Introduction to Programming

Lecture 10 : Structs

Askar Khaimuldin

Senior-lecturer

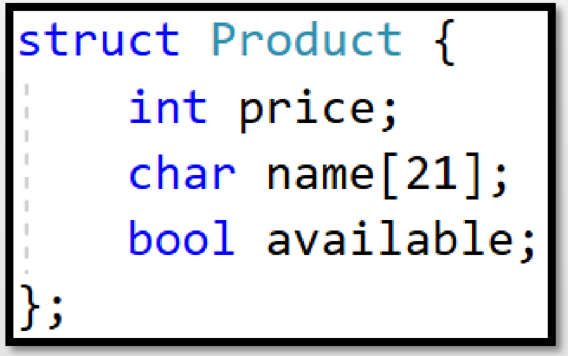
askar.khaimuldin@astanait.edu.kz

Content

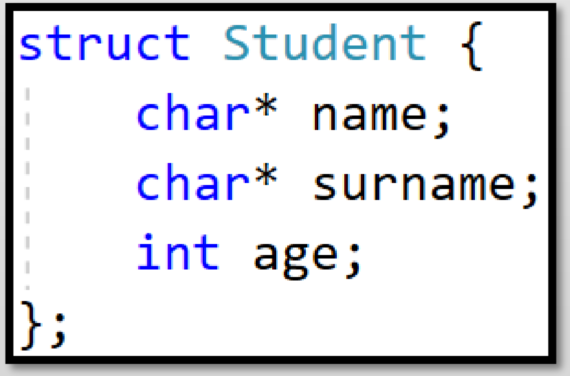
- Structures in C++
- Array of structures
- Structure pointers
- Nested structures
- Structure functions
- Structure Member Alignment
- Bonus topic (template functions)

Structures in C++

- A structure is a set of variables that are referenced under the same name
- It ensures a convenient way to store information as one single unit
- It holds variables of different data types
- All data members are public by default
- All data members of a structure are logically related
- One of the possible ways to create a user-defined type
- It is a blueprint from which instances can be created



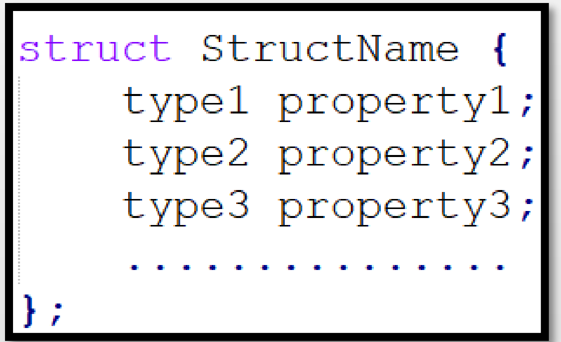
```
struct Product {
    int price;
    char name[21];
    bool available;
};
```



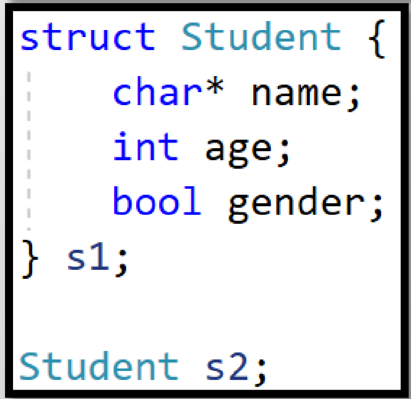
```
struct Student {
    char* name;
    char* surname;
    int age;
};
```

Structures in C++

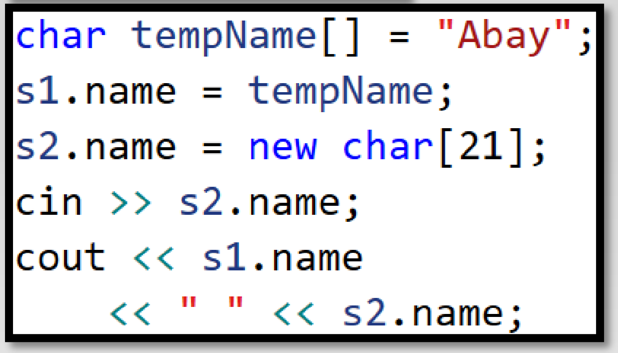
- General form of structure declaration
- A variable of structure can be declared during structure declaration and afterwards
- A structure can be both global and local
- Hint: Every pointer variables inside a structure should point to properly allocated place in memory
- Every member of structure is accessed using dot (.)



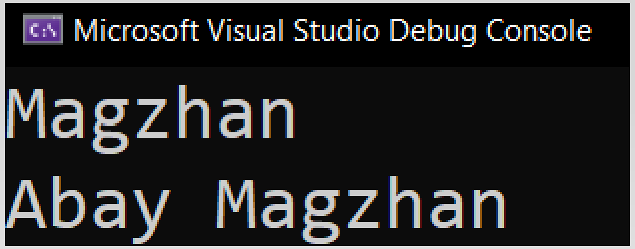
```
struct StructName {
    type1 property1;
    type2 property2;
    type3 property3;
    ...............
};
```



```
struct Student {
    char* name;
    int age;
    bool gender;
} s1;

Student s2;
```

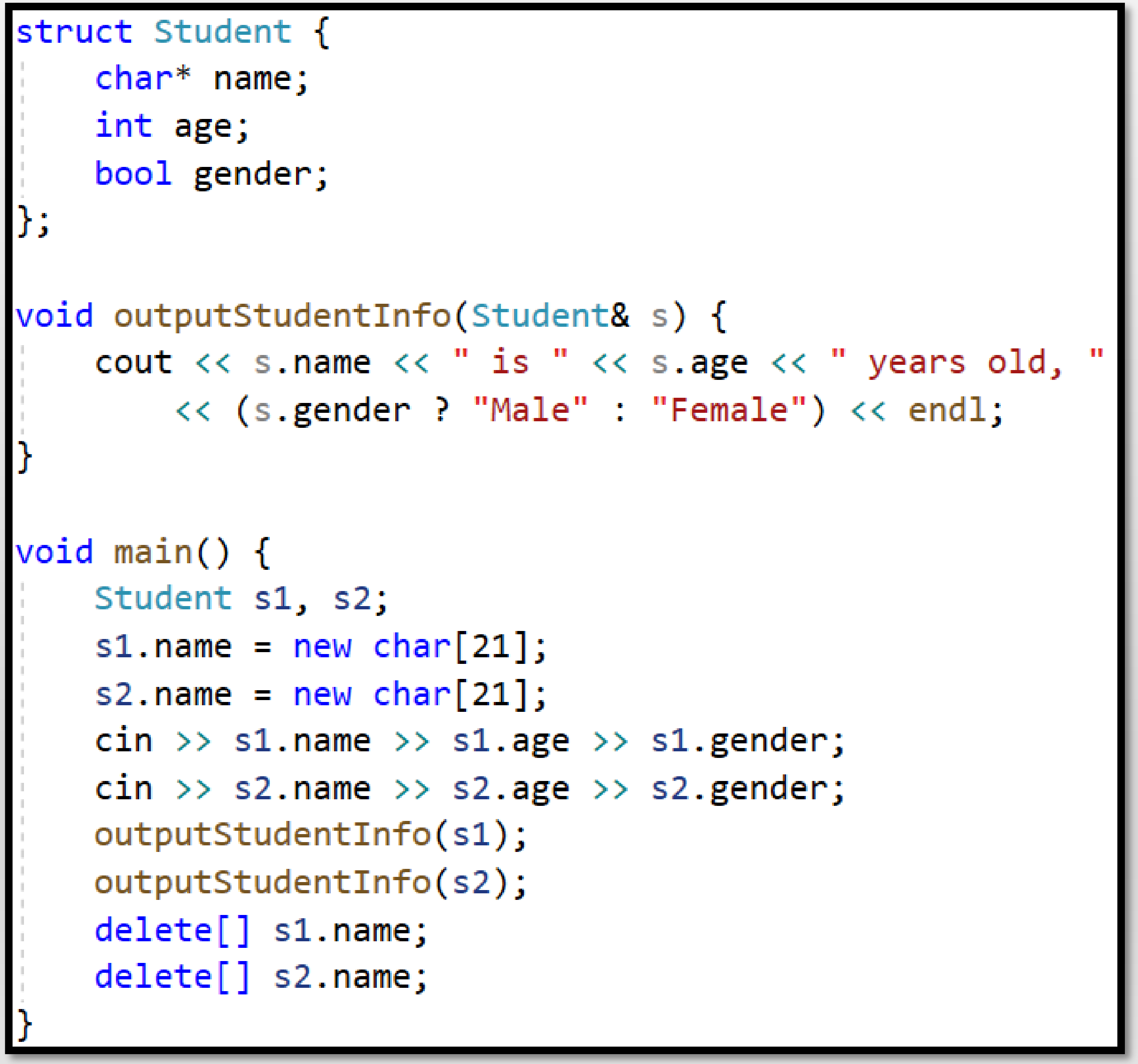


```
char tempName[] = "Abay";
s1.name = tempName;
s2.name = new char[21];
cin >> s2.name;
cout << s1.name
    << " " << s2.name;
```

