Výjimky

aneb když se něco nepovede

- Když se něco nepovede, program může buďto drasticky zastavit,
- nebo chybu ignorovat,
- nebo funkce místo výsledku vrátí chybový kód...
- a tím přichází o obor hodnot.
- Výjimky reprezentují možnost dát vědět, že je problém.

Výjimky

pravidla

- klíčová slova `try`, `catch` a `throw`.
- `try` uvádí blok, ve kterém mohou padat výjimky,
- `catch` uvádí blok, kde se má výjimka daného typu ošetřit,
- `throw` hází (vyvolává) výjimky syntakticky jako `return`.

Příklad

výjimky

```
int main()
{     int a=nacti(),b=nacti();
      try
      {     if(b==0) throw 0;
            printf("%d\n",a/b);
      }
      catch(int a){printf("NELZE\n");}
      return 0;
}
```

Objektová výjimka

aneb házíme objektem

```
class mavyjimka
{    public: char*cosestalo;
          mavyjimka(char*cosestalo)
     {    this->cosestalo=cosestalo;}
}
...
throw mavyjimka("prase kozu potrkalo");
... catch(const mavyjimka & a)
{    printf("%s\n",a.cosestalo);
...}
```

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.
- Vypadneme-li dříve z programu, výjimku zachytí defaultní handler výjimek (a program ukončí).

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.
- Vypadneme-li dříve z programu, výjimku zachytí defaultní handler výjimek (a program ukončí).
- `catch`-bloků může jít více za sebou, výjimku vyřídí první nalezený blok.

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.
- Vypadneme-li dříve z programu, výjimku zachytí defaultní handler výjimek (a program ukončí).
- `catch`-bloků může jít více za sebou, výjimku vyřídí první nalezený blok.
- Pozor, pokud máte handler výjimky synovské za handlerem výjimky rodičovské!

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.
- Vypadneme-li dříve z programu, výjimku zachytí defaultní handler výjimek (a program ukončí).
- `catch`-bloků může jít více za sebou, výjimku vyřídí první nalezený blok.
- Pozor, pokud máte handler výjimky synovské za handlerem výjimky rodičovské!
- `catch(...){.....}` – mělo by zachytit všechny výjimky, jenže...

Výjimky

fungování a podrobnosti

- Výjimka propadá programem, dokud se nenajde odpovídající `catch`-blok.
- Vypadneme-li dříve z programu, výjimku zachytí defaultní handler výjimek (a program ukončí).
- `catch`-bloků může jít více za sebou, výjimku vyřídí první nalezený blok.
- Pozor, pokud máte handler výjimky synovské za handlerem výjimky rodičovské!
- `catch(...){.....}` – mělo by zachytit všechny výjimky, jenže...
- některé výjimky jsou (aspoň v některých prostředích) nezachytitelné (např. `5/0`).

Namespacy a jejich využití

- Při programování se nám jména identifikátorů divně rozlézají,
- `insertovat` má například smysl do lecčeho a ne pokaždé můžeme chtít pod tímto aliasem vidět jinou funkci.
- Proto můžeme části kódu uzavřít do jmenného prostoru (`namespace`).
- Prvky namespacu definujeme poblíž sebe syntakticky podobně jako třídu:
- `namespace jmeno{ prvky...}`
- Přistupujeme buďto operátorem čtyřtečky, nebo řekneme `using namespace jmeno`
- Často se používá předdefinovaný C++ový prostor `std`.

Příklad

jmenných prostorů

```
namespace spojak
{    pridej(int co){...}//pridej do spojaku
     int uber(){...}//ze spojaku
}
namespace btree
{    pridej(int co){...}//pridej do b-stromu
     int uber(){...}//z b-stromu
}
... btree::pridej(spojak::uber());
```

Šablony

a to bude asi to poslední, co ode mě uslyšíte

- Opět chceme implementovat funkce (a třídy) pro různé datové typy,
- nechceme používat makra (protože se jich bojíme).
- Šablona nám umožňuje vytvořit parametrizovanou třídu nebo funkci.
- Za parametr dosazujeme.
- Definujeme pomocí klíčového slova `template`, do špičatých závorek popíšeme parametry, se kterými pak pracujeme.
- Následuje definice funkce nebo třídy (podle toho, jestli je to šablona na třídu nebo na funkci).

Slovo `template`

aneb co píšeme do špičatých závorek

- Do špičatých závorek popisujeme parametry šablony, jednotlivé parametry oddělujeme čárkou,

Slovo `template`

aneb co píšeme do špičatých závorek

- Do špičatých závorek popisujeme parametry šablony, jednotlivé parametry oddělujeme čárkou,
- každý parametr je ve formátu `typ název`, tedy to vypadá jako definice parametrů funkce, jenže...

