14 Principy objektového programování. Vlastnosti jazyka C++, datové proudy, STL kontejnery, šablony, zpracování výjimek. (Programování v C)

Principy objektového programování

Výkonný kód je v objektovém programování přidružen k datům (<u>metody jsou zapouzdřeny v objektech</u>), což umožňuje snadnější přenos kódu mezi různými projekty (abstrakce a zapouzdření).

Objekty – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).

• má atributy (stav) a metody (chování)

Třída (class) – Třída je schéma, které popisuje vnitřní strukturu objektu a jeho vnější rozhraní. Na základě třídy je možné tvořit jednotlivé objekty.

OOP is a way how to design a program to fulfill requirements and make the sources easy maintain.

OOP is a way how to design a program to fulfill requirements and make the sources easy maintain.

- Abstraction concepts (templates) are organized into classes
 - Objects are instances of the classes
- Encapsulation
 - Object has its state hidden and provides interface to communicate with other objects by sending messages (function/method calls)
- Inheritance
 - Hierarchy (of concepts) with common (general) properties that are further specialized in the derived classes
- Polymorphism
 - An object with some interface could replace another object with the same interface

A class instance (object) is created by calling a constructor to initialize values of the instance variables

Implicit/default one exists if not specified

The name of the constructor is identical to the name of the class

```
Class definition
                                                  Class implementation
class MyClass {
                                     MyClass::MyClass(int i) : _i(i)
   public:
      // constructor
                                        _ii = i * i;
      MyClass(int i);
                                        _d = 0.0;
      MyClass(int i, double d);
                                     // overloading constructor
                                     MyClass::MyClass(int i, double d) : _i
      const int _i;
                                        _ii = i * i;
      int _ii;
      double _d;
                                        _d = d;
};
  MyClass myObject(10); //create an object as an instance of MyClass
} // at the end of the block, the object is destroyed
MyClass *myObject = new MyClass(20, 2.3); //dynamic object creation
```

The value of the object is structured, i.e., it consists of particular values of the object data fields which can be of different data type

Heterogeneous data structure unlike an array

- Object is an abstraction of the memory where particular values are stored
 - Data fields are called attributes or instance variables
- Data fields have their names and can be marked as hidden or accessible in the class definition

Following the encapsulation they are usually hidden

Object:

- Instance of the class can be created as a variable declaration or by dynamic allocation using the new operator
- Access to the attributes or methods is using . or -> (for pointers to an object)

Relationship between Objects

- Objects may contain other objects
- Object aggregation / composition
- Class definition can be based on an existing class definition so, there is a relationship between classes
 - Base class (super class) and the derived class
 - The relationship is transferred to the respective objects as instances of the classes

By that, we can cast objects of the derived class to class instances of ancestor

 Objects communicate between each other using methods (interface) that is accessible to them

Access Modifiers

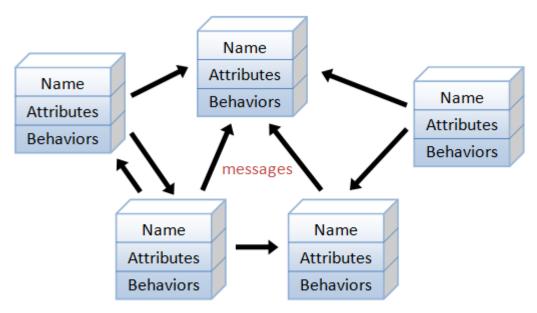
- Access modifiers allow to implement encapsulation (information hiding) by specifying which class members are private and which are public:
 - public: any class can refer to the field or call the method
 - protected: only the current class and subclasses (derived classes) of this class have access to the field or method
 - private: only the current class has the access to the field or method

Modifier	Class	Access Derived Class	"World"
public	✓.	√.	✓
protected	√,	✓	X
private	✓	×	×

Funkce objektů spočívá v tom, že vlastní procedury objektu mohou přistupovat k datovým polím sebe sama a často je upravovat (objekty mají představu o this nebo o self). V OOP jsou počítačové programy navrženy tak, že jsou vyrobeny z objektů, které na sebe vzájemně působí. Jazyky OOP jsou různorodé, ale nejoblíbenější jsou založené na třídách, což znamená, že objekty jsou instancemi tříd, které také určují jejich typy.

<u>Objekty</u> – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).

