FAST NATIONAL UNIVERSTIY OF COMPUTER AND EMERGING SCIENCES, PESHAWAR

DEPARTMENT OF COMPUTER SCIENCE

CL217 – OBJECT ORIENTED PROGRAMMING LAB

LAB MANUAL # 07

INSTRUCTOR: MUHAMMAD ABDULLAH

SEMESTER SPRING 2021

OBJECT ORIENTED PROGRAMMING LANGUAGE

Table of Contents

Structure:

Structure is a collection of variables under a single name. Variables can be of any type: int, float, char etc. The main difference between structure and array is that arrays are collections of the same data type and structure is a collection of variables under a single name.

Why we need Structures?

As we noticed arrays are very powerful device that allow us to group large amounts of data together under a single variable name.

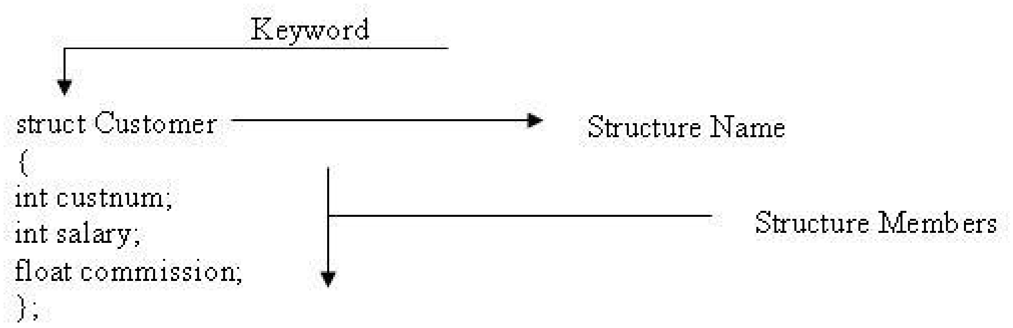
How would you write a program if, instead of being asked for simple list of either integers or characters, you were asked to combine integers, floating point numbers, and string with one variable name? You could not do it by using array.

It is actually quit often we want to group logically connected data that are of different types together. Just think about writing a program to store student record. You would need string for your name, integers to store the ID number, and a floating-point number to store the grades.

Declaring a Structure

Example:

Three variables: custnum of type int, salary of type int, commission of type float are structure members and the structure name is Customer. This structure is declared as follows:



```
struct Customer
{
int custnum;
int salary;
float commission;
};
```

In the above example, it is seen that variables of different types such as int and float are grouped in a single structure name Customer.

Arrays behave in the same way, declaring structures does not mean that memory is allocated. Structure declaration gives a skeleton or template for the structure.

```
struct st_name

{

        type l;

        type m;

        type n;

} ;
```

Where

st_name	Represents the structure name. It is also called **Structure tag**. It is used to declare variables of structure type.
type	Represent the type of the variable.
l,m,n	Represent members of the structure. These may have different or same data type. The members are enclosed in braces.

A semicolon after the closing bracket (**} ;**) indicates the end of the structure.

Each member of a structure must have unique name but different structures may have same names.

A structure is first defined and then variables of structure type are declared. A variable of a structure type is declared to access data in the members of the structure.

Example: Declare a structure with ***address*** as a tag and having two members ***name*** of character type and ***age*** of integer type.

```
struct address

{

        char name [15];

        int age;

} ;
```

The structure member **name** is of character type with 15 characters length (including the null character) and **age** is of integer type.

Structure Variables

After declaring the structure, the next step is to define a structure variable.

A structure is a collection of data items or elements. It is defined to declare its variables. These are called **structure variables**. A variable of structure type represents the members of its structure.

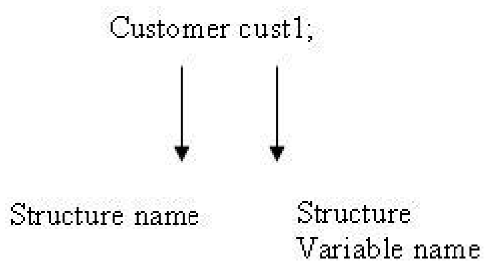
When a structure type variable is declared a space is reserved in the computer memory to hold all members of the structure. The memory occupied by a structure variable is equal to the sum of the memory occupied be each member of the structure.

