§8 Structures

ENGG1111

Computer Programming and Applications

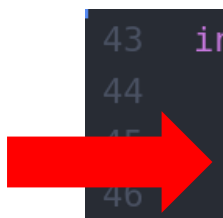
Dirk Schnieders

Outline

- Structures
- Structures and Functions
- Structures and Arrays
- Nested Structures
- Object Oriented Programming
  - For reference only (will not appear in exam)

Motivation

- In our supermarket application, we employed two arrays to store the product information



```
43  int main() {
44    const int numberOfProducts = 10;
45    string name[numberOfProducts] = {};
46    double price[numberOfProducts] = {};
47    readProducts(name, price, numberOfProducts);
48    double total = 0;
49    char input = 'm';
50    while (input == 'm') {
51      total += purchase(name, price, numberOfProducts);
52      cout << "Enter 'm' to purchase more! --> ";
53      cin >> input;
54    }
55    cout << "Total: $" << total << endl;
56    createReceipt(total);
57  }
```

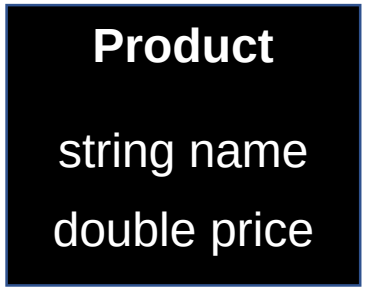
Motivation

- To access the information for a single product we had to access two different arrays

```
26  int productSelection(string name[], double price[], int numberOfProducts) {
27    for (int i=0;i<numberOfProducts;i++)
28      cout << i <<": "<< name[i] <<" ($" << price[i] << ")" << endl;
29    cout << "--> ";
30    int productID;
31    cin >> productID;
32    if (productID >= 0 && productID < numberOfProducts)
33      return productID;
34    return productSelection(name, price, numberOfProducts);
35  }
```

Motivation

- To group related variables together, we can define our own Product type variable
- A struct can be used for this purpose



Don’t forget the ;

```
1  #include <iostream>
2  using namespace std;
3  struct Product {
4    string name;
5    double price;
6  };
7  int main() {
8  }
```

Motivation

- We can create an ‘instance’ of the product as follows

```
Product p1;
```

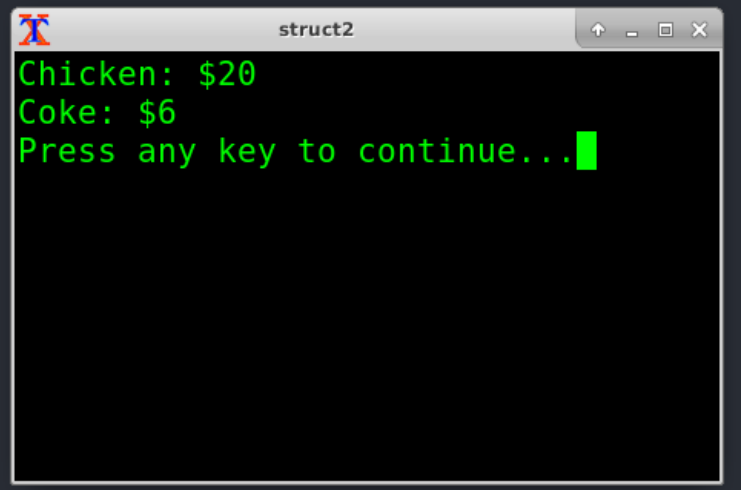
- Use a ‘.’ to access the members of a struct

```
p1.name = "Chicken";
p1.price = 20.0;
```

Motivation

- Example

```cpp
#include <iostream>
using namespace std;
struct Product {
  string name;
  double price;
};
int main() {
  Product p1, p2;
  p1.name = "Chicken";
  p1.price = 20.0;
  p2.name = "Coke";
  p2.price = 6.0;
  cout << p1.name << ": $" << p1.price << endl;
  cout << p2.name << ": $" << p2.price << endl;
}
```



