"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Object oriented programming

Exercises 2

Version 1.0, 24 February, 2017

Table of Contents

1. Introduction in C++ 1
   - 1.1. Introduction 1
   - 1.2. Input/Output streams in C++ `<iostream>` 1
   - 1.3. Declaration of variables in C++ 2
   - 1.4. `inline` functions 2
   - 1.5. Additional keywords in C++ 3
   - 1.6. Using `typedef` 3
   - 1.7. Functions as struct members 3
   - 1.8. Usage of function as struct member 4
2. References 4
   - 2.1. Definition and declaration 4
   - 2.2. Differences and similarities with pointers 5
   - 2.3. Usage 5
   - 2.4. Example arguments struct reference 5
3. New casting syntax 6
4. Solutions of problems with structures in C++ 7
5. Source code of the examples and problems 10

1. Introduction in C++

1.1. Introduction

- C++ includes many characteristics of the C programming language enriched with object-oriented flavour (OOP)
- C++ is developed at Bell laboratories and at the beginning was called "C with classes"
- The name C is increment of C () and this means that C++ is extended C
- C++ compiler can be used to compile C programs

1.2. Input/Output streams in C++ `<iostream>`

In C++ to work with input/output streams instead of functions `printf` and `scanf` we use the operators:

- `<<` from the object `cout` (`cout <<`)
- `>>` from the object `cin` (`cin >>`)

Example C

```
printf("Enter new value: ");
scanf("%d", &value);
printf("The new value is: %d\n", value);
```

Example C++

```
cout << "Enter new value: ";
cin >> value;
cout << "The new value is: " << value << '\n';
```

It is recommended to replace the usage `\n` with `endl`:

```
cout << "The new value is: " << value << endl;
```

To work with input/output streams in C++ you should include the header `<iostream>`

Example `oop_av21_en.cpp`

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

int main () {
    cout << "Enter your age: ";
    int myAge ;
    cin >> myAge ;
    cout << "Enter your friend's age: ";
    int friendsAge ;
    cin >> friendsAge ;
    if (myAge > friendsAge)
        cout << "You are older.\n";
    else if (myAge < friendsAge)
        cout << "Your friend is older.\n";
    else
        cout << "You and your friend are same age.\n";
    return 0;
}
```

1.3. Declaration of variables in C++

The variables in C++ can be declared anywhere in the program until their declaration is before the usage.

```
for(int i = 0;  i < 5; i++)
    cout << i << endl;
```

The scope of the local variables begins with the declaration and ends at the end of the block `}`. The declaration can be done in the conditions or in the loop expressions `while`, `do/while`, `for` or `if`.

1.4. `inline` functions

Each standard call of function consumes additional time in the process of calling the function. In C++ while defining small and simple function the keyword `inline` can be used, so each call of this function can be replaced with the body of the function, so the additional overhead of calling the function will be lost.

Example function for computing volume of cube

```
#include <iostream>
using namespace std;
inline float cube(const float s) {
    return s * s * s;
}
int main() {
    cout << "Side of the cube: ";
    float side;
    cin >> side;
    cout << "Volumen of a cube with side " << side << " is: " << cube(side)
            << endl;
    return 0;
}
```

1.5. Additional keywords in C++

asm	explicit	operator
catch	friend	private
class	inline	protected
const_cast	mutable	public
delete	new	reinterpret_cast
dynamic_cast	namespace	static_cast
template	throw	using
this	try	virtual

1.6. Using `typedef`

Using `typedef` is permitted, but not necessary in defining structs, unions or enums) in C++.

Example

```
#include <iostream>
using namespace std;
struct Person {
    char ime[80], adresa[90];
    double plata;
};
int main() {
    Person Vraboten[50]; // pole so elementi od tip struct Person
    Person Rakovoditel;
    Rakovoditel.ime = "Aleksandar";
    cout << "Imeto na rakovoditelot e" << Rakovoditel.ime << endl;
    return 0;
}
```

1.7. Functions as struct members

In C++ functions can be defined inside of a struct.

Example

```
struct Person {
    char ime[80], adresa[80];
    // declaration of function for printing the name and the address
    void print(void);
};
```

- The function `print()` is only declared; the body is in another part of the program.
- The size of the struct (`sizeof`) is determined **only** from the size of the data members. Functions that are declared does not affect its size.
- Access of the function defined as part of the struct is identical as the access of the data members, i.e. using the operator `.` or for pointers `->`.

1.8. Usage of function as struct member

Example

```
#include <iostream>
using namespace std;
struct Person {
    char ime[80], adresa[90];
    double plata;
    void printperson();
};
void Person::printperson() {
    cout << "Imeto na vraboteniot e:" << ime << "a, negovata adresa e:"
            << adresa << endl;
}
int main() {
    Person Rakovoditel;
    //...fali inicijalizacija
    Rakovoditel.printperson();
    return (0);
}
```

2. References

2.1. Definition and declaration

Reference is new data type in C++ similar to the pointer type in C but safer and more convenient for usage.

Declaration