A structure can be defined prior to or inside the main() function. If it is defined inside the main() function then the structure variable can be declared only inside the main function. If it is declared prior to the main() function, its variables can be defined anywhere in the program.

```
struct address

{

        char city[15];

        int pcode;

} ;

address taq  ,  aye;
```

In the above example, **city** and **pcode** are members of the structure **address**. The variable **taq** and **aye** are declared as structure variables.

The structure **address** has two members **city** and **pcode**. The variable **city** occupies 15 bytes and **pcode** occupies 2 bytes. Thus, each variable of this structure will occupy 17 bytes space in memory, i.e. 15 bytes for **city** and 2 bytes **pcode**.



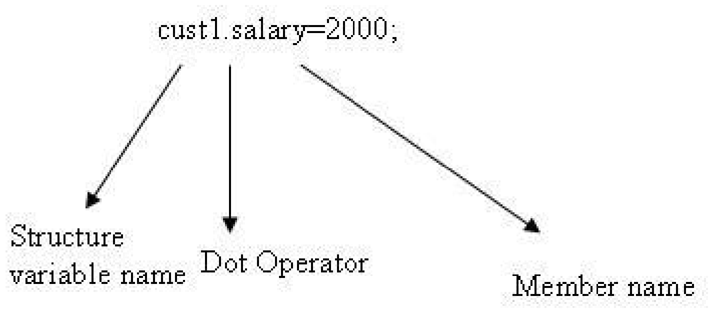
How to access structure members in C++?

To access structure members, the operator used is the dot operator denoted by (.). The dot operator for accessing structure members is used thusly:

```
Structure_Variable_name.member_name;
```

For example:

A programmer wants to assign 2000 for the structure member salary in the above example of structure Customer with structure variable cust1 this is written as:



```
// A Complete example by declaring, using structs as well as data
members.
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct employe
{
// members of the structures
string name;
double hours_worked;
double salary;
};
int main()
{
      employe a,b; //structure variables

      //accessing structure variables
      a.name="Asia";
      a.hours_worked=30;
      a.salary=a.hours_worked*50;

      b.name="Naveed";
      b.hours_worked=60;
      b.salary=b.hours_worked*50;
      //*for 1st employee
      cout<<"Employee Name:\t"<<a.name<<endl;
      cout<<"Hours worked:\t"<<a.hours_worked<<endl;
      cout<<"Salary:\t\t"<<a.salary<<endl;

      //*for second employee
      cout<<"Employee Name:\t"<<b.name<<endl;
      cout<<"Hours worked:\t"<<b.hours_worked<<endl;
      cout<<"Salary:\t\t"<<b.salary<<endl;

}

/*Sample Output

Employee Name:  Asia
Hours worked:   30
Salary:         1500
Employee Name:  Naveed
Hours worked:   60
Salary:         3000
```

Members of Structures

Structures in C++ can contain two types of members:

Data Member: These members are normal C++ variables. We can create a structure with variables of different data types in C++.

Member Functions: These members are normal C++ functions. Along with variables, we can also include functions inside a structure declaration.

Example:

```
// Data Members
int roll;
int age;
int marks;

// Member Functions
void printDetails()
{
      cout<<"Roll = "<<roll<<"\n";
      cout<<"Age = "<<age<<"\n";
      cout<<"Marks = "<<marks;
}
```

In the above structure, the data members are three integer variables to store *roll number, age and marks* of any student and the member function is *printDetails()* which is printing all of the above details of any student.

Function inside Structure

```
// Function inside structs
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct employe
{
string name;
double hours_worked;
double salary;

//this function will initialize members of structure
void input(string n, double h)
{
name=n;
hours_worked=h;
}

//this function will print members of structure
void print()
{
cout<<"Employee Name:\t"<<name<<endl;
cout<<"Hours worked:\t"<<hours_worked<<endl;
salary=50*hours_worked;
cout<<"Salary:\t\t"<<salary<<endl;
}
};    //structure body closed
int main()
{
      employe a;     //structure variable
      a.input("Aisha",10);
      a.print();
}
/*
Sample output:
Employee Name:  Aisha
Hours worked:   10
Salary:         500
*/
```