Structure

- A structure is a collection of one or more variables grouped together under a single name
- The data elements in a structure are known as its member variables
  - The members can be of different types
- Structures help to organize complex data
  - Allow a group of related variables to be treated as a unit instead of separate entities

Definition

- In C++, a structure is defined using the keyword struct
  - Followed by a name
  - Followed by a list of member variables (with types and name) enclosed within a pair of curly braces {}
  - And end with a semicolon
- Once a structure type is defined, its structure name can be used to declare variables of this structure type

Definition

- Example

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {
4    string name;
5    string position;
6    double salary;
7  };
8  int main() {
9    Employee e;
10 }
```

Initialization

- A structure variable can be initialized in an initialization list when it is declared

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {
4    string name;
5    string position;
6    double salary;
7  };
8  int main() {
9    Employee e = {"Dirk", "Lecturer", 20000};
10 }
```

Member Variables

- A member variable can be used in any expression like other regular variables of the basic types

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {…
8  int main() {
9    Employee e = {"Dirk", "Lecturer", 20000};
10   e.salary *= 2;
11   e.name.insert(e.name.length(), " Schnieders");
12 }
```

Assignment Operator

- The assignment operator = works for structure variables
  - It copies the variable

```
#include <iostream>
using namespace std;
struct Product {
  string name;
  double price;
};
int main() {
  Product p1 = {"Chicken", 20};
  Product p2 = p1;
  cout << p2.name << endl;
  p2.name = "Coke";
  cout << p2.name << endl;
  cout << p1.name << endl;
}
```

Other Operators

- Other operators like +, -, *, /, >, <, ==, etc. won't work for structure variables
  - E.g., Operator == does know how to compare two structs

```
1  #include <iostream>
2  using namespace std;
3  struct Product {…
7  int main() {
8    Product p1 = {"Chicken", 20};
9    Product p2 = {"Coke", 6};
10   if (p1 == p2)
11     cout << "test" << endl;
12 }
```

```
gcc-make-run: Compile Error

struct7.cpp: In function 'int main()':
struct7.cpp:10:10: error: no match for 'operator==' (operand types are 'Product' and 'Product')
   if (p1 == p2)
       ~~~^~~~
In file included from /usr/include/c++/7/iosfwd:40:0,
                 from /usr/include/c++/7/ios:38,
                 from /usr/include/c++/7/ostream:38,
                 from /usr/include/c++/7/iostream:39,
                 from struct7.cpp:1:
/usr/include/c++/7/bits/postypes.h:216:5: note: candidate: template<class _StateT> bool std::operator==(const std::fpos<_StateT>&, const std::fpos<_StateT>&)
     operator==(const fpos<_StateT>& __lhs, const fpos<_StateT>& __rhs)
     ^~~~~~~~
```

Structures and Functions

Structures and Functions

- Structure variables can be
  1. Returned by a function
  2. Passed to a function
     - either by value or by reference like regular variables

Structures and Functions

- Example 1

```
#include <iostream>
using namespace std;
struct Employee {
  string name;
  string position;
  double salary;
};
Employee createEmployee() {
  Employee e;
  cout << "Name: ";
  cin >> e.name;
  cout << "Position: ";
  cin >> e.position;
  cout << "Salary: ";
  cin >> e.salary;
  return e;
}
```

Structures and Functions

- Example 2 – passed by value

```
 1  #include <iostream>
 2  using namespace std;
 3 › struct Employee {⋯
 8 › Employee createEmployee() {⋯
18  void outputEmployee(Employee e) {
19    cout << "Name: " << e.name << endl;
20    cout << "Position: " << e.position << endl;
21    cout << "Salary: " << e.salary << endl;
22  }
23  int main() {
24    outputEmployee(createEmployee());
25  }
```

Structures and Functions

- Example 2 – passed by reference