<u>Jazyky</u> - Java, C++, C#, Python, R, PHP, Visual Basic.NET, JavaScript, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Smalltalk.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Vlastnosti jazyka C++

- V současné době patří C++ mezi nejrozšířenější programovací jazyky.
- Primitivní typy: char, int, short, long, unsigned , float, double, apod.
- jazyk vyšší úrovně
- zdrojový kód mívá příponu .cpp
- hlavičkové soubory mívají příponu .h

C

- C was developed by Dennis Ritchie (1969-1973) at AT&T Bell Labs
- C is a procedural (aka) structural) programming language
- C is a subset of C++
- a sequence of procedures or steps
- C is a function driven language

C++

- Developed by Bjarne Stroustrup in 1979 with C++'s predecessor "C with Classes"
- C++ is procedural but also an object oriented programming language
- C++ can run most of C code
- The solution is achieved through C++ can model the whole solution in terms of objects and that can make the solution better organized
 - C++ is an object driven language

C

- Concept of virtual functions is not present in C
- No operator overloading
- Data can be easily accessed by other external functions
- C is a middle level language
- C programs are divided into modules and procedures
- C programs use top-down approach

C++

- C++ offers the facility of using virtual functions
- C++ allows operator overloading
- Data can be put inside objects, which provides better data security
- C++ is a high level language
- C++ programs are divided into classes and functions
- C++ programs use bottom-up approach

Does not provide namespaces

Exception handling is not easy in C

Inheritance is not possible

Function overloading is not possible

Functions are used for input/output, e.g., scanf() and printf()

Does not support reference variables

Does not support definition (overloading) operators

Namespaces are available

- Exception handling through Try and Catch block
- Inheritance is possible
- Function overloading is possible (i.e., functions with the same name)
- Objects (streams) can be use for input/output, e.g., std::cin and std::cout
- Supports reference variables, using &
- C++ supports definition (overloading) of the operators



Provides malloc() (calloc()) for dynamic memory allocation It provides free() function for memory de-allocation

friend functions

Polymorphism is not possible

C supports only built-in data

Mapping between data and functions is difficult in C

C programs are saved in files with extension .c.

- C++ provides new operator for memory allocation
- It provides delete and (delete[]) operator for memory de-allocation
- Does not support for virtual and C++ supports virtual and friend functions
 - C++ offers polymorphism
 - It supports both built-in and user-defined data types
 - In C++ data and functions are easily mapped through objects
 - C++ programs are saved in files with extension .cc, .cxx or .cpp _

Datové proudy

Datové proudy (anglicky stream; dále jen "proudy") jsou sekvence dat. Proud je definován svým vstupem a výstupem, těmi mohou být například soubor na disku, zařízení (vstupní klávesnice nebo myš, výstupní displej) nebo jiný program. Primitivní proudy pouze přenášejí binární data, specializované proudy umí s přenášenými daty manipulovat. Tyto manipulační schopnosti se liší podle typu přenášených dat.

In connection-oriented communication, a data stream is a sequence of digitally encoded coherent signals (packets of data or data packets) used to transmit or receive information that is in the process of being transmitted.[1] A data stream is a set of extracted information from a data provider.[2] It contains raw data that was gathered out of users' browser behavior from websites, where a dedicated pixel is placed. Data streams are useful for data scientists for big data and AI algorithms supply

STL kontejnery

Standard Template Library (STL) is a library of the standard C++ that provides efficient implementations of the data containers, algorithms, functions, and iterators

High efficiency of the implementation is achieved by templates with compile-type polymorphism (*Polymorfismus* – odkazovaný objekt se chová podle toho, jaké třídy je instancí)

Jedná se o standardní knihovnu šablon jazyka C++. Knihovna by měla být dodávána s každým překladačem C++. V STL můžeme najít mnoho různých a užitečných šablon. Mimo jiné zde jsou šablony datových kontejnerů, iterátory a šablony algoritmů, které pomocí iterátorů pracují s datovými kontejner.

Příklady těchto datových kontejnerů: *list, map, queue* (fronta), *stack* (zásobník). Jsou to vlastně šablony tříd.

