Въведение в ООП

титуляр на курса: д-р Тодор Цонков (ttsonkov@gmail.com)

Бакалавър комп. науки ФМИ, Магистър изкуствен интелект ФМИ, Доктор изкуствен интелект ФМИ

20 години опит със C++

От какво ще се състои настоящия курс?

1) Наблягане на модерни ООП парадигми илюстрирани с обяснения на езика C++.

2) Множество примери с код симулиращи максимално близко реалната работа.

3) Модерни версии на C++ и сравнения с Java и Rust без да е необходимо да се знаят/учат.

4) Най-важното от курса обобщено :

- Живота на обекта и ownership – управление на ресурси чрез RAII, ясно разграничаване между owning и non-owning зависимости.

- Value semantics като default – копируемост, move семантика, ясно дефинирани инварианти.

Обобщение на курса - продължение

- Rule of Zero като основен дизайн принцип, а при нужда – коректно прилагане на Rule of Five; използване на STL типове (std::string, std::vector, smart pointers и др.) вместо ръчно управление на ресурси.

- Композиция преди наследяване – ownership не се моделира чрез inheritance; наследяването се използва единствено за полиморфизъм и substitutability.

- Различни форми на полиморфизъм – runtime (virtual), compile-time (templates, concepts), type erasure (std::function, variant).

- Обработка на грешки като част от дизайна – exceptions, std::optional, std::expected, strong exception safety.

Версии в езика C++

- C++98 / C++03

Първи стандарти. Класове, шаблони, изключения, стандартна библиотека (STL).

- C++11 – „Modern C++”

auto, range-for цикли, lambda функции, move semantics, smart pointers.

- C++14

Малки подобрения над C++11

- C++17

std::optional, std::variant, std::any

- C++20

Concepts

- C++23

std::expected

Оценяване

Текущ контрол – 100 т. + 10т. бонус от асистента
2 контролни, които се състоят от задачи (30т) + теория(снипети) (20т) – всяко от по 50 т.

Изпит – 200 т., състоящ се от:
Изпит – задачи – 100 т.
Изпит – теория (снипети) (50 т.) и събеседване(50т.) – 100 т.

Формиране на крайна оценка (макс. 300 т.)
0 – 149 точки → Слаб (2)
150 – 189 точки → Среден (3)
190 – 229 точки → Добър (4)
230 – 264 точки → Много добър (5)
265 – 300 точки → Отличен (6)

Задължителни минимуми Текущ контрол ≥ 40 т.(задачи 25т, теория 15т)
Изпит – задачи ≥ 40 т. , Изпит – теория ≥ 51 т.

Материали за курса

- learncpp.com - материал за C++ за начинаещи и напреднали
- cppreference.com - документацията на езика, подходяща за хора с повече опит
- C++ primer - Barbara E. Moo, Josée Lajoie, and Stanley B. Lippman - подходящо за начинаещи
- The C++ Programming Language pdf - Bjarne Stroustrup - създателят на езика, подходяща за начинаещи и по-опитни
- Youtube: The Cherno, Mike Shah - ако искате да видите как пишат код и как мислят по-опитни програмисти, но на достъпно за начинаещ ниво.

От какво ще се състои настоящата лекция?

Процедурен стил vs ООР - какво подобрява ООП. Основни принципи в ООП. Класове и обекти на високо ниво. Структури в езика C++. Namespaces. Enum classes. Сравнение на основните принципи с Java. Примери.

Видове памет - Stack vs Heap

• Stack: Локални променливи, бърза, автоматично управление.
Количеството заделена памет е определена по време на компилация;
Последно заделената памет се освобождава първа (First in Last out);
Заделя се в момента на дефиниция на променливите и се освобождава в момента на изход от scope-а, в която е дефинирана;

• Heap: Динамична памет (new/delete), голям обем, ръчно управление.
Заделя се и се освобождава по всяко време на изпълнение на програмата;
Програмата може да заяви блок с произволна големина
Имаме контрол над управлението на паметта

• Глобална (Статична): в тази памет се записват статичните/глобалните променливи.

• Program Code: памет, в която се пази нашият компилиран код

Пример за използване на указатели и референции, stack и heap.