```
 1  #include <iostream>
 2  using namespace std;
 3 › struct Employee {⋯
 8 › Employee createEmployee() {⋯
18 › void outputEmployee(Employee e) {⋯
23  void salaryRaise(Employee &e, double factor) {
24    e.salary *= factor;
25  }
26  int main() {
27    Employee e = createEmployee();
28    salaryRaise(e, 1.05);
29    outputEmployee(e);
30  }
```

Structures and Arrays

Structures and Arrays

- We can define an array of structure variables
  - It can be initialized with an initialization list
  - Each element of the array is a struct variable, which can be accessed using its index with the subscript operator []

Structures and Arrays

- Example

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {
8  int main() {
9    Employee a[] = {
10     {"Dirk", "Sales", 20000},
11     {"George", "Supervisor", 40000},
12     {"Loretta", "Manager", 60000}
13   };
14   for (int i;i<3;i++)
15     cout << a[i].name << " " << a[i].position << endl;
16 }
```

Example - Supermarket

- Let's update our supermarket application

```
1   #include <iostream>
2   #include <fstream>
3   using namespace std;
4   struct Product {
5     string name;
6     int id;
7     double price;
8   };
9   void createReceipt(double total) {…
22  int readProducts(Product products[]) {…
32  int productSelection(Product products[], int numberOfProducts) {…
42  double purchase(Product products[], int numberOfProducts) {…
49  int main() {…
```

Example – Supermarket - Continued

```
49  int main() {
50    const int maxNumberOfProducts = 100;
51    Product products[maxNumberOfProducts];
52    int numberOfProducts = readProducts(products);
53    double total = 0;
54    char input = 'm';
55    while (input == 'm') {
56      total += purchase(products, numberOfProducts);
57      cout << "Enter 'm' to purchase more! --> ";
58      cin >> input;
59    }
60    cout << "Total: $" << total << endl;
61    createReceipt(total);
62  }
```

Example – Supermarket - Continued

```
 9 › void createReceipt(double total) {⋯
22   int readProducts(Product products[]) {
23     ifstream fin;
24     fin.open("products.txt");
25     if (fin.fail()) cout << "Error" << endl;
26     int i=0;
27     while (fin >> products[i].name >> products[i].price)
28       i++;
29     fin.close();
30     return i;
31   }
32 › int productSelection(Product products[], int numberOfProducts) {⋯
42 › double purchase(Product products[], int numberOfProducts) {⋯
49 › int main() {⋯
```

Example – Supermarket - Continued

```
 9 › void createReceipt(double total) {⋯
22 › int readProducts(Product products[]) {⋯
32 › int productSelection(Product products[], int numberOfProducts) {⋯
42   double purchase(Product products[], int numberOfProducts) {
43     int productID = productSelection(products, numberOfProducts);
44     int quantity;
45     cout << "How many? --> ";
46     cin >> quantity;
47     return products[productID].price * quantity;
48   }
49 › int main() {⋯
```