Structs of Arrays:

```
#include <iostream>
#include <conio.h>
#include <string.h>
using namespace std;
struct STUDENT
{
//members of structure
int idNum;
int testScores[3];
int finalExam;

//initialize members of structure with 0
void init()
{
idNum=0;
testScores[0]=0;
testScores[1]=0;
testScores[2]=0;
finalExam=0;
}

//initialize members of structure with arguments passed to it
void init(int a, int b[], int c)
{
idNum=a;
testScores[0]=b[0];
testScores[1]=b[1];
testScores[2]=b[2];
finalExam=c;
}
// this function will print values
void printStudent()
{
cout<<" ID : "<<idNum
<<"\tSrcor-1 : "<<testScores[0]
<<"\tSrcor-2 : "<<testScores[1]
<<"\tSrcor-3 : "<<testScores[2]
<<"Final : "<<finalExam<<endl<<" ";
}
};
int main()
{
STUDENT s1,s2;
// initializing s1 elements one by one without function
s1.idNum=111;
s1.testScores[0]=95;
s1.testScores[1]=80;
s1.testScores[2]=98;
s1.finalExam=100;
s1.printStudent();

}
/*
 ID : 111   Srcor-1 : 95    Srcor-2 : 80    Srcor-3 : 98Finale : 100
*/
```

```
// initializing s1 and s2 elements one by one with function

#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct STUDENT
{
int idNum;
int testScores[3];
int finalExam;
void init()
{
idNum=0;
testScores[0]=0;
testScores[1]=0;
testScores[2]=0;
finalExam=0;
}
void init(int a, int b[], int c)
{
idNum=a;
testScores[0]=b[0];
testScores[1]=b[1];
testScores[2]=b[2];
finalExam=c;
}


void printStudent(string sName)
{
cout<<"Student : "<<sName
     <<" ID : "<<idNum
<<"\tSrcors : "<<testScores[0]
<<", "<<testScores[1]
<<", "<<testScores[2]
<<"\tFinal : "<<finalExam<<endl;
}
};
int main()
{
STUDENT s1,s2;    //declaring structure variables
s1.init();        // calling non parameterized init() function
s1.printStudent("Ali");
int scores[]={80,55,78};   //declaring and initializing array
s2.init(123,scores,98);   // calling parameterized init() function
s2.printStudent("Saad");
}
/*
Student : Ali ID : 0     Srcors : 0, 0, 0 Final : 0
Student : Saad ID : 123   Srcors : 80, 55, 78 Final : 98

*/
```

Arrays of Structs:

What is an array of structures?

Like other primitive data types, we can create an array of structures.

```
#include <iostream>
using namespace std;

struct Point {
	int x, y;
};

int main()
{
	// Create an array of structures
	struct Point arr[10];

	// Access array members
	arr[0].x = 10;
	arr[0].y = 20;

	cout << arr[0].x << " " << arr[0].y;

	return 0;
}
```

Output

```
10    20
```

Arrays of Structs Example 2:

```
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct STUDENT
{
string sName;
int idNum;
float cgpa;
void init()
{
sName="Student";
idNum=0;
cgpa=0;
}
void init(string name, int id, float c)
{
sName=name;
idNum=id;
cgpa=c;
}
void printStudent()
{
cout<<"Name : "<<sName
<<"\tID : "<<idNum
<<"\tCGPA : "<<cgpa<<endl;
}
};
int main()
{

STUDENT s[5];
s[0].init();
s[1].init("Najeeb",1,3.67);
s[2].init("Nadeem",2,3.57);
s[3].init("Basit",3,3.60);
s[4].init("Washaq",3,3.60);

for(int i=0 ; i<5 ; i++)// printing Studetns Strcut's Array
{
s[i].printStudent();
}

}
```

```
/*
Name : Student  ID : 0  CGPA : 0
Name : Najeeb   ID : 1  CGPA : 3.67
Name : Nadeem   ID : 2  CGPA : 3.57
Name : Basit    ID : 3  CGPA : 3.6
Name : Washaq   ID : 3  CGPA : 3.6
*/
```

What is a structure pointer?

Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator instead of the dot (.) operator.

```
#include <iostream>
using namespace std;

struct Point {
    int x, y;
};

int main()
{
    Point p1, *p2;    // structure variables

    p1.x=2;
    p1.y=3;

    // p2 is a pointer to structure p1
    p2 = &p1;

    // Accessing structure members using
    // structure pointer
    cout << p2->x << " " << p2->y;
    return 0;
}

Output
2   3
```

Nested Structure in C++

When a structure contains another structure, it is called nested structure. For example, we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Syntax for structure within structure or nested structure

```
struct structure1
       {
            - - - - - - - - - -
            - - - - - - - - - -
       };
struct structure2
        {
             - - - - - - - - - -
             - - - - - - - - - -
             struct structure1 obj;
         };
```

Nested Structs Example 1

```
#include <iostream>
using namespace std;
struct Address
        {
               char HouseNo[25];
               char City[25];
               char PinCode[25];
        };

struct Employee
       {
           int Id;
           char Name[25];
           float Salary;
           struct Address Add;
       };

int main()
    {
        int i;
        Employee E;

        cout << "\n\tEnter Employee Id : ";
        cin >> E.Id;

        cout << "\n\tEnter Employee Name : ";
        cin >> E.Name;
        cout << "\n\tEnter Employee Salary : ";
        cin >> E.Salary;

        cout << "\n\tEnter Employee House No : ";
        cin >> E.Add.HouseNo;

        cout << "\n\tEnter Employee City : ";
        cin >> E.Add.City;

        cout << "\n\tEnter Employee House No : ";
        cin >> E.Add.PinCode;

        cout << "\nDetails of Employees";
        cout << "\n\tEmployee Id : " << E.Id;
        cout << "\n\tEmployee Name : " << E.Name;
        cout << "\n\tEmployee Salary : " << E.Salary;
        cout << "\n\tEmployee House No : " << E.Add.HouseNo;
        cout << "\n\tEmployee City : " << E.Add.City;
        cout << "\n\tEmployee House No : " << E.Add.PinCode;
    }
```

Output :

```
             Enter Employee Id : 101
             Enter Employee Name : Suresh
             Enter Employee Salary : 45000
             Enter Employee House No : 4598/D
             Enter Employee City : Delhi
             Enter Employee Pin Code : 110056
Details of Employees
             Employee Id : 101
             Employee Name : Suresh
             Employee Salary : 45000
             Employee House No : 4598/D
             Employee City : Delhi
             Employee Pin Code : 110056
```

Nested Structs Example 2

```
#include <iostream>
#include <conio.h>
#include <string>
using namespace std;
struct COURSE
{
int code;
string title;
int crdHours;
double gradePoints;
void init(int cd, string tl, int ch,double pts)
{
code=cd;
title=tl;
crdHours=ch;
gradePoints=pts;
}
void printCourseInfo()
{
cout<<"\t"<<code<<"\t"<<title<<"\t"<<crdHours<<"\t"<<gradePoints<<endl;
}
};
struct STUDENT
      {
string sName;
int idNum;
double cgpa;
COURSE myCourses[5];
void init(string name, int id)
{
sName=name;
idNum=id;
}
void calculateCGPA()
{
double temp1=0,temp2=0;
for(int i=0;i<5;i++)
{
temp1+=myCourses[i].gradePoints*myCourses[i].crdHours;
temp2+=myCourses[i].crdHours;
}
cgpa= (temp1/temp2);
}
void printStudent()
{
cout<<" Name : "<<sName<<"\tID : "<<idNum
<<"\tCGPA : "<<cgpa<<endl;
cout<<"\tCode\tTitle\tCrd.Hours\tGrade Points"<<endl;
for(int i=0;i<5;i++)
{
myCourses[i].printCourseInfo();
}
}
};
```

```
int main()
{
STUDENT s1,s2;
s1.init("Muhammad Abdullah",132);
// assigning values to courses struct one by one access
s1.myCourses[0].code=102;
s1.myCourses[0].title="Into To Com";
s1.myCourses[0].crdHours=3;
s1.myCourses[0].gradePoints=3.33;
s1.myCourses[1].init(103,"Calculus-I",3,3.00);
s1.myCourses[2].init(105,"English - I",3,2.67);
s1.myCourses[3].init(205,"Linear Algebra",3,3.67);
s1.myCourses[4].init(212,"Physics - I",3,3.33);
s1.calculateCGPA();
s1.printStudent();
}

/*
Name : Muhammad Abdullah    ID : 132        CGPA : 3.2
        Code    Title   Crd.Hours       Grade Points
        102     Into To Com     3       3.33
        103     Calculus-I      3       3
        105     English - I     3       2.67
        205     Linear Algebra  3       3.67
        212     Physics - I     3       3.33
Press any key to continue . . .
*/
```