Пример за указатели и референции

```cpp
#include <iostream>

int main()
{
   // 1️⃣ Stack allocation
   int a = 10;                 // 'a' lives on the stack

   // 2️⃣ Heap (dynamic) allocation
   int* heapInt = new int(30); // int lives on the heap  // pointer itself lives on the
stack// Print values
   std::cout << "Stack value (a): " << a << '\n';
   std::cout << "Heap value (*heapInt): " << *heapInt << '\n';
   // Print addresses
   std::cout << "Address of a (stack): " << &a << '\n';
   std::cout << "Address of heap value: " << heapInt << '\n';
   // 3️⃣ Clean up heap memory
   delete heapInt;             // deletes HEAP memory (not stack!)
   heapInt = nullptr;          // avoid dangling pointer
   // 4️⃣ Pointer can point to stack memory
   heapInt = &a;               // now points to stack variable 'a'
   std::cout << "Pointer now points to a, value: " << *heapInt << '\n';
   // 5️⃣ Reference example
   int& ref = a;               // ref is an alias for 'a'
   ref = 50;                   // modifies 'a'
   std::cout << "a after reference change: " << a << '\n';

   return 0;
}
```

Процедурно програмиране - пример и проблеми в кода

Характеристики

- Данните и функциите са отделени
- Логиката се разпределя на много функции
- Подходящ за малки програми

Пример

```
struct BankAccount {
    double balance;
};

void deposit(BankAccount& acc,
double amount) {
    acc.balance += amount;
}

void withdraw(BankAccount& acc,
double amount) {
    if (acc.balance >= amount)
        acc.balance -= amount;
}

int main() {
    BankAccount acc{100.0};
    deposit(acc, 50);
    withdraw(acc, 30);
}
```

Проблеми

- Няма контрол върху достъпа до balance
- Трудна поддръжка при разрастване на кода
- Всеки може да промени данните неправилно

Обектно-ориентирано програмиране

• Програмна парадигма - представлява фундаменталния стил на програмиране, който се дефинира с редица от принципи, концепции и техники за това как да имплементираме нашите програми.

• Обектно-ориентирано програмиране е програмна парадигма, при която една програмна система се моделира като набор от обекти, които взаимодействат помежду си, за разлика от традиционното виждане, в което една програма е списък от инструкции, които компютърът изпълнява. Всеки обект е способен да получава съобщения, обработва данни и праща съобщения на други обекти.

ООП пример

Основни идеи :

- Данните + поведението са заедно
- Контрол чрез енкапсулация
- По-ясен модел на реалния свят

Пример

```
class BankAccount {
    double balance;
public:
    BankAccount(double initial) : balance(initial)
{}
    void deposit(double amount) {
        balance += amount;
    }

    bool withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
            return true;
        }
        return false;
    }

    double getBalance() const {
        return balance;
    }
};

int main() {
    BankAccount acc(100.0);
    acc.deposit(50);
    acc.withdraw(30);
}
```

Предимства на ООП

- Енкапсулация – защита на данните
- Модулност – кодът е по-структуриран
- Повторна употреба (reuse)
- По-лесна поддръжка и разширяване
- По-подходящо за големи системи
- Възможност за абстракция - скриване на имплементацията и ненужните детайли от потребителя на обекта. Той работи с неговия интерфейс без да знае как точно са имплементирани неговите атрибути.

Пример

Процедурен стил	ООП
1)Данни отделно	1)Данни + логика
2)Трудна поддръжка	2)Лесно разширяване
3)По-малко сигурност	3)Контрол на достъпа
4)Често дублиране	4)Преизползване на код
5)Трудно тестване	5)По-лесно тестване
6)По-трудна поддръжка	6)По-добра четимост

Какво е struct в езика C++?

- Подобна на клас
- По подразбиране: членовете са public
- Ключовата дума е въведена заради backward compatibility с езика за програмиране C
- Елементите, наричани още членове, могат да бъдат от различен тип(int, int[], bool и т.н.) и с различна големина
- Единствената разлика с клас е, че при него видимостта на променливите е public а при класа - private

Инициализации

```
struct Box {
    double height;
    double width;
    double length;
    // double height, width, length; is also
};ssible

//Initialization:
Box b; //dangerous uninitialized.
Box b{}; // safe default - all are 0.
Box b{10.0, 5.0, 2.0}; //order matters
Box b{10.0}; //height=10.0, width=0, length=0

//modern C++ 20:
Box b{
    .height = 10.0,
    .width  = 5.0,
    .length = 2.0
};
```

Енумерации

Енумерацията е отделен тип, чиято стойност е ограничена до диапазон от стойности, който може да включва няколко изрично посочени константи(енумератори).

Стойностите на константите са стойности от интегрален тип, известен като основен тип на енумерацията.

Енумерацията има същия размер, представяне на стойност и като неговия основен тип. Освен това всяка стойност на енумерацията има същото представяне като съответната стойност на основния тип.