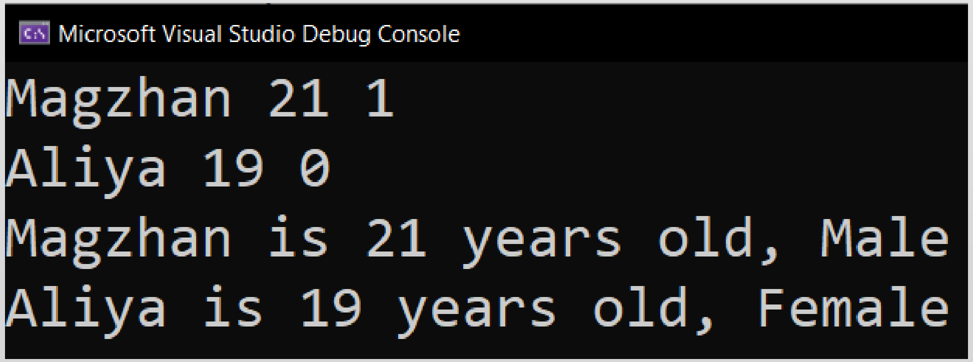


```
Magzhan
Abay Magzhan
```

Structures in C++ (Example)



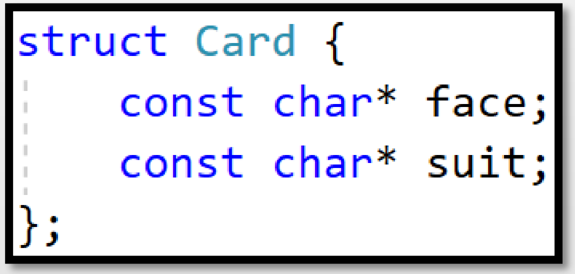
```cpp
struct Student {
    char* name;
    int age;
    bool gender;
};

void outputStudentInfo(Student& s) {
    cout << s.name << " is " << s.age << " years old, "
        << (s.gender ? "Male" : "Female") << endl;
}

void main() {
    Student s1, s2;
    s1.name = new char[21];
    s2.name = new char[21];
    cin >> s1.name >> s1.age >> s1.gender;
    cin >> s2.name >> s2.age >> s2.gender;
    outputStudentInfo(s1);
    outputStudentInfo(s2);
    delete[] s1.name;
    delete[] s2.name;
}
```



```
Magzhan 21 1
Aliya 19 0
Magzhan is 21 years old, Male
Aliya is 19 years old, Female
```

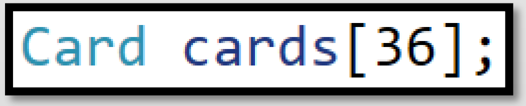
Array of structures

- The most common usage of structures is in arrays of structures



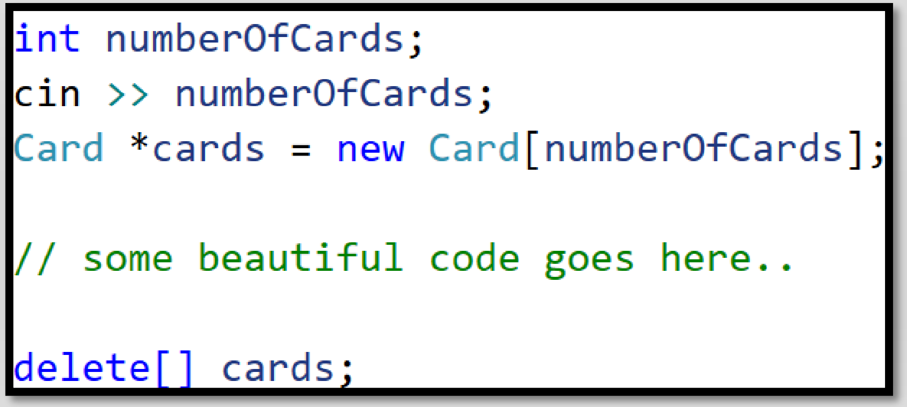
```
struct Card {
    const char* face;
    const char* suit;
};
```

- To declare an array of structures, you must first define a structure and then declare an array variable of that type



