Въведение в ООП.

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

д-р Тодор Цонков: todort@uni-sofia.bg

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, C# 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, модули, spaceship operator

- C++23

std::expected, std::print, std::format

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.

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

Процедурно програмиране - примери

При **процедурното** програмиране: 1)данните и функциите са отделени
2)Логиката се разпределя на много функции
3)Подходящ за малки програми

Проблеми:
1) Няма контрол върху достъпа до променливите
2)Трудна поддръжка при разрастване на кода
3)Всеки може да промени данните неправилно

```
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);
}
```

Какво е обектно ориентираното 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.

Основни идеи в ООП

Основни идеи :

- Данните + поведението са заедно
- Контрол кой какво може да достъпва
- По-ясен модел на реалния свят

```
#include <iostream>
struct BankAccount {
double balance;
void deposit(double amount) {
        balance += amount;
}
void withdraw(double amount) {
if (balance >= amount)
        balance -= amount;
        }
};
int main() {
        BankAccount acc{100.0};
        acc.deposit(50);
        acc.withdraw(30);
        std::cout << acc.balance << "\n";
}
```

Предимства на ООП

Основни идеи :

- Енкапсулация – защита на данните
- Модулност – кодът е по-структуриран
- Повторна употреба (reuse)
- По-лесна поддръжка и разширяване
- По-подходящо за големи системи

При процедурния стил - данните са отделно, имаме по-трудна поддържка, по-малка сигурност, често дублиране, трудно тестване и по-малка поддръжка.

При ООП стила данните и логиката са заедно, което води до по-лесна поддръжка. Имаме контрол на достъпа, лесна разширяемост, преизползване на код, тестваемост и четимост

Какво е ключовата дума class?

В C++ class е потребителски дефиниран тип данни, който обединява:
1)данни (член-данни)
2)функции (член-функции)
3)контрол на достъпа

Това е основният механизъм за реализиране на обектно-ориентирано програмиране (ООР).

```
class ИмеНаКлас {
  // членове
};
```

Формално, класът представлява:
1)Типова декларация
2)Област на обхват (scope)
3)Логическа единица за абстракция
Класът е описание на множество от обекти със споделена структура и поведение.

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. Примери за инициализация:

1) Point p1{10, 20};
2) Point p2 = {5, 7};
3) Point p3; // x and y не са инициализирани
4) Point s{
   .x = 10,
   .y = 20
};

Енумерации в езика C++

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.

```
enum Color {
    Red,
    Green,
    Blue
};
```

Енумерации в езика C++ примери

```
enum Status {
      OK = 200,
      NotFound = 404,
      Error = 500
};
enum class ErrorCode : int {
   None = 0,
   Minor = 1,
   Major = 2
};
enum class Direction {
      Left,
      Right,
};
```

Защо enum class е по-добър?

```
enum Color { Red, Green, Blue };
Color c = Red;
int x = c;  // ✔ позволено (опасно)
```

```
enum class Color { Red };
enum class Size { Red };
Color c = Color::Red;
Size s = Size::Red;
// if (c == s) ❌ Грешка – различни типове
```

Наименовани пространства (namespaces)

В езика C++ namespace представлява механизъм за организиране на декларации и дефиниции в отделни логически области от имена.

Основната му цел е:
1)избягване на конфликти на идентификатори
2)структуриране на големи кодови бази
3)ясно дефиниране на модулни граници

В исторически план namespaces са въведени, защото в C (а и в ранния C++) всички символи съществуват в глобалното пространство от имена, което води до проблеми при големи системи и библиотеки.
Да разгледаме следната функция:
void print();
Ако две различни библиотеки дефинират функция със същото име print, ще възникне конфликт по време на линкване (нарушение на One Definition Rule).

Namespaces - примери

Namespaces решават този проблем чрез разделяне на идентификаторите в различни области:

```
namespace A {
   void print();
}
namespace B {
   void print();
}
namespace {
      int counter = 0;
      void increment() {
            counter++;
      }
}
```

В C++ namespace { } се използва за анонимни (или unnamed) пространства от имена. Те имат специална роля – всичко, което е вътре в тях, става локално за текущия преведен файл (translation unit), т.е. не се вижда в други файлове, дори ако се #include другаде.

Създаване на динамични обекти

В C++ динамични обекти са обекти, които се създават в heap паметта с new и се унищожават с delete. Това е различно от статичните обекти, които се създават на stack и автоматично се изтриват при излизане от блока.
Пример:

```
Person* p = new Person("Todor");

p->greet(); // Използваме стрелка -> за достъп до методите

// Унищожаване на обекта
delete p;
```

```
 int n = 3;
Person* people = new Person[n] {
{"Alice"}, {"Bob"}, {"Charlie"} };

for (int i = 0; i < n; ++i) {
      people[i].greet();
}

delete[] people; // важен момент: освобождаваме масив с delete[]
```

Влагане на обекти

Влагане (composition) означава: Един клас съдържа обект от друг клас като член-данна. Тоест: един обект "има" друг обект вътре в себе си.

```
class Engine {
int horsePower_;

void print() {
      std::print("Horse power: {}\n",
      horsePower_);
      }
};
```

```
class Car {
   Engine engine_;   // <-- ВЛОЖЕН ОБЕКТ
public:
   void print() const {
      engine_.print();   // използваме вложения обект
   }
};
```

Цялостен пример на наученото досега

```
#include <print>
#include <string>

struct Student {
    int age;
    double grade;

void printInfo() {
    std::print("Име: {}\n", name);
    std::print("Възраст: {}\n", age);
    std::print("Оценка: {}\n", grade);
}
};
```

```
int main() {
  Student s;       // създаваме обект
  s.age = 20;
  s.grade = 5.50;

  s.printInfo();   // извикваме метод

  return 0;
}
```

Демонстрираме създаване на обект, член данни, принтиране и извикване на метод.

Въпроси?

Благодаря за вниманието!

допълнителни материали: learncpp.com, глава 12