David Matoušek

C++

BEZ PŘEDCHOZÍCH ZNALOSTÍ

Datové typy, operátory, řídicí struktury

Funkce, ukazatele, odvozené datové typy

Pole, řetězce, vstup a výstup programu

Práce se soubory, obsluha výjimek

computer press

David Matoušek

C++ bez předchozích znalostí

Computer Press
Brno
2016

C++ bez předchozích znalostí

David Matoušek

Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4640-8

Vydalo nakladatelství Computer Press v Brně roku 2016 ve společnosti Albatros Media a. s.
se sídlem Na Pankráci 30, Praha 4. Číslo publikace 23 292.

1. vydání

ALBATROS MEDIA a.s.

Obsah

Předmluva

Kniha, kterou právě držíte v ruce, je určena zájemcům, kteří se chtějí naučit programovat v jazyce C++. Provádí čtenáře od začátků programování až k pokročilým záležitostem, jako je objektově orientované programování a práce se soubory. Text je doplněn četnými příklady.

Prakticky se pracuje s vývojovým prostředím **Dev-C++**, které bylo zvoleno především s ohledem na nízké nároky při instalaci, dobrou lokalizaci do češtiny a relativně snadné ovládání. Prostředí je volně k dispozici, je poskytováno na základě GNU licence.

V úvodní kapitole jsou vysvětleny základní pojmy programování, projdeme instalaci vývojového prostředí (IDE) Dev-C++ a sestavíme první program. Nakonec se seznámíme se základním dělením datových typů.

Kapitoly 2 a 3 jsou věnovány datovým typům pro celá a reálná čísla. Jsou také vysvětleny základní vstupně/výstupní operace (načítání čísel z klávesnice a jejich výpis na obrazovku) a operátory. Pozornost je věnována i problematice tzv. konverzí (převodů hodnot různých datových typů).

Kapitoly 4 a 5 jsou věnovány větvení programu pomocí příkazů `if` a `switch` a konstrukci cyklů typu `while`, `do...while` a `for`. Použita jsou i související klíčová slova `break` a `continue`.

Kapitola 6 vysvětluje pojem pole, základní operace s poli a směřuje výklad k používání funkcí.

Kapitola 7 ukazuje používání funkcí (tedy podprogramů) za účelem přehlednější tvorby programů v jazyce C++. Je vysvětleno předávání parametrů hodnotou a návratová hodnota. Poté je vytvořena skupina funkcí pro práci s poli a jsou uvedeny základní knihovní funkce (tedy již hotové funkce). Závěr je věnován otázce globálních a lokálních proměnných.

Kapitoly 8 a 9 jsou věnovány ukazatelům. Jsou vysvětleny pojmy reference a dereference, dynamická alokace paměti (operátory `new` a `delete`), předávání parametrů přes ukazatel a předávání parametrů odkazem. Jsou také zařazeny pokročilejší záležitosti, jako jsou ukazatelová aritmetika a souvislost ukazatele s polem.

Kapitoly 10 a 11 se zaměřují na znaky a řetězce. Je vysvětlen způsob reprezentace znaků a řetězců, jsou předvedeny základní knihovní funkce pro práci s těmito typy dat společně se vstupně/výstupními operacemi.

Kapitola 12 popisuje odvozené datové typy. Kromě úvodního rozdělení datových typů jsou postupně probírány nové datové typy: výčet, struktura, sjednocení a bitové pole. Datový typ třída je popsán samostatně až v kapitole 16.

Kapitola 13 je zařazena jako shrnutí operátorů. Doplněny jsou informace k dalším operátorům, jako jsou ternární operátor, bitové operátory, operátory přiřazení, operátor čárka a na závěr je uvedena přehledová tabulka priority a asociativity všech operátorů.

Kapitola 14 představuje modulární programování, tedy tvorbu programů založenou na více zdrojových souborech (používá se pro tvorbu rozsáhlejších programů). Souběžně s tím jsou vysvětleny direktivy překladu a paměťové třídy proměnných.

Kapitola 15 vysvětluje pojmy přetížení funkcí a implicitní parametry funkcí.