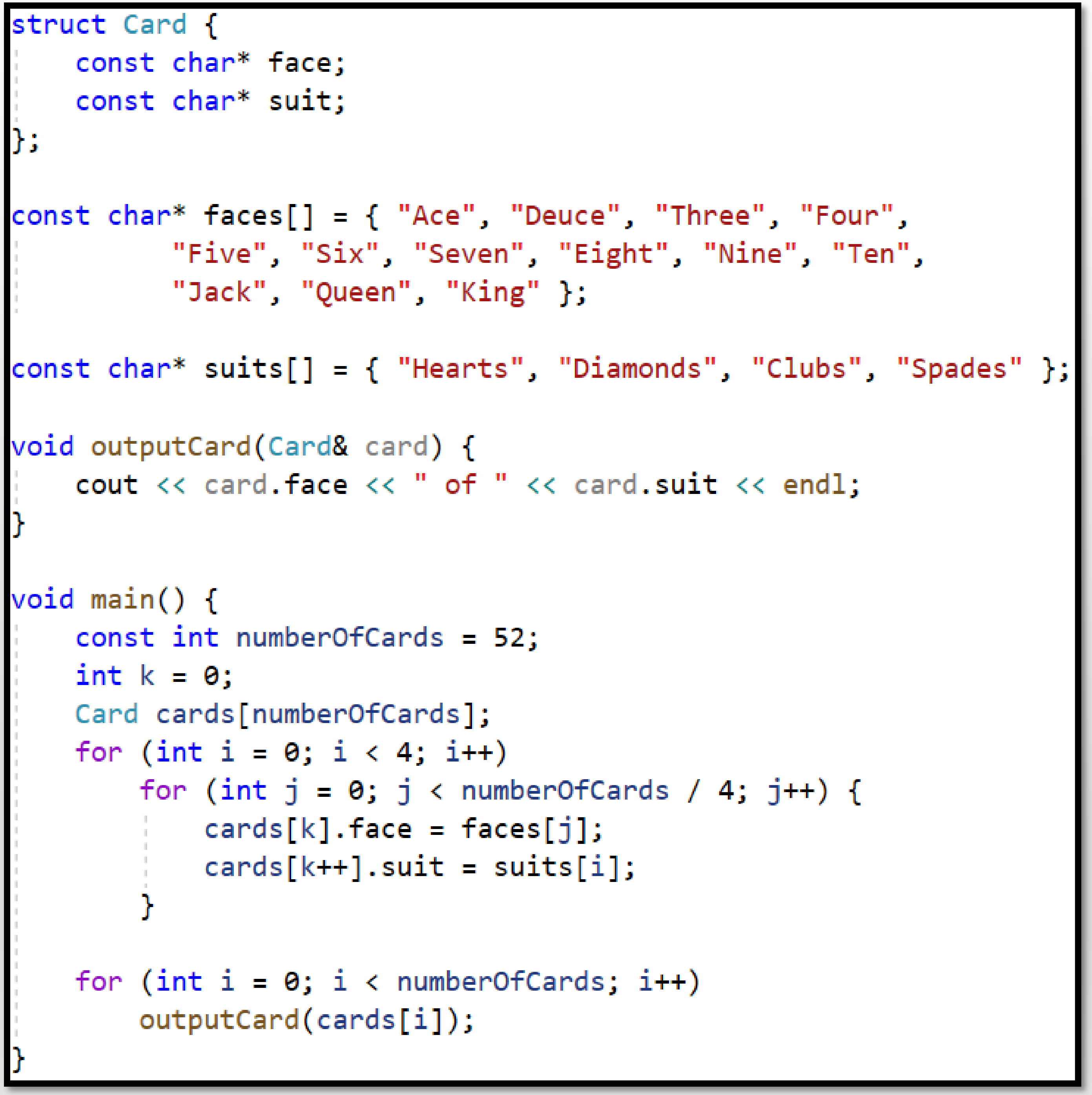
```
Card cards[36];
```

- It is possible to create a dynamic array of structures

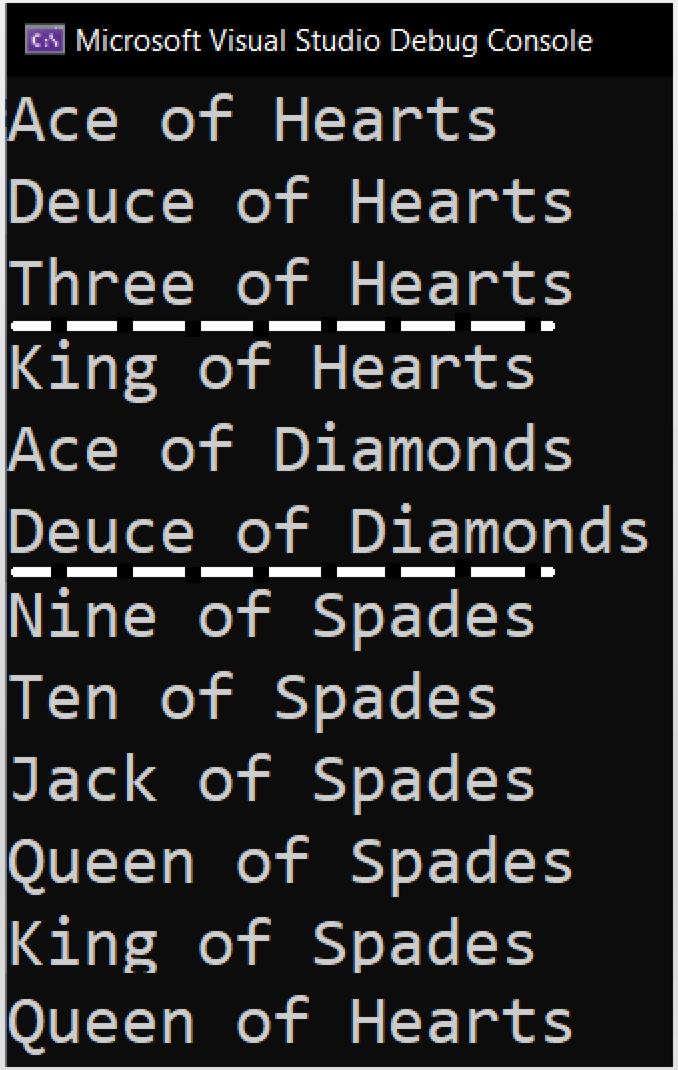


```
int numberOfCards;
cin >> numberOfCards;
Card *cards = new Card[numberOfCards];

// some beautiful code goes here..

delete[] cards;
```

- Hint: If dynamic memory was used for any of the structure members, don`t forget to free that memory



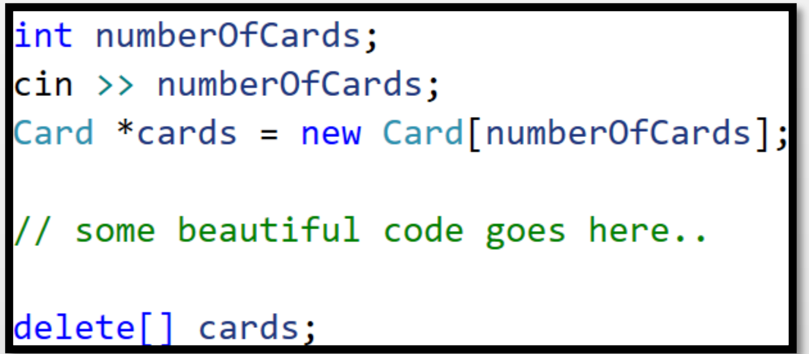
```cpp
struct Card {
    const char* face;
    const char* suit;
};

const char* faces[] = { "Ace", "Deuce", "Three", "Four",
          "Five", "Six", "Seven", "Eight", "Nine", "Ten",
          "Jack", "Queen", "King" };

const char* suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };

void outputCard(Card& card) {
    cout << card.face << " of " << card.suit << endl;
}

void main() {
    const int numberOfCards = 52;
    int k = 0;
    Card cards[numberOfCards];
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < numberOfCards / 4; j++) {
            cards[k].face = faces[j];
            cards[k++].suit = suits[i];
        }

    for (int i = 0; i < numberOfCards; i++)
        outputCard(cards[i]);
}
```



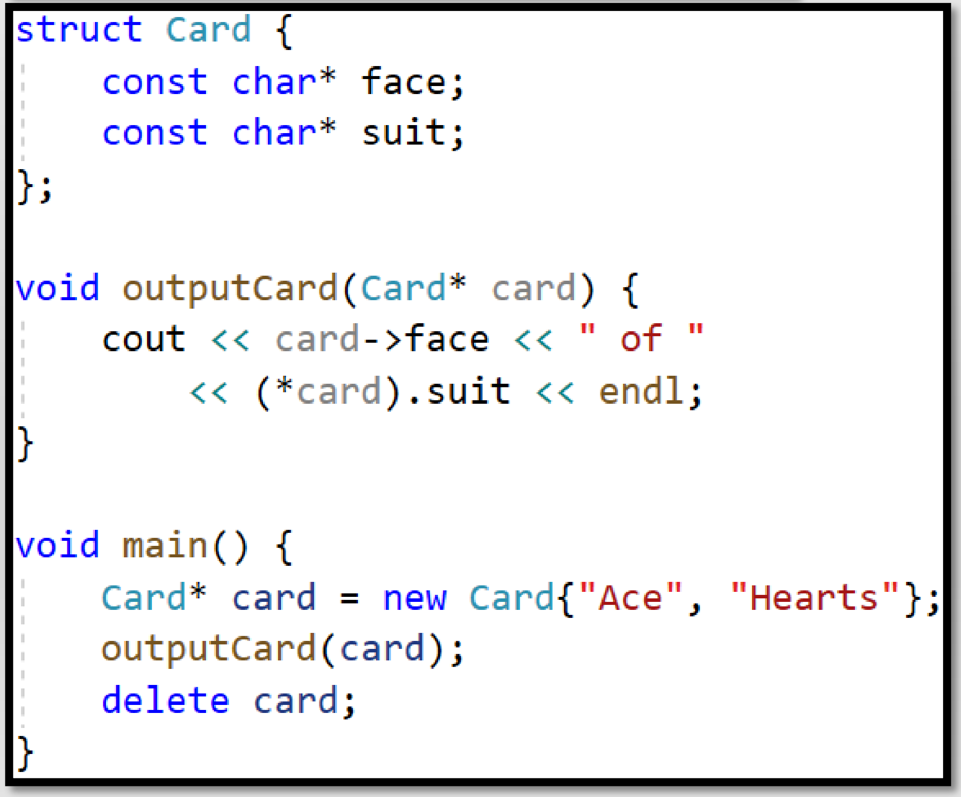
```
Ace of Hearts
Deuce of Hearts
Three of Hearts
King of Hearts
Ace of Diamonds
Deuce of Diamonds
Nine of Spades
Ten of Spades
Jack of Spades
Queen of Spades
King of Spades
Queen of Hearts
```

Structure pointers

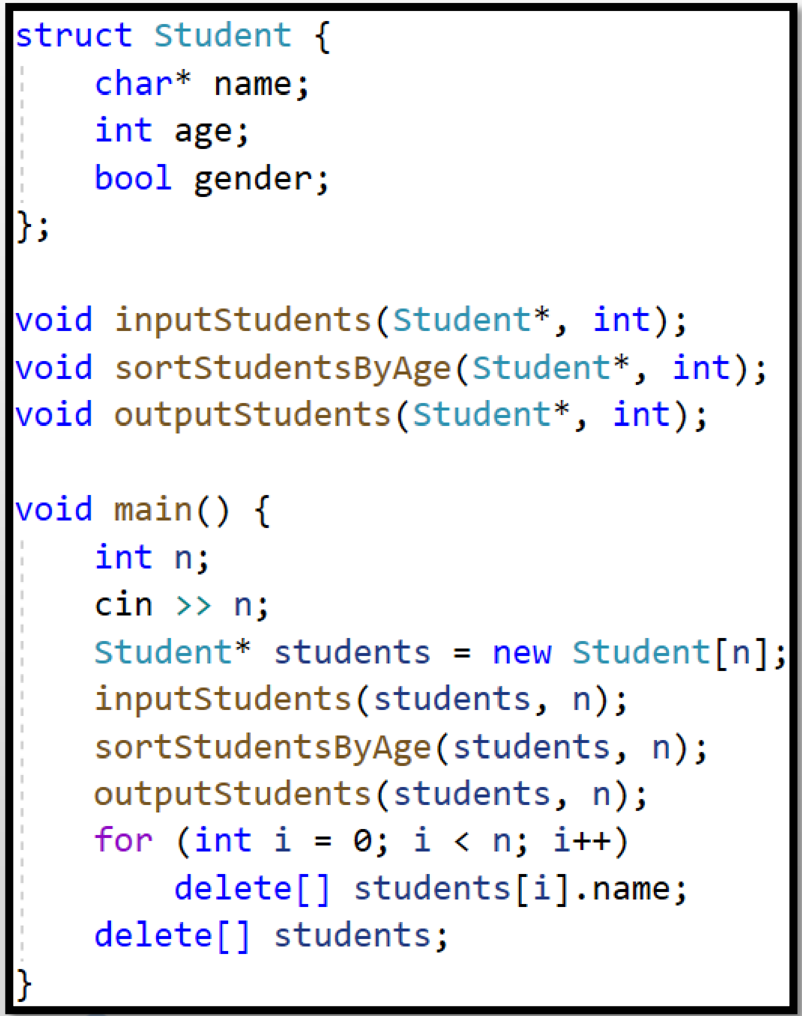
- Structure pointers are useful in two primary cases:
  - Passing a structure to a function
  - Dynamic allocation
- It is possible to create a dynamic array of structures
- All data members are accessed using arrow operator (->) or direct dereferencing
- Hint: If dynamic memory was used for any of the structure members, don`t forget to free that memory