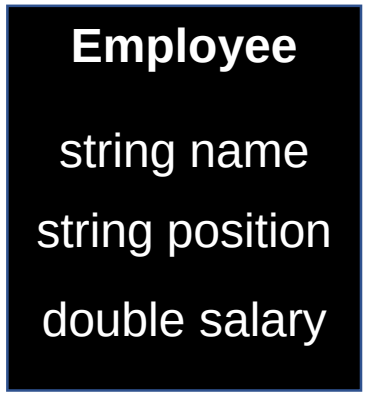
Example – Supermarket - Continued

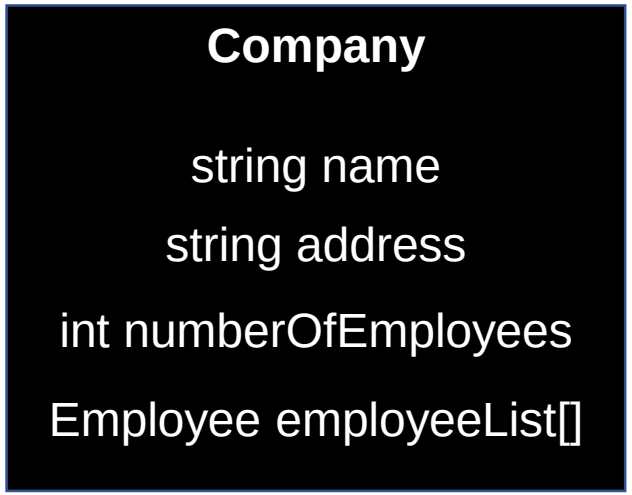
```
 9 › void createReceipt(double total) {⋯
22 › int readProducts(Product products[]) {⋯
32   int productSelection(Product products[], int numberOfProducts) {
33     for (int i=0;i<numberOfProducts;i++)
34       cout << i <<": "<< products[i].name <<" ($" << products[i].price << ")" << endl
35     cout << "--> ";
36     int productID;
37     cin >> productID;
38     if (productID >= 0 && productID < numberOfProducts)
39       return productID;
40     return productSelection(products, numberOfProducts);
41   }
```

Nested Structures

Nested Structures

- Structures can be nested
  - A member of a structure can be another structure
- Example





Nested Structures - Example

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {
4    string name;
5    string position;
6    double salary;
7  };
8  struct Company {
9    string name;
10   string address;
11   int numberOfEmployees;
12   Employee employeeList[100];
13 };
14 void addEmployee(Company &c) {
24 void outputCompany(Company c) {
34 int main() {
```

Nested Structures – Example - Continued

```
 1  #include <iostream>
 2  using namespace std;
 3 › struct Employee {⋯
 8 › struct Company {⋯
14 › void addEmployee(Company &c) {⋯
24 › void outputCompany(Company c) {⋯
34  int main() {
35    Company c1 = {"ABC company", "HK", 0};
36    addEmployee(c1);
37    addEmployee(c1);
38    Company c2 = {"XYZ company", "US", 0};
39    addEmployee(c2);
40    outputCompany(c1);
41    outputCompany(c2);
42  }
```

Nested Structures – Example - Continued

```
 1  #include <iostream>
 2  using namespace std;
 3 › struct Employee {⋯
 8 › struct Company {⋯
14  void addEmployee(Company &c) {
15    cout << "Adding employee to " << c.name << endl;
16    cout << "Name: ";
17    cin >> c.employeeList[c.numberOfEmployees].name;
18    cout << "Position: ";
19    cin >> c.employeeList[c.numberOfEmployees].position;
20    cout << "Salary: ";
21    cin >> c.employeeList[c.numberOfEmployees].salary;
22    c.numberOfEmployees++;
23  }
24 › void outputCompany(Company c) {⋯
34 › int main() {⋯
```

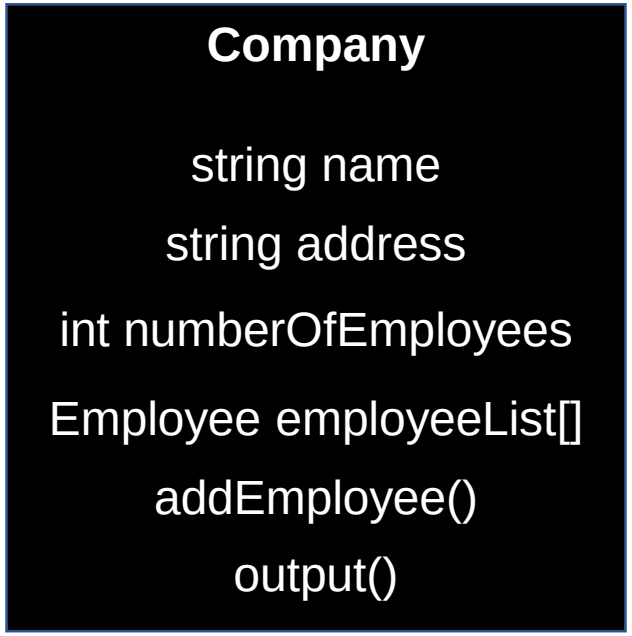
Nested Structures – Example - Continued