Kapitola 16 je úvodem do objektově orientovaného programování (OOP). Vysvětluje jeho výhody a základní rysy na postupně budovaném příkladu třídy `TClovek`. Jsou tak postupně vysvětleny pojmy: atribut, metoda, zapouzdření, přístupové úrovně, konstruktor, destruktor. Rovněž se seznámíme s pokročilejšími termíny, jako je mělká a hluboká kopie nebo statické a konstantní členy třídy. Na závěr je pak vysvětlen pojem dědičnost.

Kapitola 17 ukazuje možnosti přetěžování operátorů (možnost definovat vlastní význam operace pro danou třídu) a použití výjimek (řeší havarijní stavy programu).

Kapitola 18 popisuje další možnosti proudové knihovny a ukazuje použití proudů pro ovládání souborů. Kromě textových souborů je ukázána i práce s binárními soubory.

Kapitola 19 je věnována třídě `string`, která zjednodušuje práci s řetězci.

Kapitola 20 se zaměřuje na „napojení" programu na zbytek operačního systému. Řeší parametry příkazové řádky programu a návratovou hodnotu programu. Vše je předvedeno na příkladu programu pro kopírování souborů. Příklad používá pokročilejší práci se soubory včetně datového bufferu.

Dále jsou zařazeny dvě přílohy. Příloha A vysvětluje číselné soustavy a reprezentaci čísel v počítači. Jedná se o vysvětlení základních a násobných jednotek informace, převody mezi číselnými soustavami, reprezentaci celých čísel bez znaménka a se znaménkem a reprezentaci čísel v plovoucí řádové čárce. Dále jsou vysvětleny základní logické operace. Příloha B je věnována stručnému popisu vývojového prostředí Dev-C++ (popis položek nabídky) a také ukazuje ladění programu na krátkém příkladu.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/K2237* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2237* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA 1

Úvod do programování v jazyce C++

V této kapitole:
- Základní pojmy
- IDE – integrované vývojové prostředí
- První program
- Pár zajímavostí
- Rozdělení základních datových typů

V této úvodní kapitole se seznámíme s úplnými základy programování, které jsou společné nejen pro programovací jazyk C++, ale pro všechny programovací jazyky. Budou vysvětleny základní pojmy (termíny), které budeme v dalším textu používat. Vzhledem k obsáhlosti celé problematiky budeme výklad pojmů „dávkovat" a s dalšími (složitějšími) termíny se budeme seznamovat postupně v následujících kapitolách.

Základní pojmy

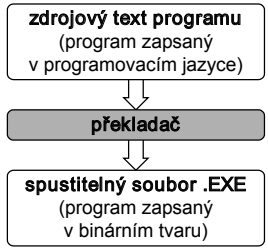
Než začneme psát jakýkoli program, musíme si provést rozbor řešeného problému. Pokud tuto fázi vynecháme („To je přeci otrava – přemýšlet, lepší je začít psát program!"), může se nám lehce stát, že tvorbou programu strávíme mnohem více času, než jsme původně předpokládali.

Následně pak dojdeme k tomu, jaké činnosti má počítač provádět a v jakém pořadí. Přesný návod, jak vyřešit daný typ úlohy, se nazývá **algoritmus**.

Pro zápis algoritmu do počítače používáme speciální **programovací jazyk**. Programovací jazyk musí být sestaven tak, aby bylo možno snadno vyjadřovat algoritmy a aby byl blízký lidskému uvažování.

Počítačový **program** vznikne tak, že algoritmus vyjádřený v programovacím jazyce převedeme do tvaru spustitelného na počítači. Přechod od algoritmu zapsaného v programovacím jazyce k programu spustitelnému na počítači se nazývá **překlad** a provádí jej speciální program označovaný jako **překladač** (kompilátor). Překladem se zdrojový text (je zapsaný v programovacím jazyce) převede do tzv. binární formy, což je vlastně sled instrukcí, které bude provádět procesor počítače.

Činnost, kterou počítač provádí při vykonávání programu, označujeme jako **proces**. Označení proces je na místě, přetváří vstupní údaje na výstupní.