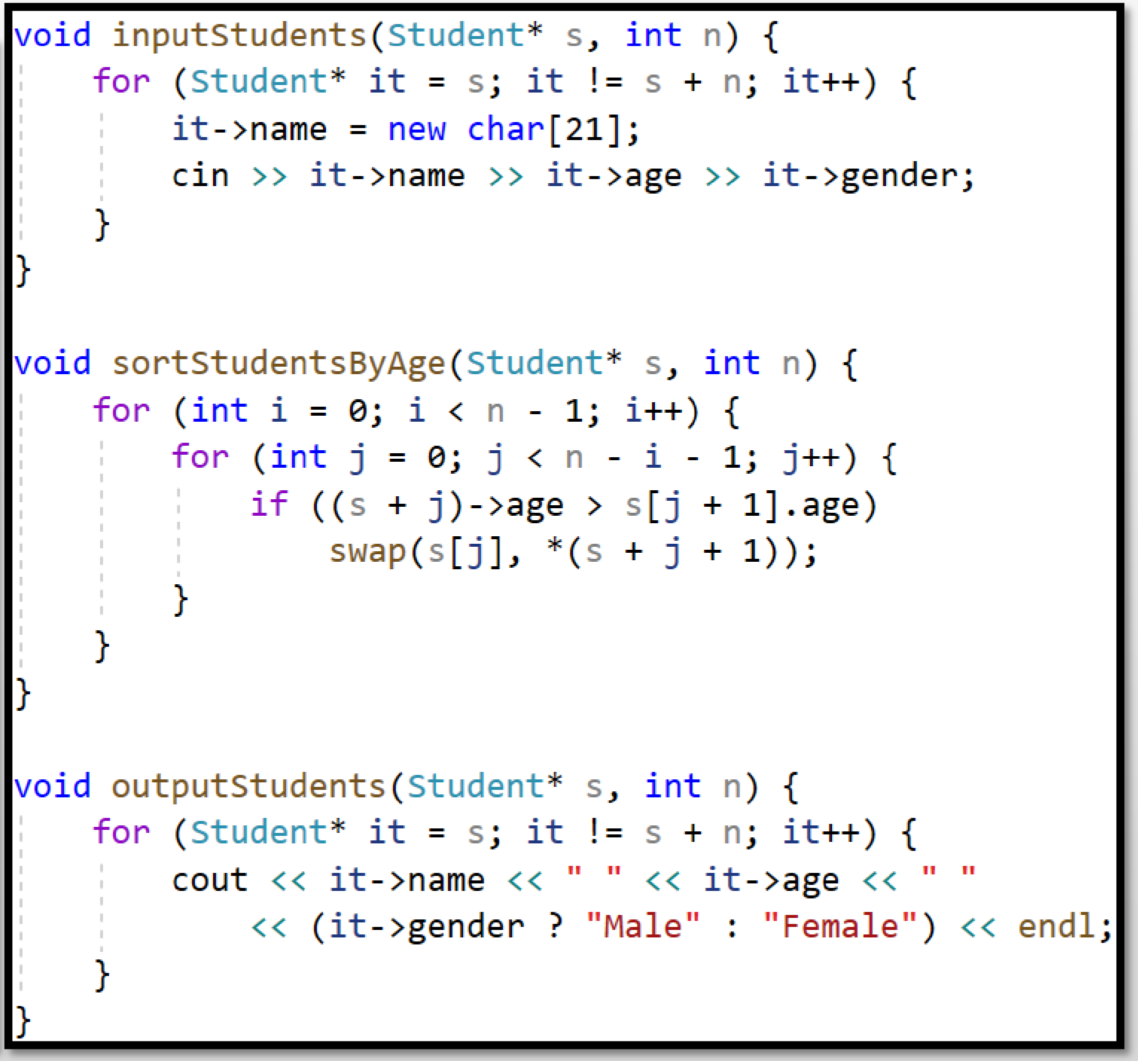
```
int numberOfCards;
cin >> numberOfCards;
Card *cards = new Card[numberOfCards];

// some beautiful code goes here..

delete[] cards;
```



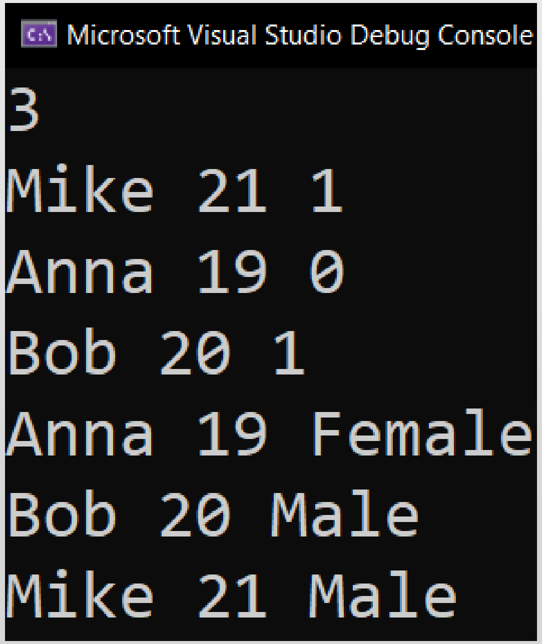
```
struct Card {
    const char* face;
    const char* suit;
};

void outputCard(Card* card) {
    cout << card->face << " of "
        << (*card).suit << endl;
}

void main() {
    Card* card = new Card{"Ace", "Hearts"};
    outputCard(card);
    delete card;
}
```



```cpp
struct Student {
    char* name;
    int age;
    bool gender;
};

void inputStudents(Student*, int);
void sortStudentsByAge(Student*, int);
void outputStudents(Student*, int);

void main() {
    int n;
    cin >> n;
    Student* students = new Student[n];
    inputStudents(students, n);
    sortStudentsByAge(students, n);
    outputStudents(students, n);
    for (int i = 0; i < n; i++)
        delete[] students[i].name;
    delete[] students;
}
```



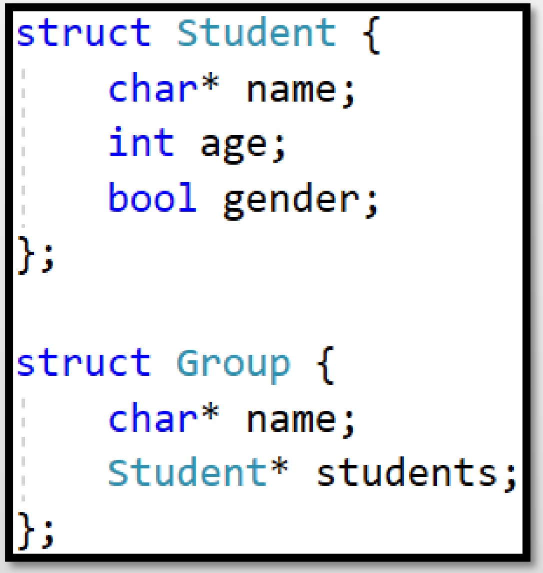
```cpp
void inputStudents(Student* s, int n) {
    for (Student* it = s; it != s + n; it++) {
        it->name = new char[21];
        cin >> it->name >> it->age >> it->gender;
    }
}

void sortStudentsByAge(Student* s, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if ((s + j)->age > s[j + 1].age)
                swap(s[j], *(s + j + 1));
        }
    }
}

void outputStudents(Student* s, int n) {
    for (Student* it = s; it != s + n; it++) {
        cout << it->name << " " << it->age << " "
            << (it->gender ? "Male" : "Female") << endl;
    }
}
```



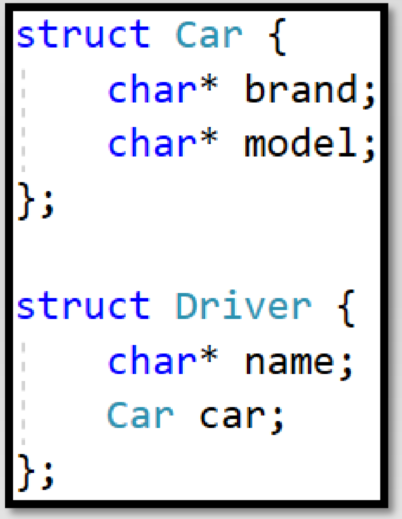
```
3
Mike 21 1
Anna 19 0
Bob 20 1
Anna 19 Female
Bob 20 Male
Mike 21 Male
```

