výjimky (exceptions)

- mechanizmus ošetření chyb

- jak se dá ošetřit chyba v programu?

výjimky (exceptions)

- chyby zřejmé za překladu – test (tzv. statické testy), který vyřeší kompilátor
- chyby vzniklé za chodu programu – ošetření programátorem vložením testovacího kódu při překladu (makro asssert (ukončení programu je-li parametr false), 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 okno s textem chyby „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 řádků …) pro vyřešení rovnice stanovující …“.

Nevýhody posledně uvedeného řešení?

výjimky (exceptions)

nevýhody při řešení chyb:
- k chybě může dojít „hluboko“ v programu (například přes více funkcí; v cizí knihovně)
- 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{…};//struktura pro zaznamenání chybových stavů

struct SChyba chyba = funkce( );
if (chyba.err)
 {
   ošetři ukončení funkce
   return chyba; // „přepošli chybu“ dál
 }
```

- 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…)
- používat s rozmyslem – jsou pomalejší než standardní řešení returnem

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 – tj. ruší se proměnné a volají se destruktory objektů.
  Automaticky se neruší objekty vzniklé pomocí new uchované pomocí ukazatele. Proto se vytvářejí objekty pracující s pamětí, zapouzdřující ukazatel, které se ve svém destruktoru postarají o odalokaci paměti.
- výjimka se dá odchytit, pouze je-li v označeném bloku

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

výjimky (exceptions) – zpracování výjimek

místa kde má být chyba reprezentovaná daným typem ”odchycena” (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 odchytit i více výjimek
- lze odchytávat i postupně catch(V) {… catch(V1);}
- catch (…) odchytí všechny výjimky (je tedy uváděna jako poslední z bloků catch)
- je možné poslat výjimku dál pomocí throw;. Výjimka se totiž bere jako zpracovaná jakmile se začne zpracovávat – pokud nedojde k vyřešení, je možné/nutné ji poslat znovu

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

výjimky příklad

```
int funkce (int delka,int sirka, int volba)
{
  if (delka <= 0) throw 1;
  if (sirka <= 0) throw 2;
  if (volba != 0) throw “spatny parametr volba“;
  return delka * sirka;
}

try    {
    int vysledek = funkce(a,b,0);
}
catch(int x) // x nabývá hodnoty 1 nebo 2 (throw ve funkci)
  {/* zde je řešení pro výjimku typu int
   tj. špatná délka nebo šířka */
   printf(“%d”,x); /* v x je hodnota hozená výjimkou */  }
catch (char * txt)
  {/* zde je řešení pro výjimku řetězec */ }
catch(…)
  { /*zde je řešení pro (všechny) ostatní výjimky */
   throw; } //neznám je = nedokáži vyřešit, přepošlu dál
```

výjimky – příklad s definicí typu pro výjimku

```
enum class Err1{CH1,CH2,CH3}; // typ a hodnoty pro vyj.typu 1
enum class Err2 { CH1, CH2, CH3}; // a pro výjimku typu 2
struct E3  { unsigned TypChyby; double *data; int hodnota;}
// složený typ pro řešení s předáním více dat

int fce(int x) // funkce generující výjimku
{  // pouze příklady vygenerování výjimky (bez podmínek a kódu)
throw E1::CH1;  throw E1::CH2;  throw E2::CH1;  throw E2::CH2;
struct E3 vyj = {8,nullptr,x}; throw vyj;
// složitější objekt výjimky
}

int main()
{try {
    fce(i);
}  // řešení výjimek podle generovaného typu
catch (E1 &x) { return 1;}  // vyslaná hodnota se zapíše do x
catch (E2 &x) {                  return 2;  }
catch (E3 &x) {                  return 3;  }
return 0;}
```

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) (v C++11 je ignorováno)
  void f() {} nebo void f() noexcept(false) {} může hodit cokoli
  void f() throw () {} nebo (nově) void f() noexcept(true) {} či void f() noexcept {} nemůže hodit žádnou výjimku (optimalizátor kódu při překladu má lepší prostor k optimalizacím)

- Při výjimce v konstruktoru se nevolá destruktor třídy, v jejímž konstruktoru k výjimce došlo
- Výjimka v konstruktoru (raději) nebo destruktoru (vždy) ho nesmí opustit (ale může v něm být, pokud je odchycena)

výjimky (exceptions) – dokončení

- před hozením výjimky je dobré nastavit data do stavu "chyba" (například ukazatele odalokovat a nastavit na nullptr …), nebo nechat v chybovém stavu, pokud z něj později potřebujeme určit, co se stalo - zbytek aplikace tomu musí být přizpůsoben
- pokud nepotřebujeme bližší specifikaci chyby, lze hodit a odchytit pouze typ bez proměnné (implicitní objekt)

- předdefinovaný typ ukazatel na výjimku
  std_exception_ptr ep; // chytrý ukazatel s odkazy - objekt výjimky zmizí až s posledním
- při řešení výjimek (nejčastěji v sekci catch(...))
  std_exception_ptr ep = std::current_exception() // vrátí buď ukazatel na aktuální
  // výjimku nebo ukazatel na kopii (implementačně závislé)
- umožňuje vyřešit výjimku později mimo standardní proces výjimek
  ep = std::current_exception() // v catch si zapamatuju výjimku
  std::rethrow_exception(ep) // mimo sekci try-catch, znovu hodí uloženou výjimku
  // uložená výjimka zanikne až s koncem životnosti ep
- make_exception_ptr(ee) // vytvoří ukazatel z kopie výjimky (ee předáno hodnotou)

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 std::exception