```
<Type> & <Name>
```

Example

```
int A = 5;
int& rA = &A;
```

2.2. Differences and similarities with pointers

References in C++ have these differences with pointers:

- Direct access to the reference after its declaration is not possible, each access is actually access to the variable/object it is referencing.
- Once initialized it can not be dereferenced or refer to other variable/object.
- It can not be NULL (referencing nothing).
- Once declared they must be initialized.

2.3. Usage

One of the most important usage is passing function arguments.

Example

```
int swap(int &a, int &b) {
    a += b;
    b = a - b;
    a -= b;
}
int main() {
    int x = 10;
    int y = 15;
    swap(x, y);
    // x = 15, y = 10
    return 0;
}
```

2.4. Example arguments struct reference

Example

```
// declaration of struct
struct Person {
    char name[80], address[90];
    double sallary;
};
Person employee[50]; // array of Person
// print accepts reference to Person struct
void print(Person const &p) {
    cout << "Name of the employee is: " << p.name << " and his address is: "
            << p.address << endl;
}
// return data of the employee by his index
Person const &getEmployee(int index) {
... return (person[index]); // returns reference
}
int main() {
    Person employee;
    print(employee); // passing is same as is there was no reference
    return (0);
}
```

3. New casting syntax

In C we used the following cast syntax:

```
(typename)expression
(float) nominator;
```

In C++ it's available new notation:

```
typename(expression)
float(nominator);
```

Also there are 4 new ways of casting in C++:

- `static_cast<type>(expression)` - standard static casting
- `const_cast<type>(expression)` - used for modifying the type of constants
- `reintrepret_cast<type>(expression)` - used to reinterpret the information
- `dynamic_cast<type>(expression)` - polymorphism casting

4. Solutions of problems with structures in C++

Solution oop_av24_en.cpp

```
#include <iostream>
#include <cstring> // string.h math.h -> cmath
using namespace std;

struct student {
    char firstName[50];
    char lastName[50];
    int number;
    int totalPoints;
    void print() {
        cout << firstName << "\t";
        cout << lastName << "\t";
        cout << number << "\t";
        cout << totalPoints << "\n";
    }
};


void normalize(char *name, bool allUppercase = false) {
    *name = toupper(*name);
    ++name;
    while(*name) {
        if(allUppercase) {
            *name = toupper(*name);
        } else {
            *name = tolower(*name);
        }
        ++name;
    }
}

/*void normalize(char *name) {
    normalize(name, false);
}
*/
void read(student &s) {
    cin >> s.firstName;
    cin >> s.lastName;
    normalize(s.firstName);
    normalize(s.lastName);
    cin >> s.number;
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    s.totalPoints = a + b + c + d;
}



void sort(student s[], int n) {
    for(int i = 0; i < n; ++i) {
        for(int j = 0; j < n - 1; ++j) {
            if(s[j].totalPoints < s[j + 1].totalPoints) {
                student tmp = s[j];
                s[j] = s[j + 1];
                s[j + 1] = tmp;
            }
        }
    }
}

int main() {
    student s[100];
    int n;
    cin >> n;
    for(int i = 0; i < n; ++i) {
        read(s[i]);
    }
    sort(s, n);
    cout << "======== ORDERED ========" << endl;
    //read(s);
    for(int i = 0; i < n; ++i) {
        s[i].print();
        //print(s[i]);
    }

    return 0;
}
```

Solution `oop_av25_en.cpp`

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

struct city {
    char name[100];
    int population;
};

struct country {
    char name[100];
    char president[100];
    city capital;
    int population;
    void print() {
        cout << name << "\t" << president << "\t";
        cout << capital.name << "\t";
        cout << capital.population << "\t";
        cout << population << endl;
    }
};

void print(country &c) {
    cout << c.name << "\t" << c.president << "\t";
    cout << c.capital.name << "\t";
    cout << c.capital.population << "\t";
    cout << c.population << endl;
}

void read(int n, country c[]) {
    for (int i = 0; i < n; ++i) {
        cin >> c[i].name;
        cin >> c[i].president;
        cin >> c[i].capital.name;
        cin >> c[i].capital.population;
        cin >> c[i].population;
    }
}

int sum(int a = 0, int b = 5) {
    return a + b;
}

void maxCapitalPopulation(int n, country c[]) {
    country max = c[0];
    for (int i = 1; i < n; ++i) {
        if (c[i].capital.population > max.capital.population) {
            max = c[i];
        }
    }

    cout << max.president << endl;
}

int main() {
    country countries[100];
    country &first = countries[0];
    first = countries[1];
    int n;
    cin >> n;
    read(n, countries);
    for (int i = 0; i < n; ++i) {
        //print(countries[i]);
        countries[i].print();
    }

    maxCapitalPopulation(n, countries);
/*    cout << sum(3, 10) << endl;
    cout << sum(10) << endl;
    cout << sum() << endl;*/


    return 0;
}
```

5. Source code of the examples and problems

https://github.com/finki-mk/SP/

Source code ZIP