Nested structures

- A structure can contain an instance of another structure (association)
- An order of structures` declarations matters
- Example 1: A student can have a name, his/her age and gender, whereas a group can have group name and a set of students
- Example 2: A car can have a brand name and model name, whereas a driver can have own name and a car



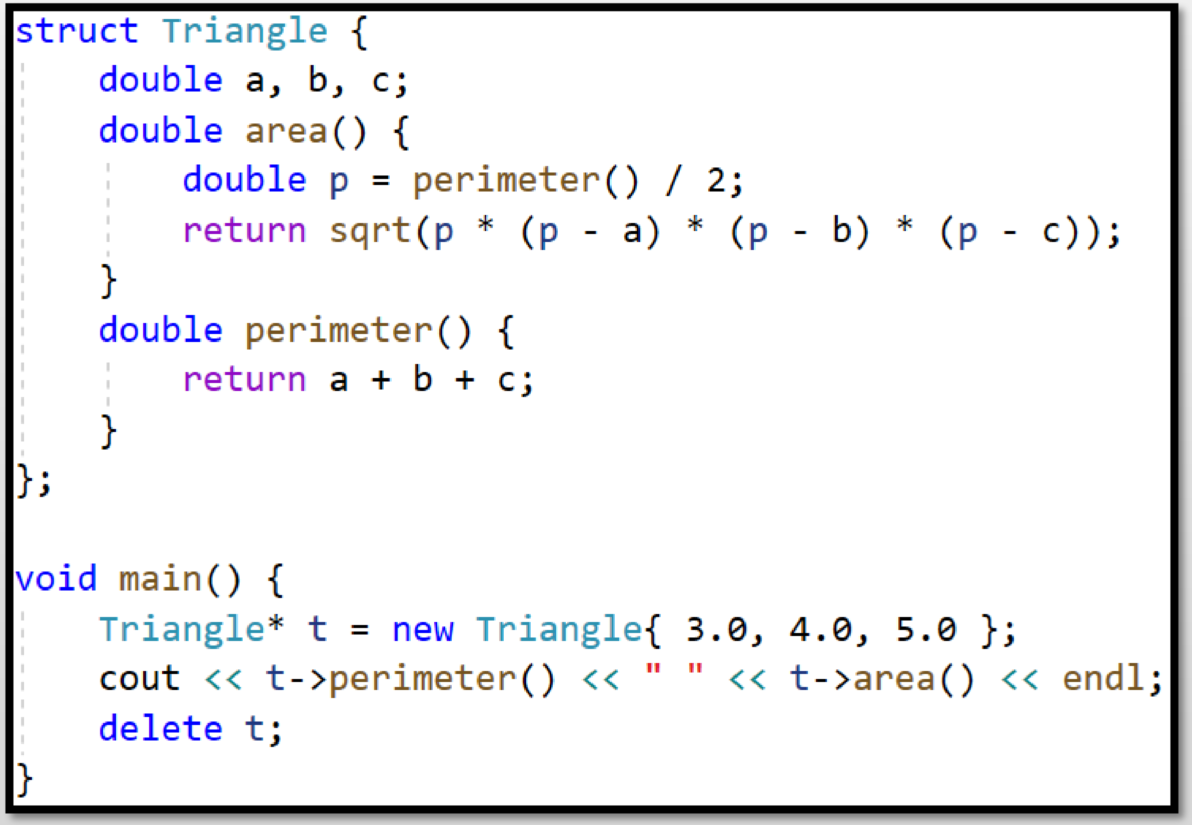
```
struct Student {
    char* name;
    int age;
    bool gender;
};

struct Group {
    char* name;
    Student* students;
};
```



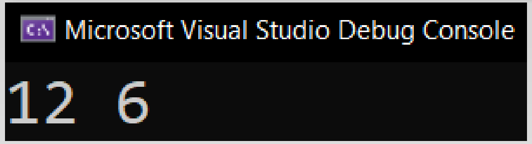
```
struct Car {
    char* brand;
    char* model;
};

struct Driver {
    char* name;
    Car car;
};
```

Structure functions

- A structure can contain **member functions** (structure functions)
- All member functions are accessible through instance only
- Structure function is such a function that logically connected to its structure
- A member function has direct access to all data members and other member functions of its structure
- It defines an instance behavior



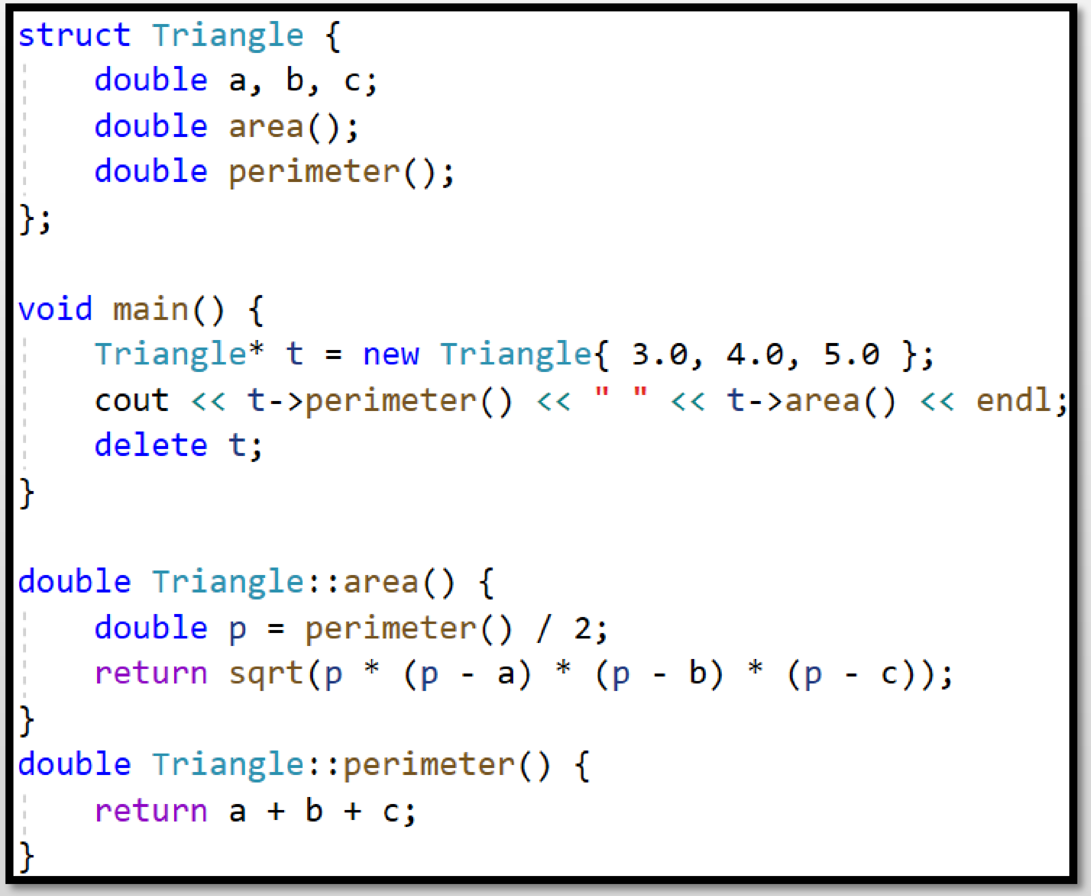
```
struct Triangle {
    double a, b, c;
    double area() {
        double p = perimeter() / 2;
        return sqrt(p * (p - a) * (p - b) * (p - c));
    }
    double perimeter() {
        return a + b + c;
    }
};

void main() {
    Triangle* t = new Triangle{ 3.0, 4.0, 5.0 };
    cout << t->perimeter() << " " << t->area() << endl;
    delete t;
}
```



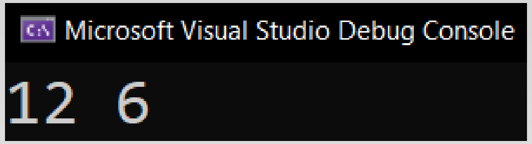
```
12 6
```

Structures (continued)