Obrázek 1.1. Průběh překladu (zjednodušeno)

Pro správný zápis programu pomocí programovacího jazyka musíme dodržet syntaxi (věcnou správnost) a sémantiku (význam konstrukcí):

- **Syntaxe** popisuje, z čeho se může skládat zápis programu. Určuje například tzv. *klíčová slova*. Nemůžeme tedy použít slova, která daný programovací jazyk nezná. Syntaktická kontrola pak znamená, že se kontroluje dodržení korektnosti zdrojového textu (tedy „dodržení pravopisu").
- **Sémantika** přiřazuje jednotlivým konstrukcím jazyka význam. Je jistě možné napsat syntakticky správný program, který však nemá správný smysl.

Syntaxe + sémantika = správně fungující program

Proměnné a konstanty

Předměty, se kterými program pracuje (čte jejich hodnoty nebo je mění), nazýváme **data**. Může se jednat o čísla, znaky apod.

Většina dat obvykle během provádění programu mění svůj obsah, proto je označujeme jako **proměnné**. Data, jejichž hodnota se v průběhu provádění programu nemění, označujeme jako **konstanty**.

Typy příkazů

Zápis programu v programovacím jazyce se skládá z popisu použitých dat (tzv. deklarace) a z jednotlivých příkazů. Nejčastějšími variantami příkazů jsou:

- **deklarace proměnné** – nahlášení datového typu a názvu proměnné, kterou budeme používat (překladač pak může sledovat, jestli s proměnnými provádíme správné operace, a kontroluje, zda má dost paměti),
- **příkaz vstupu** – příkaz, který zajistí načtení dat například z klávesnice nebo jiného vstupního zařízení (například ze souboru),

- **příkaz výstupu** – příkaz, který vypíše data například na obrazovce nebo je uloží do jiného výstupního zařízení (například do souboru),
- **přiřazovací příkaz** – příkaz, který přiřadí do proměnné novou hodnotu,
- **větvení** – příkaz, který podle určité podmínky rozdělí další postup do více cest,
- **volání podprogramu** – příkaz, který provede dříve vytvořenou část programu; takto můžeme například provést výpočet funkce sin(x), pokud je tento podprogram v daném jazyce k dispozici; rovněž můžeme vytvářet vlastní podprogramy (funkce).

IDE – integrované vývojové prostředí

V souvislosti s programováním se zkratka IDE objevuje s anglickým souslovím Integrated Development Environment. Tedy doslovně přeloženo jako „integrované vývojové prostředí".

IDE znamená pro programátora komfort, kdy v rámci jediného programu (přesněji aplikace) zapíše zdrojový text, nastaví (pokud je to nutné) parametry překladu, provede překlad, testuje a ladí činnost hotového programu.

Existuje řada překladačů jazyka C++, které však vyžadují další znalosti uživatele, což je pro začátečníky nevýhodné. Totiž, pokud nemáme k dispozici IDE, musíme napsat zdrojový text například v poznámkovém bloku a poté spustit překlad pomocí příkazové řádky obvykle s použitím speciálního dávkového souboru. Možnosti ladění jsou v takovém případě velmi omezené.

IDE Dev-C++

Vývojových prostředí existuje celá řada. K nejznámějším patří **Microsoft Visual Studio** od Microsoftu nebo **C++ Builder** od firmy Embarcadero (původně Borland).

