výjimky (exceptions)

- mechanizmus ošetření chyb

- jak se dá ošetřit chyba v programu?

výjimky (exceptions)

- chyby zřejmé za překladu – test, který vyřeší kompilátor
- chyby vzniklé za chodu programu – některé je schopen kompilátor (makro asssert, nedefinovaná proměnná…), dělení nulou, odmocnina záporného čísla, některé pomocné programy (VLD, checker), testování hodnoty proměnné pomocí knihovního makra - většinou končí ukončením programu, „výskokem“ dialogového okna – nelze s tím moc dělat z pozice uživatele, někdy ani programátora. Vhodné spíše pro ladění.
- představte si účetní, jak může reagovat na hlášku „sqrt(-)“
- chyby vzniklé za chodu programu ošetřené programátorem – provedení testu před rizikovými operacemi (dělení, odmocnina, test na přidělení paměti, souboru …) – následně „dopravit“ chybu do „rozumného“ místa a zde se snažit 1) zachránit data co se dá a uložit je, 2) převést (pravděpodobnou) chybu do srozumitelného jazyka – například místo „sqrt(-)“ uvést „Nedostatek dat (položky ř. …) pro vyřešení rovnice stanovující …“.

Nevýhody posledně uvedeného řešení?

výjimky (exceptions)

nevýhody řešení:
- k chybě může dojít „hluboko“ v programu (například přes více funkcí)
- pro specifikaci chyby je nutný složitější/složený typ
- je nutné řešit ukončení každé funkce (odalokovat paměť, zavřít soubory, …)
- cesta chyby do „rozumného“ místa tak může spočívat i v několika returnech:

```
struct SChyba chyba = funkce ();
if (chyba.err)
 {
   ošetři ukončení funkce
   return chyba;
 }
```

- řešení, které nabízí C++, které řeší výše uvedené – mechanizmus výjimek

výjimky (exceptions) - úvod

- oddělení řešení chyb od kódu programu
- při použití výjimky se oddělí parametry a návratové hodnoty od hlášení chyb
- při neošetření výjimky program „padne“ – nepokračuje tedy ve výpočtu s chybnými daty
- přenést řešení chyby do místa, kde je to možné a vhodné, aniž by bylo nutné se věnovat řešení mechanizmu přenosu a ošetření cesty mezi těmito body (odalokování paměti, zavření souborů …)
- možnost odlišit typ chyby a řešit chyby podle typu
- možnost popsat typ a příčinu chyby (k přenosu informace lze použít složený datový typ, který může obsahovat: typ chyby, místo chyby, hodnoty parametrů při vzniku chyby…)
-

výjimky (exceptions) – vytvoření výjimky

- výjimka je objekt, který se vytvoří (”hodí”, pomocí klíčového slova throw) v místě chyby (na základě testu chybového stavu if …).
- throw je příkaz pro vytvoření objektu přenášejícího informace o výjimce
- následně se ”legálně” ukončují funkce (včetně rušení proměnných - destruktory) až do místa kde má být chyba reprezentovaná daným typem ”chycena” (catch)
- při chybě se vytvoří objekt třídy výjimky pomocí throw V(), popřípadě s parametrem, který blíže popíše chybu

```
SChyba xxx; // složený objekt pro popis chyby

if (a ==0) // “hození” jednoduchého textového objektu char*
   throw “chyba číslo x v místě y“;
if (spatne){
    Napln(xxx, a, b, c ,d); // naplnění aktuálními daty
    throw xxx; // “hození” složitějšího objektu
  }
```

výjimky (exceptions) – definice oblasti ze které výjimky zpracováváme

- blok, ve kterém se mají výjimky odchytávat, je třeba ohraničit/označit (try-catch)
- provádí se pouze standardní odalokování – neruší tedy objekty vzniklé pomocí new (v metodě. Vzniklé v objektu a rušené v destruktoru ruší). Proto se vytvářejí objekty pracující s pamětí a nealokuje se přímo.
- výjimka se dá odchytit, pouze je-li v označeném bloku
-

```
try
   {
…
volání funkce, která hodí/vyvolá výjimku
…
} catch(X:V) {zde je řešení pro výjimku X:V}
catch (X:V2) {zde je řešení pro výjimku X:V2}
```

výjimky (exceptions) – zpracování výjimek

místa kde má být chyba reprezentovaná daným typem ”chycena” (catch)
- po odchycení je možné vybrané výjimky řešit
- catch může být pouze po bloku začínajícím try nebo za jiným catch
- výjimku vyřeší první příslušné catch, na které se narazí
- lze chytat i více výjimek
- lze chytat i postupně catch(X:V) {… catch(X:V1);}
- catch (…) odchytí všechny výjimky (je tedy uváděna jako poslední z bloků catch)
- je možné poslat výjimku dál pomocí throw;

```
try
   {
…
volání funkce, která hodí/vyvolá výjimku
…
} catch(X:V) {zde je řešení pro výjimku X:V}
catch (X:V2) {zde je řešení pro výjimku X:V2}
catch(…){zde je řešení pro (všechny) ostatní výjimky}
  throw; //nedokáži vyřešit, přepošlu dál}
```

výjimky (exceptions) – dokončení

- catch může mít parametry typu T, const T, T&, const T&, a zachycuje výjimku stejného typu, typu zděděného, pro ukazatel T, musí se dát zkonvertovat na T
- není-li výjimka zachycena, je program ukončen (terminated), pomocí funkce terminate (lze ji předefinovat pomocí set_terminate), která ukončí program

- výjimka se nadefinuje ve třídě, jíž se týká class V {… }

- u funkcí je možné napsat které výjimky funkce ”hází” a tím zpřehlednit a zjednodušit psaní a zrychlit program díky možným optimalizacím:

  void f() throw (v1,v2,v3) {} a jiné nesmí hodit (=abort)
  void f() {} nebo void f() noexcept(false) {} může hodit cokoli
  void f() throw () {} nebo void f() noexcept(true) {}nemůže hodit nic

- Při výjimce v konstruktoru se nevolá destruktor

výjimky (exceptions) – knihovní, předdefinované

- existuje společný základ pro výjimky – třída std::exception
- z této základní třídy jsou odvozeny další třídy, např. logic_error, runtime_error a dále z nich invalid_argument, out_of_range, underflow_error, …
- knihovny jazyka vyvolávají pouze výjimky z dané množiny odvozené od exception