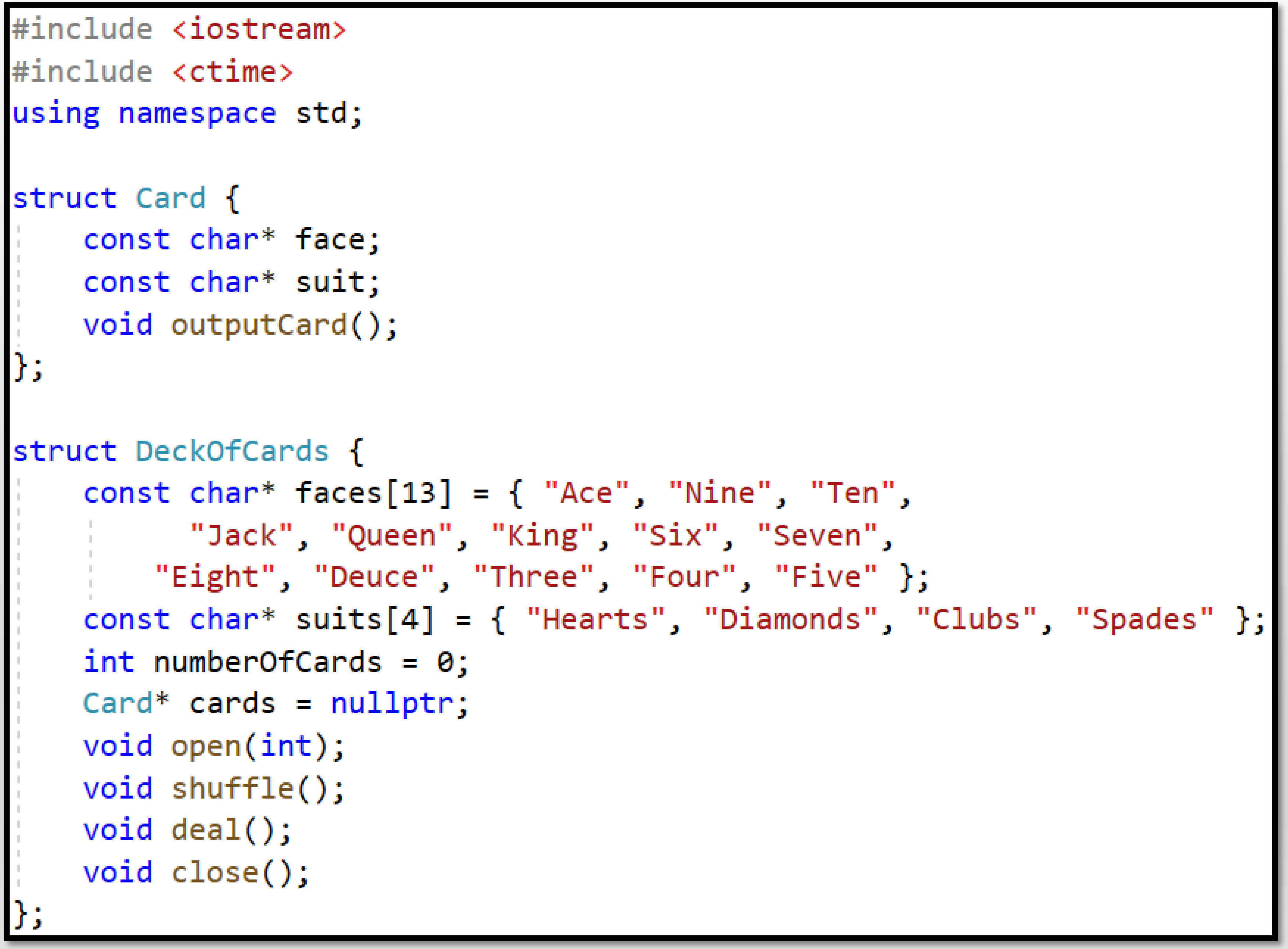
- All data members are initialized to their default values
- All member functions can be declared without their definitions
- A separate definition of member function can only be written using **Scope Operator (::)**



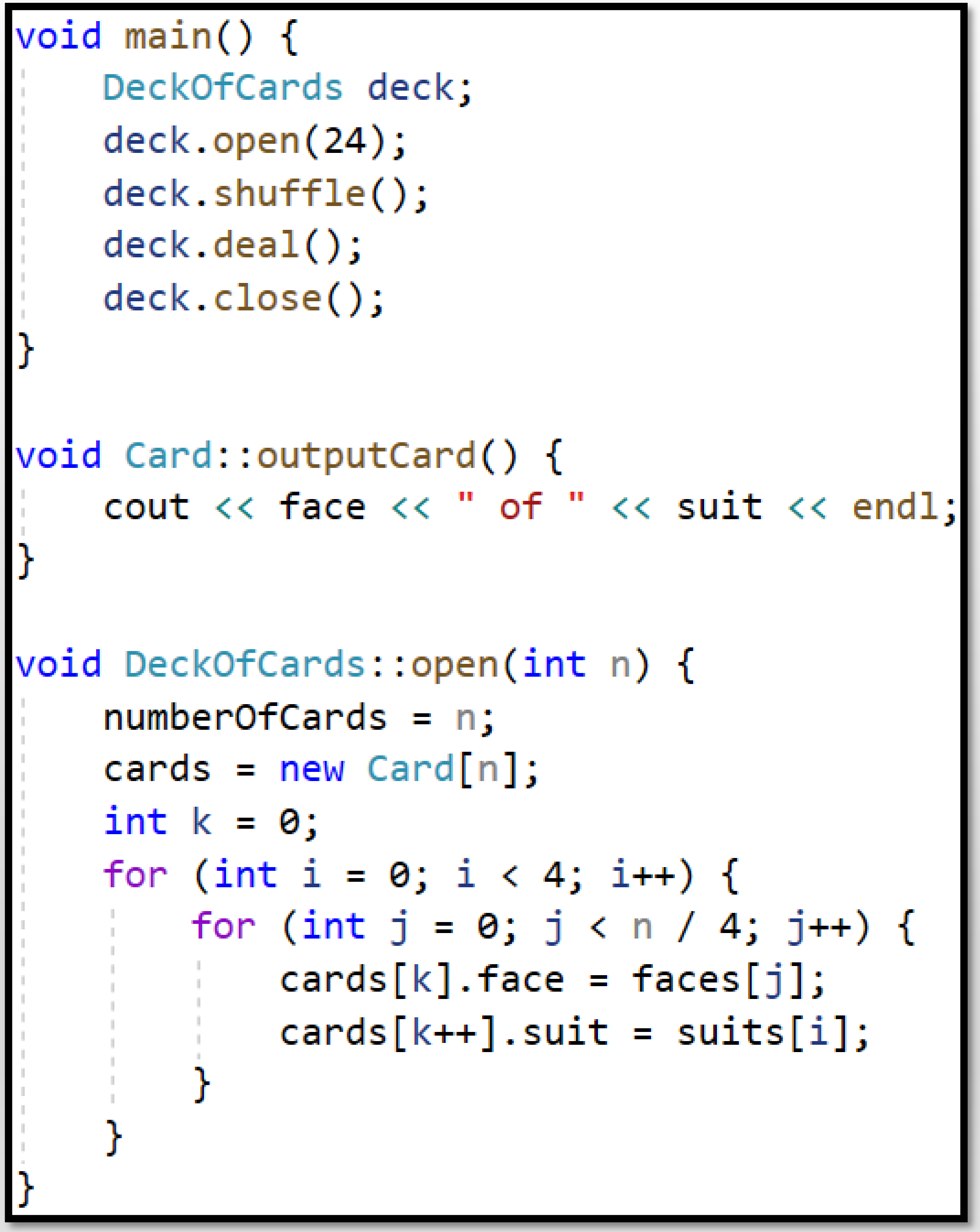
```cpp
struct Triangle {
    double a, b, c;
    double area();
    double perimeter();
};

void main() {
    Triangle* t = new Triangle{ 3.0, 4.0, 5.0 };
    cout << t->perimeter() << " " << t->area() << endl;
    delete t;
}

double Triangle::area() {
    double p = perimeter() / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}
double Triangle::perimeter() {
    return a + b + c;
}
```