Nested Structs Example 3

```
#include <iostream>
#include <conio.h>
using namespace std;
struct Distance
{
int feet;
float inches;
};
struct Room
{
Distance length;
Distance width;
};
void main()
{
Room dining; //define a room
dining.length.feet = 13; //assign values to room
dining.length.inches = 6.5;
dining.width.feet = 10;
dining.width.inches = 0.0;
//Alternative: Room dining = { {13, 6.5}, {10, 0.0} };
//convert length & width
float l = dining.length.feet + dining.length.inches/12;
float w = dining.width.feet + dining.width.inches/12;
//find area and display it
cout << "Dining room area is " << l * w<< " square feet\n" ;
system("pause");
}
/*Sample Run:
Dining room area is 135.417 square feet
*/
```

Structure and Function in C++

Using function, we can pass structure as function argument and we can also return structure from function.

Structure can be passed to function through its object therefore passing structure to function or passing structure object to function is same thing because structure object represents the structure. Like normal variable, structure variable (structure object) can be pass by value or by references / addresses

Passing Structure by Value

In this approach, the structure object is passed as function argument to the definition of function, here object is representing the members of structure with their values.

Example for passing structure object by value

```
#include<iostream>
using namespace std;
struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};

void Display(Employee);   // function declaration
```

```
int main()
{
      // Initializing structure variables
      Employee Emp = {1,"Aisha",29,45000};
      Display(Emp);   // calling function
}

void Display(Employee E)
{
    cout << "\n\nEmployee Id : " << E.Id;
    cout << "\nEmployee Name : " << E.Name;
    cout << "\nEmployee Age : " << E.Age;
    cout << "\nEmployee Salary : " << E.Salary;
}
```

Output:

```
Employee Id : 1
Employee Name : Aisha
Employee Age : 29
Employee Salary : 45000
```

Passing Structure by Reference

In this approach, the reference/address structure object is passed as function argument to the definition of function.

Example for passing structure object by reference

```
#include<iostream>
using namespace std;

struct Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};

void Display(Employee*);   // function declaration

int main()
{
    Employee Emp = {1,"Kashif",29,45000};

    Display(&Emp);

}

void Display(Employee *E)
{
    cout << "\n\nEmployee Id : " << E->Id;
    cout << "\nEmployee Name : " << E->Name;
    cout << "\nEmployee Age : " << E->Age;
    cout << "\nEmployee Salary : " << E->Salary;
}
```

Output:

```
Employee Id : 1
Employee Name : Kashif
Employee Age : 29
Employee Salary : 45000
```

Function Returning Structure

Structure is user-defined data type, like built-in data types structure can be return from function.

Example for Function Returning Structure

```
#include<iostream>
using namespace std;

struct Employee
{
         int Id;
         char Name[25];
         int Age;
         long Salary;
};

Employee Input();            //Statement 1
int main()
{
    Employee Emp;

    Emp = Input();  // calling function

    cout << "\n\nEmployee Id : " << Emp.Id;
    cout << "\nEmployee Name : " << Emp.Name;
    cout << "\nEmployee Age : " << Emp.Age;
    cout << "\nEmployee Salary : " << Emp.Salary;

}
Employee Input()
{
    Employee E;   // declaring structure variable

    cout << "\nEnter Employee Id : ";
    cin >> E.Id;

    cout << "\nEnter Employee Name : ";
    cin >> E.Name;

    cout << "\nEnter Employee Age : ";
    cin >> E.Age;

    cout << "\nEnter Employee Salary : ";
    cin >> E.Salary;

    return E;             //Statement   2
}

Output:
Enter Employee Id : 1
Enter Employee Name : Amir
Enter Employee Age : 23
Enter Employee Salary : 234235

Employee Id : 1
Employee Name : Amir
Employee Age : 23
Employee Salary : 234235
```

In the above example, statement 1 is declaring Input() with return type Employee. As we know structure is user-defined data type and structure name acts as our new user-defined data type, therefore we use structure name as function return type.