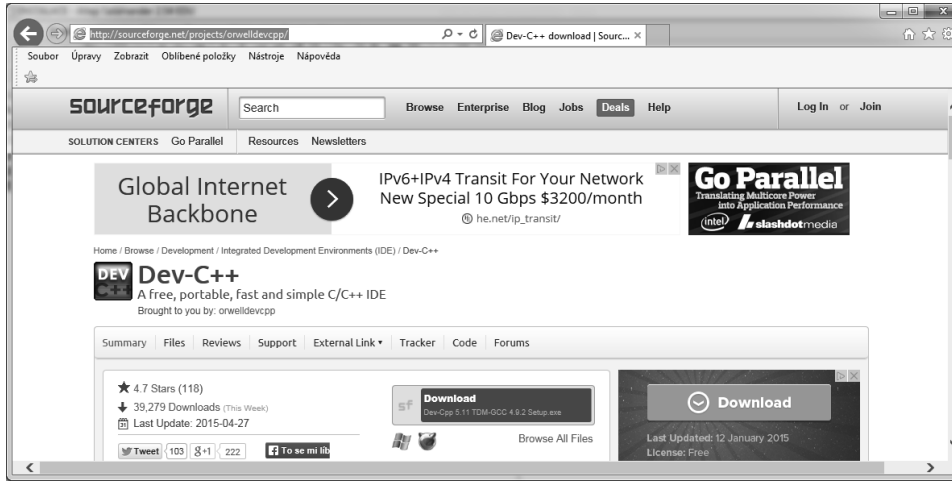
My jsme pro tuto knihu vybrali vývojové prostředí **Dev-C++**, které je vyvíjeno na základě licence GNU pod hlavičkou Bloodshed Software. Důvody byly: dobrá znalost tohoto prostředí, časově neomezená licence pro nekomerční použití, nízké nároky při instalaci a částečná lokalizace prostředí do češtiny (většina příkazů hlavní nabídky je v češtině, ale například chyby překladu jsou hlášeny v angličtině).

Vývojové prostředí Dev-C++ lze stáhnout ze stránek *http://www.bloodshed.net/dev/devcpp.html*.

Instalační soubor verze 5.9.2 z února 2015 měl velikost okolo 45 MB (tedy mnohem méně než ostatní výše uváděné produkty).

Stažení a instalace Dev-C++

Odkaz pro stažení instalace Dev-C++ je možné najít na výše uvedených stránkách (*http://sourceforge.net/projects/orwelldevcpp/*) nebo lze použít přímý odkaz (jak také ukazuje obrázek 1.2).



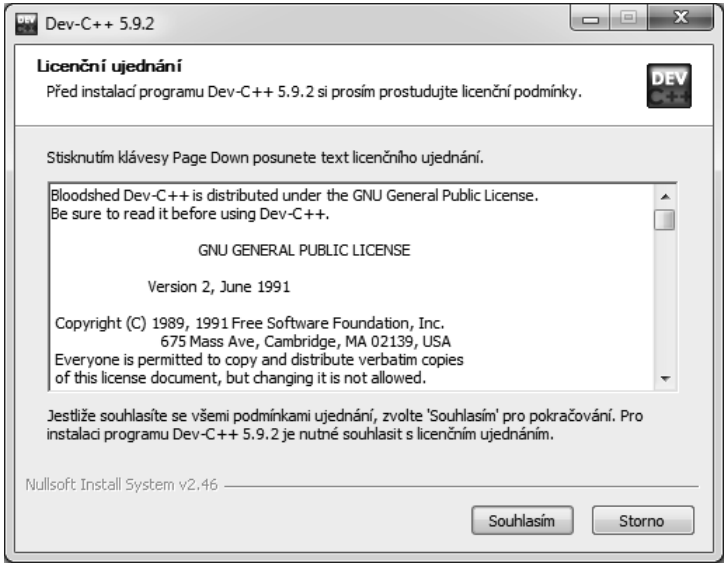
Obrázek 1.2. Stažení instalačního programu ze stránek SourceForge

Instalační soubor stáhneme pomocí tlačítka **Download**.