Пример

```
enum IceCream1 {
    vanilla, //0
    chocolate, //1
    strawberry, //2
    mango, //3
    oreo //4
};

enum IceCream2 : char {
    vanilla, //0
    chocolate, //1
    strawberry, //2
    mango, //3
    oreo //4
};

enum class Animal { dog, deer, cat, bird, human };
```

Пример за енумерации

```
enum Color { red, green, blue };                    // plain enum
enum Card { red_card, green_card, yellow_card };    // another plain enum
enum class Animal { dog, deer, cat, bird, human };  // enum class
enum class Mammal { kangaroo, deer, human };        // another enum class

void example() {
    // examples of bad use of plain enums:
    Color color = Color::red;
    Card card = Card::green_card;

    int num = color;    // no problem

    if (color == Card::red_card) // no problem (bad)
        cout << "bad" << endl;

    if (card == Color::green)   // no problem (bad)
        cout << "bad" << endl;

    // examples of good use of enum classes (safe)
    Animal a = Animal::deer;
    Mammal m = Mammal::deer;

    int num2 = a;   // error
    if (m == a)     // error (good)
        cout << "bad" << endl;

    if (a == Mammal::deer) // error (good)
        cout << "bad" << endl;
}
```

Namespaces

```
namespace A {
    int i;
}
namespace B {
    int i;
    int j;

    namespace C {
        namespace D {
            using namespace A; // all names from A injected into global namespace

            int j;
            int k;
            int a = i;         // i is B::i, because A::i is hidden by B::i
        }

        using namespace D; // names from D are injected into C
                           // names from A are injected into global namespace

        int k = 89; // OK to declare name identical to one introduced by a using
        int l = k;  // ambiguous: C::k or D::k
        int m = i;  // ok: B::i hides A::i
        int n = j;  // ok: D::j hides B::j
    }
}
/*Пространствата от имена предоставят метод за предотвратяване на конфликти с имена.
Символите, декларирани вътре в namespace block, се поставят в наименуван scope, който не
позволява да бъдат сбъркани със символи с идентични имена в други диапазони.*/
```

Създаване на динамични обекти

```cpp
#include <iostream>

struct Box {
   double height, width, length;
};

double calculcateVolume(const Box& b) {
   return b.height * b.width * b.length;
}

int main() {
   Box* boxPtr = new Box();

   // different types of assigning a value to a property when you have a
pointer
   (*boxPtr).height = 3;
   (*boxPtr).width = 1;
   boxPtr->length = 4;

   std::cout << calculcateVolume(*boxPtr) << std::endl;

   delete boxPtr; // allocated memory should always be deleted!
   return 0;
}
```

Масиви от обекти

```cpp
struct Box {
    double height, width, length;
};

int main() {
    Box arr1[30]; // 30 boxes in the stack
    Box* arr2 = new Box[20] // 20 boxes in dynamic memory and a pointer in the stack
    std::cout << arr1[0].height << " " << arr2[3].width << std::endl;

    delete[] arr2;
    return 0;
}
```

Влагане на обекти

Пример

```
struct Box {
   double height, width, length;
};

struct Warehouse {
   char name[1024];
   Box b1, b2;
};

int main() {
   Warehouse w =
   { "Ekont",
     { 1, 2, 3 }, { 4, 5, 3 }
   };

   std::cout << w.name;
   return 0;
}
```

Java Example

```
class Box {
    double height; //private by default
    double width;  //private by default
    double length; //private by default
    Box(double height, double width, double length)
{       this.height = height;
        this.width = width;
        this.length = length;
    }
}
class Warehouse {
    String name;
    Box b1;
    Box b2;

    Warehouse(String name, Box b1, Box b2) {
        this.name = name;
        this.b1 = b1;
        this.b2 = b2;
    }
}
public class Main {
    public static void main(String[] args) {
        Warehouse w = new Warehouse(
            "Ekont",
            new Box(1, 2, 3),
            new Box(4, 5, 3)
        );

        System.out.println(w.name);
    }
}
```

Базов пример за всяка лекция

```
class Student {
public:
    std::string name;
    int age;
    double grade;

    void displayInfo() const {
        std::cout << "Name: " << name
                  << ", Age: " << age
                  << ", Grade: " << grade <<
std:}endl;
};

class School {
public:
    std::string name;
    Student student1;
    Student student2;

    void displayStudents() const {
        std::cout << "School: " << name << std::endl;
        student1.displayInfo();
        student2.displayInfo();
    }
};
```

Обяснение

- Student и School са класове
- student1 и student2 са обекти в обект (School)
- Ако подадем обект по референция в метод, можем да го променяме директно.

- Вместо свободни функции и структури, методите са част от класовете.

- displayInfo() и displayStudents() демонстрират енкапсулация.

- Можем да добавим enum GradeLevel { FRESHMAN, SOPHOMORE, JUNIOR, SENIOR }; и да го включим като поле в Student

Пример

```
int main() {
    // Създаваме студенти
    Student s1;
    s1.name = "Ivan";
    s1.age = 20;
    s1.grade = 5.50;

    Student s2;
    s2.name = "Maria";
    s2.age = 19;
    s2.grade = 5.80;

    // Създаваме училище и му
    // задаваме студенти
    School school;
    school.name = "National High
School";
    school.student1 = s1;
    school.student2 = s2;

    // Показваме информация
    school.displayStudents();

    return 0;
}
```

Въпроси?

Благодаря за вниманието!

допълнителни материали:
https://learncpp.com/ - chapter 12