```
 1  #include <iostream>
 2  using namespace std;
 3 › struct Employee {⋯
 8 › struct Company {⋯
14 › void addEmployee(Company &c) {⋯
24  void outputCompany(Company c) {
25    cout << "Name: " << c.name << endl;
26    cout << "Address: " << c.address << endl;
27    cout << "Number of Employees: " << c.numberOfEmployees << endl;
28    for (int i=0;i<c.numberOfEmployees;i++) {
29      cout << c.employeeList[i].name << " ";
30      cout << c.employeeList[i].position << " ";
31      cout << "$" << c.employeeList[i].salary << endl;
32    }
33  }
34 › int main() {⋯
```

Object Oriented Programming
(for reference only)

OOP

- In object oriented programming (OOP) we add functions related to the structure directly into the struct
- Example



OOP - Example

```
1  #include <iostream>
2  using namespace std;
3  struct Employee {…
8  struct Company {
9    string name;
10   string address;
11   int numberOfEmployees;
12   Employee employeeList[100];
13   void addEmployee() {…
23   void outputCompany() {…
33 };
34 int main() {
35   Company c1 = {"ABC company", "HK", 0};
36   c1.addEmployee();
37   c1.addEmployee();
38   Company c2 = {"XYZ company", "US", 0};
39   c2.addEmployee();
40   c1.outputCompany();
41   c2.outputCompany();
```

OOP – Example - Continued

```
#include <iostream>
using namespace std;
struct Employee {…
struct Company {
  string name;
  string address;
  int numberOfEmployees;
  Employee employeeList[100];
  void addEmployee() {
    cout << "Adding employee to " << name << endl;
    cout << "Name: ";
    cin >> employeeList[numberOfEmployees].name;
    cout << "Position: ";
    cin >> employeeList[numberOfEmployees].position;
    cout << "Salary: ";
    cin >> employeeList[numberOfEmployees].salary;
    numberOfEmployees++;
  }
  void outputCompany() {…
};
int main() {…
```

OOP – Example - Continued

```
 2  using namespace std;
 3 › struct Employee {⋯
 8  struct Company {
 9    string name;
10    string address;
11    int numberOfEmployees;
12    Employee employeeList[100];
13 ›  void addEmployee() {⋯
23    void outputCompany() {
24      cout << "Name: " << name << endl;
25      cout << "Address: " << address << endl;
26      cout << "Number of Employees: " << numberOfEmployees << endl;
27      for (int i=0;i<numberOfEmployees;i++) {
28        cout << employeeList[i].name << " ";
29        cout << employeeList[i].position << " ";
30        cout << "$" << employeeList[i].salary << endl;
31      }
32    }
33  };
34 › int main() {⋯
```

OOP

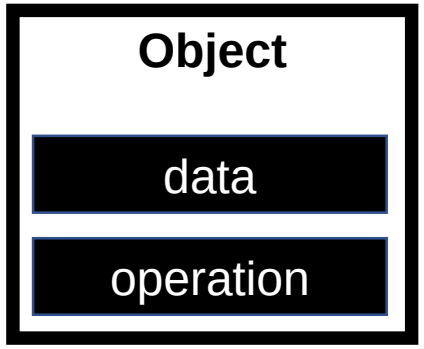
- OOP is a programming paradigm using objects
- Traditionally, programming languages have divided the world into two parts

data

operation

OOP

- In OOP, this view is restructured to a higher level
  - Operations and data are grouped into modular units called objects
  - Every object has both state (data, i.e., variables) and behavior (operations - i.e., functions on data)



Objects

- Objects in OOP are not much different from ordinary physical objects
- It is easy to see how a mechanical device, such as a pocket watch or a piano, embodies both state and behavior
- But almost anything that is designed to do a job does, too

Objects

- Even a simple bottle combines state
  - How full the bottle is
  - Weight
  - Open/closed
  - Temperature of content
- With behavior
  - Dispense content
  - Open/close
- It is this resemblance to real things objects much of their power

Task

- Write a function that will compare its two parameters of the following struct type

```
3  struct Entry {
4    string name;
5    char dorm;
6    int age;
7  };
```

- The function will return true if the member variables of the structure contain the same data and false otherwise