Pod pojmem datový kontejner si představme něco, co má schopnost "zapamatovat si" nějaké data. Data jsou v kontejneru nějakým způsobem uloženy. kontejnery v STL se dělí na posloupnosti a asociativní kontejnery. Posloupnosti jsou logické posloupnosti dat, které ale nemusejí být v paměti fyzicky za sebou. Asociativní kontejnery jsou kontejnery, kde k datům přistupujeme pomocí nějakého klíče

Šablony

Jazyk C++ poskytuje možnost napsat tzv. šablonu funkce (template function, generickou funkci) – tj. kód bez specifikace konkrétního typu parametrů a výsledku (nespecifikovaný parametr je generický parametr). Konkrétní funkce (generická instance) se vytvoří automaticky při překladu (při volání se skutečnými parametry).

Templates

- Class definition may contain specific data fields of a particular type
- The data type itself does not change the behavior of the object, e.g., typically as in
 - Linked list or double linked list
 - Queue, Stack, etc.
 - data containers
- Definition of the class for specific type would be identical except the data type
- We can use templates for later specification of the particular data type, when the instance of the class is created
- Templates provides compile-time polymorphism

In constrast to the run-time polymorphism realized by virtual methods.

Example - Template Class

 The template class is defined by the template keyword with specification of the type name

```
template <typename T>
class Stack {
  public:
    bool push(T *data);
    T* pop(void);
};
```

 An object of the template class is declared with the specified particular type

```
Stack<int> intStack;
Stack<double> doubleStack;
```

Example - Template Function

 Templates can also be used for functions to specify particular type and use type safety and typed operators

```
template <typename T>
int T const & max(T const &a, T const &b)
{
    return a < b ? b : a;
}

double da, db;
int ia, ib;
std::cout << "max double: " << max(da, db) << std::endl;
std::cout << "max int: " << max(ia, ib) << std::endl;
//not allowed such a function is not defined
std::cout << "max mixed " << max(da, ib) << std::endl;</pre>
```

Výjimky

Výjimka znamená, že se stalo něco chybového, co prostě nepatří do správného chodu programu. Například, pokud ti praskne guma u auta, není extra moudré pokračovat v jízdě. Musíš buď nasadit novou gumu, nebo úplně ukončit styk s vozovkou a nechat si ho odtáhnout.

V C++ a podobných jazycích jsou vyjímky reprezentovány třídami a můžou znamenat mnohé chyby - třeba overflow i underflow proměnné, nějaké logické výjimky, chyby u datových typů, paměti a mnoho dalšího. Navíc si můžeme svoje vlastní výjimky jednoduše přidat vytvořením nové třídy. Když pak máme ve funkci nějakou podmínku a ta není splněna, použijeme **throw JménoVýjimky**. Takový to mechanismus je normálně používaný v celé standardní knihovně. No a když máme kód, u kterého očekáváme možné výjimky, použijeme **try-catch** blok. Tento kód dáme do **try** bloku. U **catch** bloku pak dáme vždy jméno výjimky do závorky a uvnitř kód pro zpracování konkrétní výjimky. Pro každou výjimku je pak jeden catch blok. V catch bloku může být teda kód co udělá něco abychom se problému zbavili a nebo prostě ukončí program a čus, to už je na nás.

Tento obrázek hezky ilustruje. Pokud se ve funkci f někde vykoná **throw JménoVýjimky**), pak se vykoná kód v příslušném catch bloku. Pokud je throw zavoláno mimo try blok, program jednoduše končí.

```
try {
    f();
} catch (const std::overflow_error& e) {
    // this executes if f() throws std::overflow_error (same type rule)
} catch (const std::runtime_error& e) {
    // this executes if f() throws std::underflow_error (base class rule)
} catch (const std::exception& e) {
    // this executes if f() throws std::logic_error (base class rule)
} catch (...) {
    // this executes if f() throws std::string or int or any other unrelated type
}
```

Další příklad s názornějším kódem.

```
MyData md;
try {
  // Code that could throw an exception
   md = GetNetworkResource();
catch (const networkIOException& e) {
   // Code that executes when an exception of type
   // networkIOException is thrown in the try block
  // Log error message in the exception object
  cerr << e.what();</pre>
catch (const myDataFormatException& e) {
  // Code that handles another exception type
   cerr << e.what();</pre>
}
// The following syntax shows a throw expression
MyData GetNetworkResource()
   if (IOSuccess == false)
     throw networkIOException("Unable to connect");
   if (readError)
      throw myDataFormatException("Format error");
}
```

Vlastní výjimky je dobré odvodit ze standardní třídy std::exception.