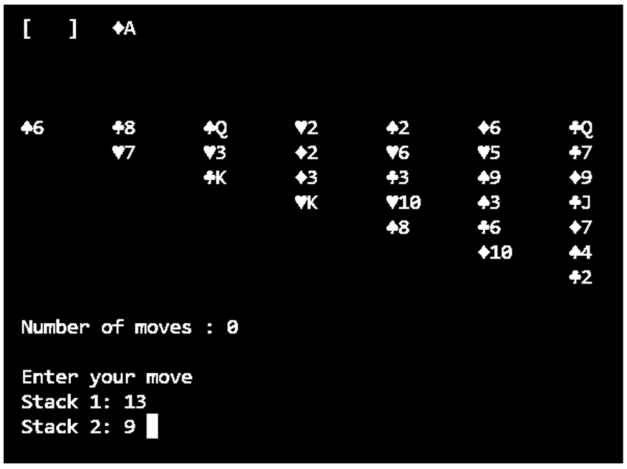


Input

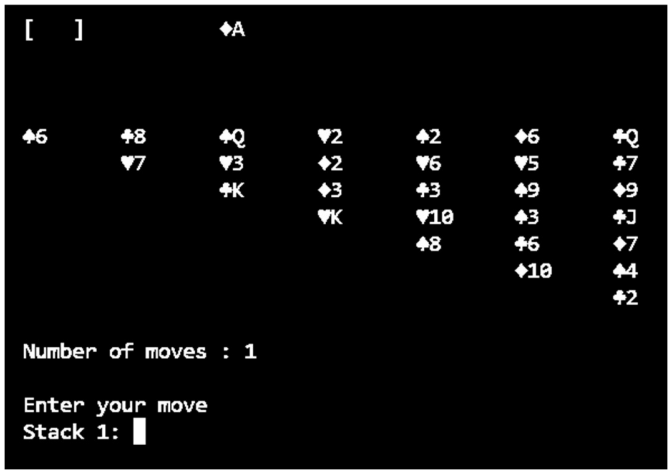


Output

Waste to open deck

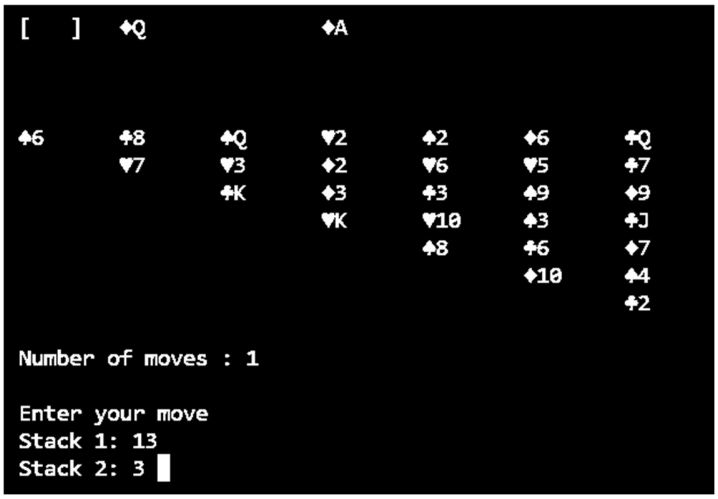


Input



Output

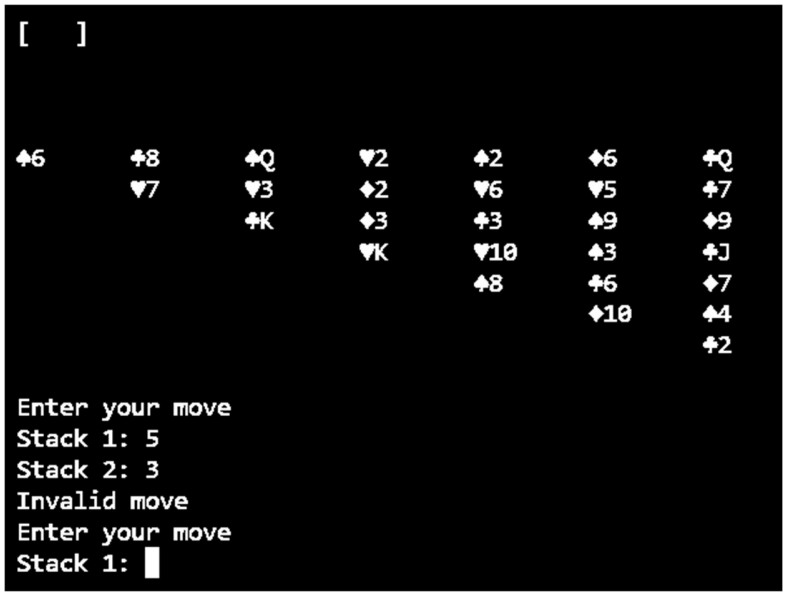
Open deck to stack



Input

Output

Invalid move



Win

Lost