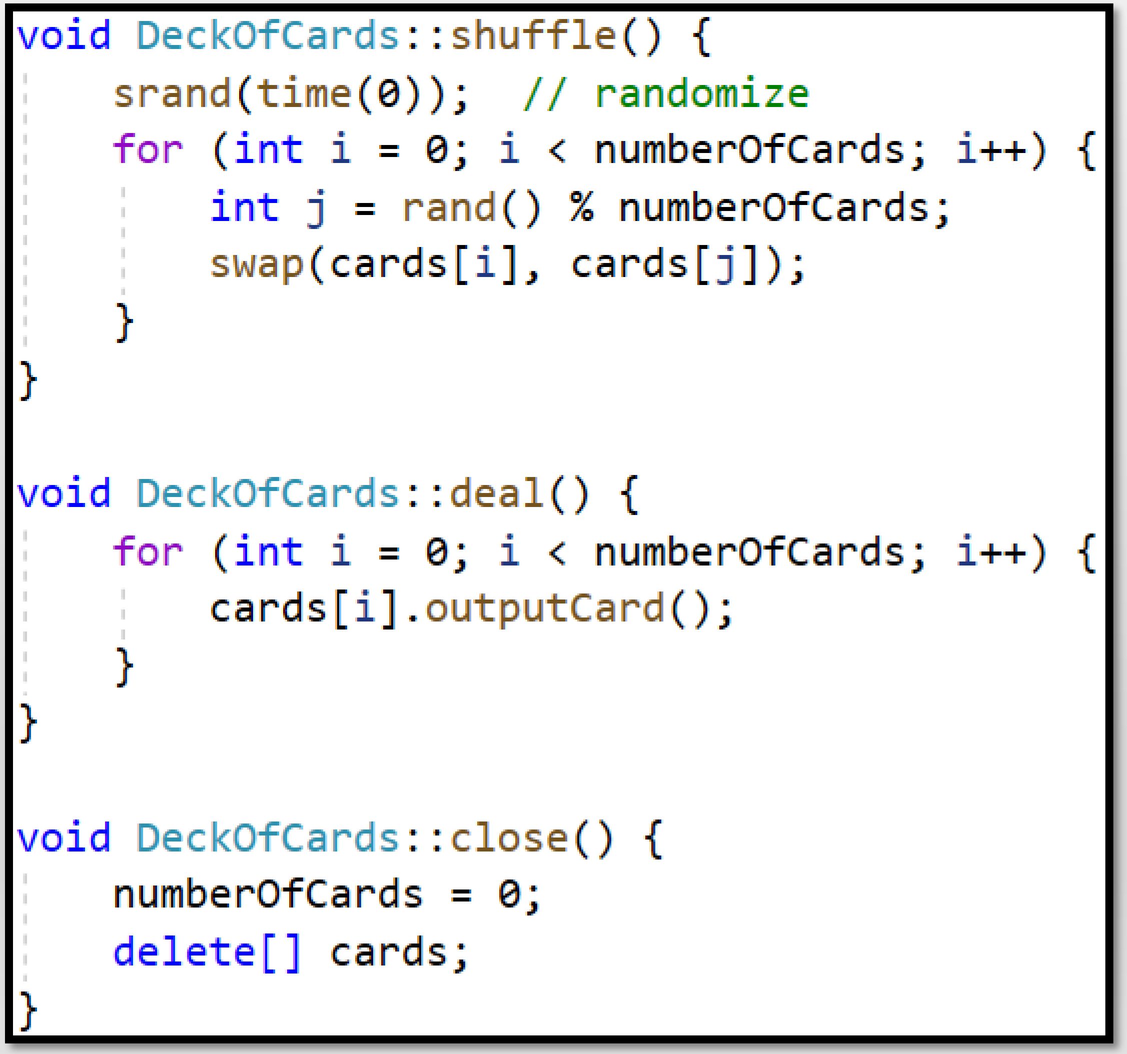
```
12 6
```



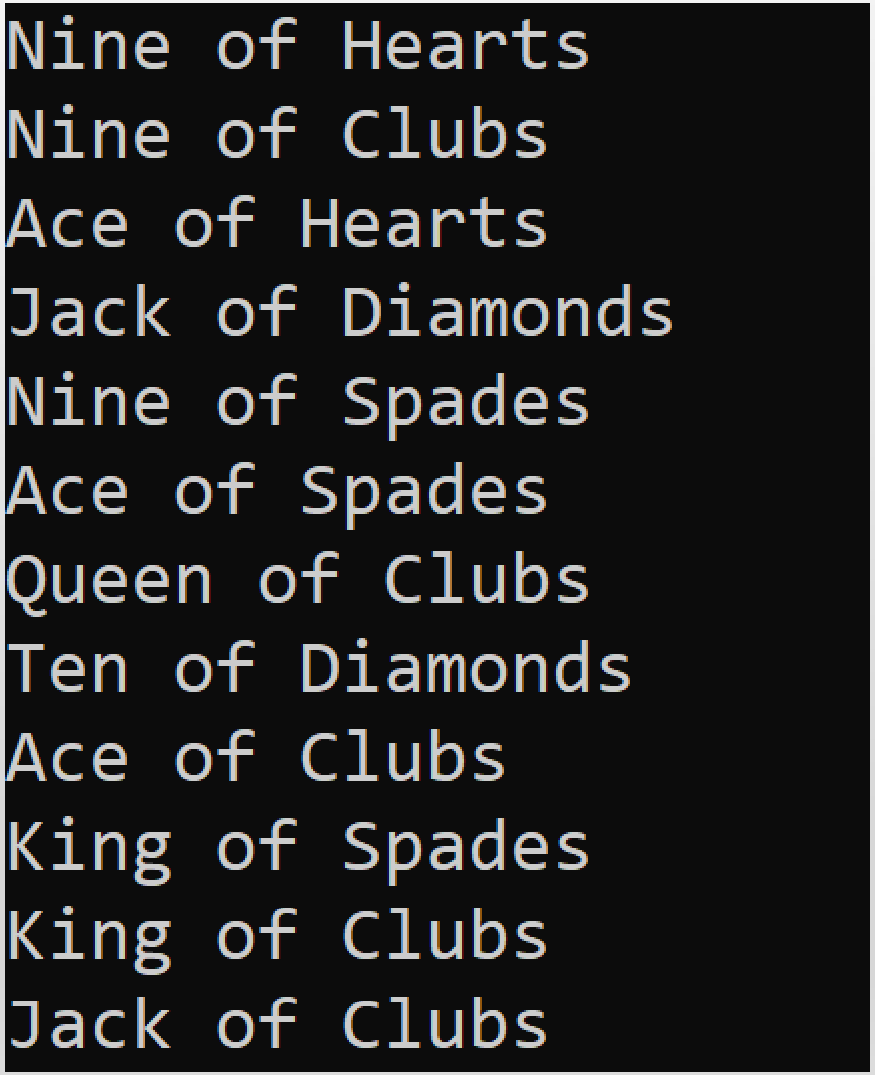
```cpp
#include <iostream>
#include <ctime>
using namespace std;

struct Card {
    const char* face;
    const char* suit;
    void outputCard();
};

struct DeckOfCards {
    const char* faces[13] = { "Ace", "Nine", "Ten",
          "Jack", "Queen", "King", "Six", "Seven",
        "Eight", "Deuce", "Three", "Four", "Five" };
    const char* suits[4] = { "Hearts", "Diamonds", "Clubs", "Spades" };
    int numberOfCards = 0;
    Card* cards = nullptr;
    void open(int);
    void shuffle();
    void deal();
    void close();
};
```



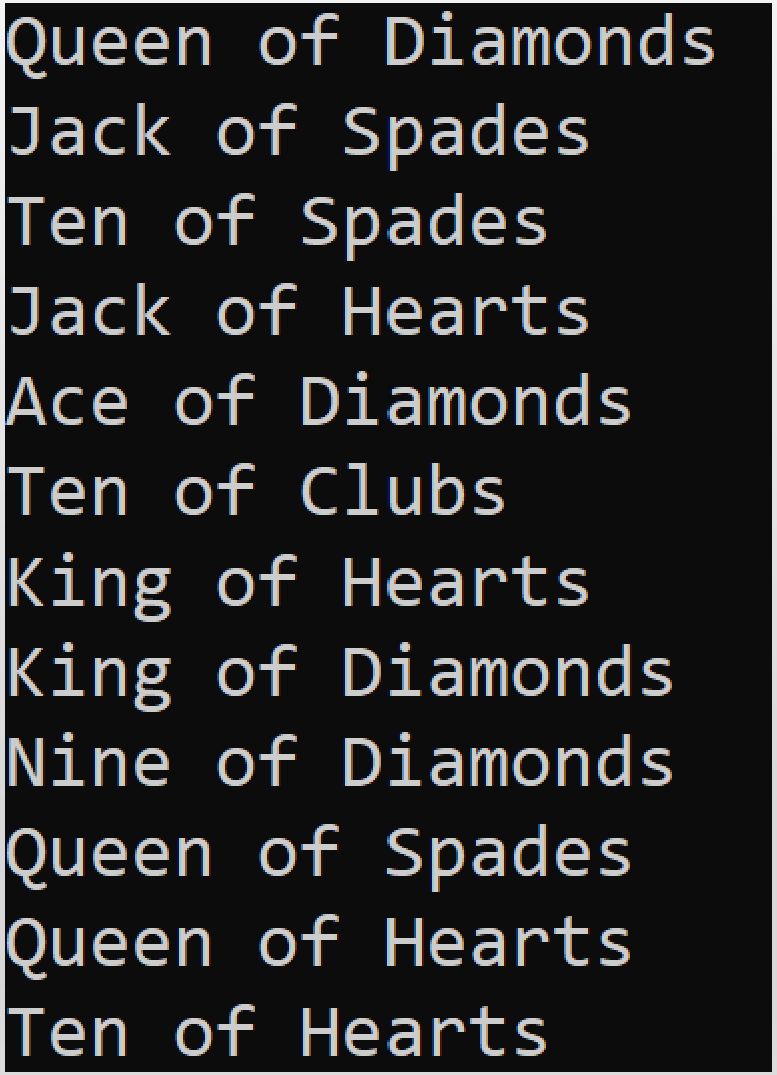
```cpp
void main() {
    DeckOfCards deck;
    deck.open(24);
    deck.shuffle();
    deck.deal();
    deck.close();
}

void Card::outputCard() {
    cout << face << " of " << suit << endl;
}

void DeckOfCards::open(int n) {
    numberOfCards = n;
    cards = new Card[n];
    int k = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < n / 4; j++) {
            cards[k].face = faces[j];
            cards[k++].suit = suits[i];
        }
    }
}
```



```cpp
void DeckOfCards::shuffle() {
    srand(time(0));  // randomize
    for (int i = 0; i < numberOfCards; i++) {
        int j = rand() % numberOfCards;
        swap(cards[i], cards[j]);
    }
}

void DeckOfCards::deal() {
    for (int i = 0; i < numberOfCards; i++) {
        cards[i].outputCard();
    }
}

void DeckOfCards::close() {
    numberOfCards = 0;
    delete[] cards;
}
```