Slovo `template`

aneb co píšeme do špičatých závorek

- Do špičatých závorek popisujeme parametry šablony, jednotlivé parametry oddělujeme čárkou,
- každý parametr je ve formátu `typ název`, tedy to vypadá jako definice parametrů funkce, jenže...
- typ v tomto případě může být (krom již známých) také `class` nebo `typename` (anebo další šablona).

Slovo `template`

aneb co píšeme do špičatých závorek

- Do špičatých závorek popisujeme parametry šablony, jednotlivé parametry oddělujeme čárkou,
- každý parametr je ve formátu `typ název`, tedy to vypadá jako definice parametrů funkce, jenže...
- typ v tomto případě může být (krom již známých) také `class` nebo `typename` (anebo další šablona).
- Můžeme například definovat fukci sčítající předem neznámé typy (jako u maker).

Šablony

poznámky

- šablona v C++ se vygeneruje při kompilaci jedna pro každou hodnotu parametrů, takže použijeme-li šablonu pro sčítání intů, doublů a charů, vygenerují se nám z šablony tři funkce.

Šablony

poznámky

- šablona v C++ se vygeneruje při kompilaci jedna pro každou hodnotu parametrů, takže použijeme-li šablonu pro sčítání intů, doublů a charů, vygenerují se nám z šablony tři funkce.
- Již v době kompilace se zkontroluje, zda s parametry lze dělat to, co chceme (například sčítat).

Šablony

poznámky

- šablona v C++ se vygeneruje při kompilaci jedna pro každou hodnotu parametrů, takže použijeme-li šablonu pro sčítání intů, doublů a charů, vygenerují se nám z šablony tři funkce.
- Již v době kompilace se zkontroluje, zda s parametry lze dělat to, co chceme (například sčítat).
- Povšimněte si později rozdílu oproti generikům v C#.

Příklad

šablony na funkci a šablon na třídu

```
template <typename T> T secti(T a, T b, T c)
{    return a+b+c; }
template<int I,typename T> class uloziste
{    T sklad[I];//mame pole I prvku typu T.
}
uloziste<10,int> sklad;
printf("%d\n",secti<int>(1,2,3));
secti(5,6,7);//nastoupi dedukce typu!
```

Standard template library

nenechám přednášení, dokud neukážu, k čemu jsou šablony dobré

- STL nabízí předpřipravené "kontejnery",
- kontejnery mohou být typů:
- `pair`, `vector`, `deque`, `unordered_multiset`, `list`,...
- Reprezentují to, co jsme vás učili v prvním ročníku,
- můžete je používat (se všemi důsledky, které z toho plynou).

Iterator

je něco jako axiom výběru

- umožňuje procházet prvky datové struktury.
- Formálně objekt s přetíženými operátory $++$, $+$ a pár dalšími.
- Iterátor přísluší dané dat. struktuře (např. `deque<int>::iterator it;`).

Příklad

vektoru, dvoukoncové fronty, iteratoru a metody insert

```
...
     deque<int>f1,f2(4,100),
//1. prazdna, ve druhe 4x 100.
     f3(f2.begin(),f2.end());//okopiruj
     int p[]={1,2,3,4,5};//pole
     vector<int>f4(p,p+5);//vektor z pole
     f3.insert(f3.end(),f2.begin(),f2.end());
     //prida prvky f2 na konec f3.
     for(deque<int>::iterator i=f3.begin();
               i!=f3.end();++i)
          cout<<' '<<*i;
     ...
```

Možnosti, pokud zbývá čas

ještě můžu říct něco o

- generování náhody (srand, rand),
- pravděpodobnostních a aproximačních algoritmech,
- formulovat problém diskrétní simulace,
- zkusit zodpovědět dotazy týkající se C/C++ (pravděpodobně neúspěšně).

Konec

To je konec!

- dotazy nebo nejasnosti ohledně C či C++ se mnou můžete nadále konzultovat.
- Od příště uvidíte, že C# už prakticky umíte (je v něm jen několik odlišností, syntax vypadá velmi podobně).
- Až se naučíte i C#, zjistíte, že vlastně umíte už i Javu...
- ... a že opravdu všechny jazyky z rodiny C jsou prakticky stejné.
- Toto byl jen úvod, podrobnosti u kolegy Bednárka.
- Děkuji za pozornost.