Input() have local variable E of Employee type. After getting values from user statement 2 returns E to the calling function and display the values.

Students Exercise 1

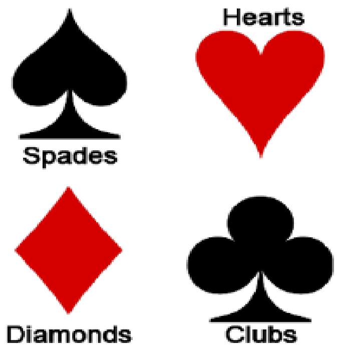
```
#include <iostream>
#include <conio.h>
```

```
using namespace std;
struct student
{
int rollno;
int examScore;
int labScore;
};
void getStudent(student& stu);
void showStudent(student stu);
student newStudent();
int main()
{
student stu1,stu2;
getStudent(stu1);
showStudent(stu1);
cout<<"\nNow with struct returning function"<<endl;
stu2 = newStudent();
showStudent(stu2);
system("pause");
}

void getStudent(student& stu)
{
cout<<"Enter Student's Information"<<endl;
cout<<"Roll No: ";
cin>>stu.rollno;
cout<<"Exam Score: ";
cin>>stu.examScore;
cout<<"Lab Score: ";
cin>>stu.labScore;
}
void showStudent(student stu)
{
cout<<"\nStudent's Informantion"<<endl;
cout<<"Roll No: "<<stu.rollno<<endl;
cout<<"Exam Score: "<<stu.examScore<<endl;
cout<<"Lab Score: "<<stu.labScore<<endl;
}
student newStudent()
{
student std1;
std1.rollno = 123;
std1.examScore = 60;
std1.labScore = 15;
return std1;
}
```

```
/*Sample Run:
Enter Student's Information
Roll No: 5
Exam Score: 40
Lab Score: 8
Student's Informantion
Roll No: 5
Exam Score: 40
Lab Score: 8
Now with struct returning function
Student's Informantion
Roll No: 123
Exam Score: 60
Lab Score: 15
*/
```

Students Exercise 2

Game of cards



```
#include <iostream>
#include <conio.h>
using namespace std;
const int clubs = 0;
const int diamonds = 1;
const int hearts = 2;
const int spades = 3;
const int jack = 11;
const int queen = 12;
const int king = 13;
const int ace = 14;
struct card
{
	int number; //2 to 10, jack, queen, king, ace
	int suit; //clubs, diamonds, hearts, spades
};

int main()
{
	card temp, chosen, prize; //define cards
	int position;

	card card1 = { 7, clubs }; //initialize card1
	cout << "Card 1 is the 7 of clubs\n";
```

```
      card card2 = { jack, hearts }; //initialize card2
      cout << "Card 2 is the jack of hearts\n";

      card card3 = { ace, spades }; //initialize card3
      cout << "Card 3 is the ace of spades\n";

      prize = card3; //copy this card, to remember it
      cout << "I'm swapping card 1 and card 3\n";
      temp = card3;
      card3 = card1;
      card1 = temp;

      cout << "I'm swapping card 2 and card 3\n";
      temp = card3;
      card3 = card2;
      card2 = temp;

      cout << "I'm swapping card 1 and card 2\n";
      temp = card2; card2 = card1; card1 = temp;

      cout << "Now, where (1, 2, or 3) is the ace of spades? ";
      cin >> position;
      switch (position)
      {
      case 1: chosen = card1; break;
      case 2: chosen = card2; break;
      case 3: chosen = card3; break;
      }
      if(chosen.number == prize.number && chosen.suit == prize.suit)
           cout << "That's right! You win!\n";
      else
           cout << "Sorry. You lose.\n";
      system("pause");
}
/*
Card 1 is the 7 of clubs
Card 2 is the jack of hearts
Card 3 is the ace of spades
I'm swapping card 1 and card 3
I'm swapping card 2 and card 3
I'm swapping card 1 and card 2
Now, where (1, 2, or 3) is the ace of spades? 2
That's right! You win!
Press any key to continue . . .
*/
```