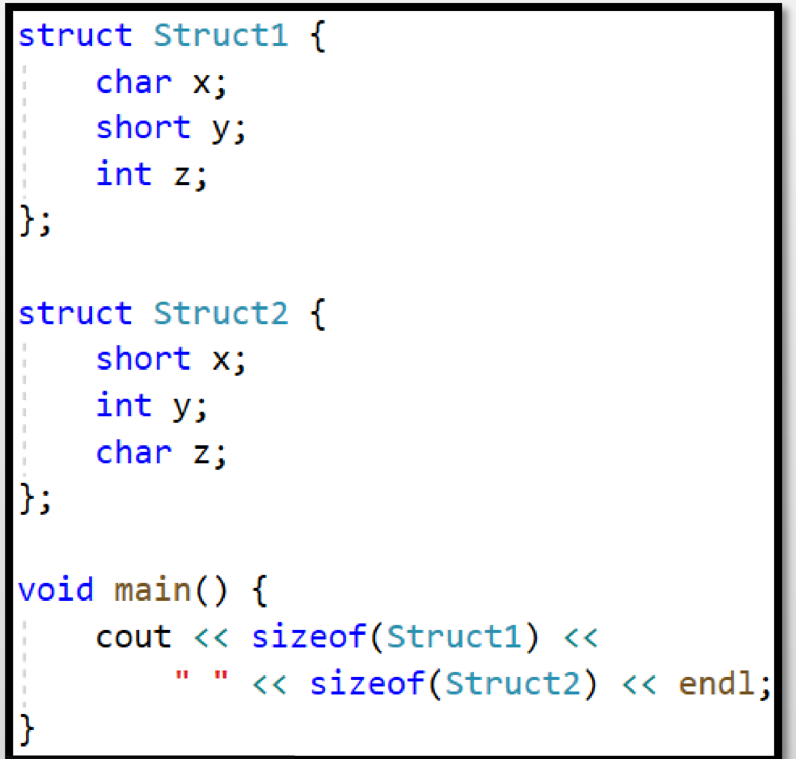
```
Nine of Hearts
Nine of Clubs
Ace of Hearts
Jack of Diamonds
Nine of Spades
Ace of Spades
Queen of Clubs
Ten of Diamonds
Ace of Clubs
King of Spades
King of Clubs
Jack of Clubs
```



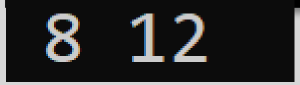
```
Queen of Diamonds
Jack of Spades
Ten of Spades
Jack of Hearts
Ace of Diamonds
Ten of Clubs
King of Hearts
King of Diamonds
Nine of Diamonds
Queen of Spades
Queen of Hearts
Ten of Hearts
```

Structure Member Alignment

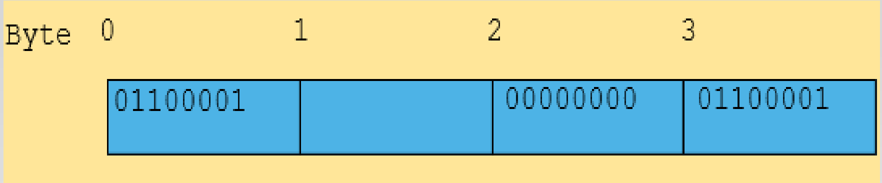
- Every data type in C/C++ will have alignment requirement
- Structure may not be in consecutive bytes of memory
- Byte-alignment (2 or 4 bytes) may cause "holes“
- Suppose 2-byte boundary for structure members
  - Use structure with a char and a short
  - char in first byte
  - short on a 2-byte boundary
  - Value in 1-byte hole undefined
- The task is to decrease the number of holes
- A type with maximum size is increasing step



```
struct Struct1 {
    char x;
    short y;
    int z;
};

struct Struct2 {
    short x;
    int y;
    char z;
};

void main() {
    cout << sizeof(Struct1) <<
        " " << sizeof(Struct2) << endl;
}
```



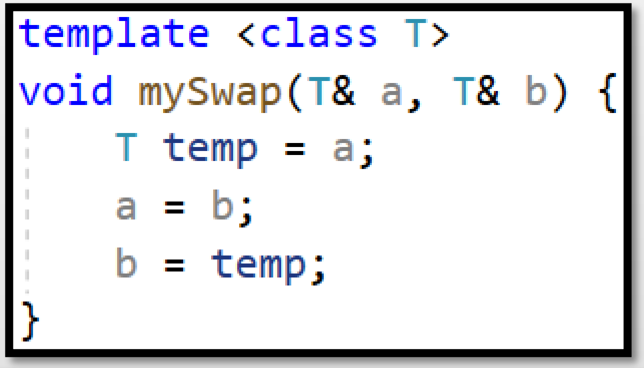
```
8 12
```



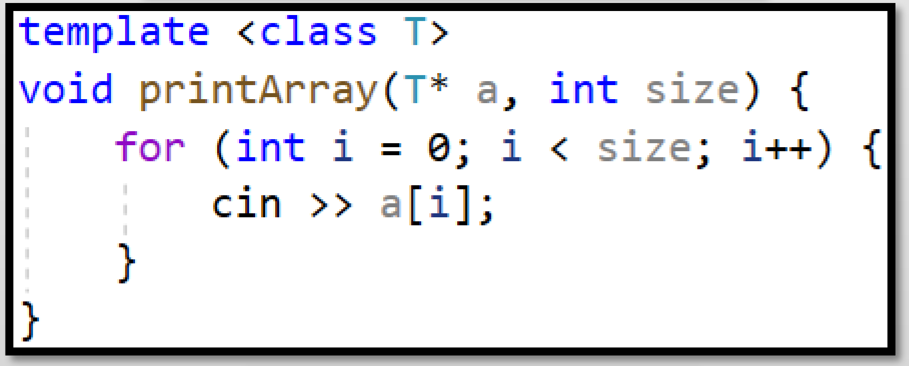
Byte	0	1	2	3
	01100001		00000000	01100001

Bonus topic (template functions)

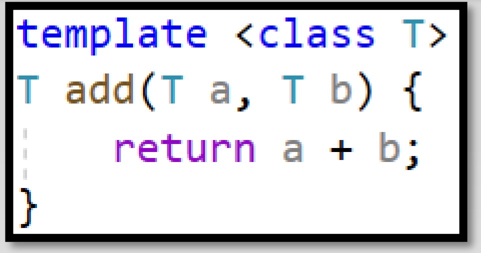
- In C++, a template is a special tool used for operating with generic types
- Templates are expanded at compiler time
- It eliminates writing the same code for different data types
- General form:
  - template <typename T, typename P, …> where T, P, etc. are just fictitious names for incoming types
  - or template <class T, class P, …> (newer version)
- The datatype for T is set at compile time
- Example functions: sort(), swap(), max, etc.



```
template <class T>
void mySwap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```



```
template <class T>
void printArray(T* a, int size) {
    for (int i = 0; i < size; i++) {
        cin >> a[i];
    }
}
```



```
template <class T>
T add(T a, T b) {
    return a + b;
}
```

Literature

- Deitel P.J. and Deitel H.M. 2017. C++ How to Program, 10th global edition (Chapter 22.2)
- Herbert Schildt. 2003. The Complete Reference C++, 4th edition (Chapter 7)
- Bonus topic : Herbert Schildt. 2003. The Complete Reference C++, 4th edition (Chapter 18)

Good luck