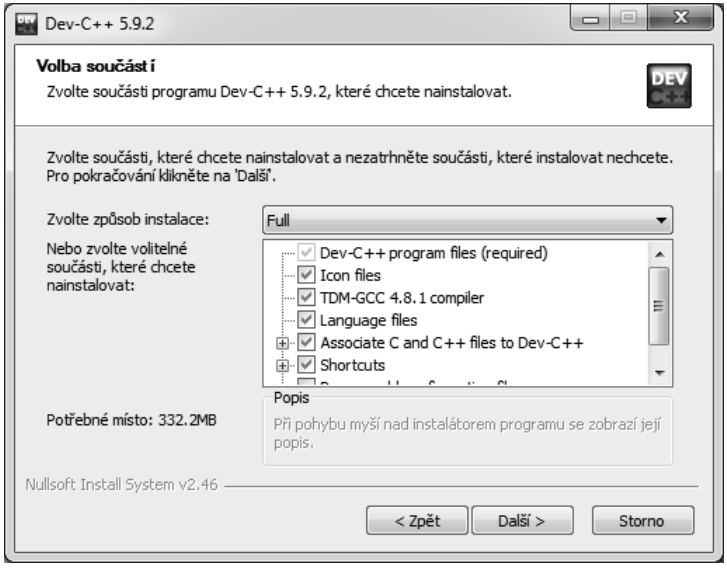
Spustíme instalaci pomocí souboru s názvem (název může být upraven při stažení novější verze): **Dev-Cpp 5.9.2 TDM-GCC 4.8.1 Setup.exe**. Nejdříve je provedena dekomprese a následně se zobrazí dialog pro volbu jazyka instalace dle obrázku 1.3. Vybereme **Cesky** a potvrdíme stiskem tlačítka **OK**.

Obrázek 1.3. Volba jazyka instalace

Následuje dialog s licenčním ujednáním dle obrázku 1.4, který je nutné potvrdit stiskem tlačítka **Souhlasím**.



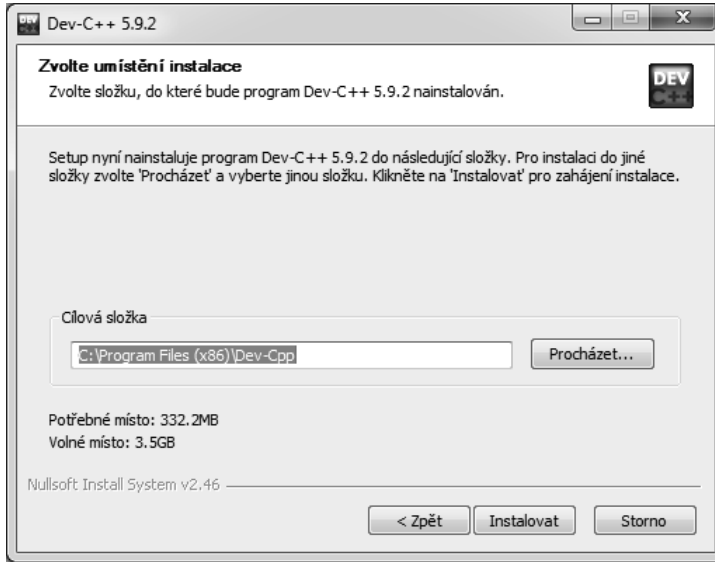
Obrázek 1.4. Dialog s licenčním ujednáním



Obrázek 1.5. Volba součástí pro instalaci

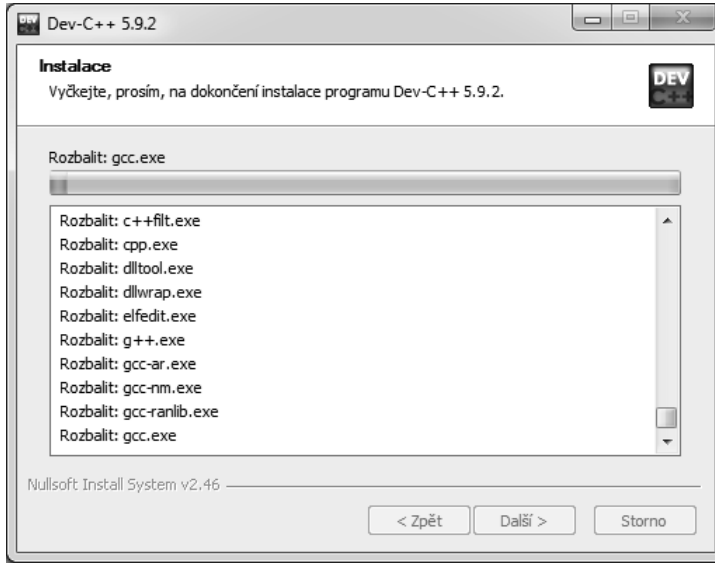
Následuje dotaz na výběr součástí pro instalaci dle obrázku 1.5. Zde kromě jiného dochází k asociaci s koncovkami souborů, které se používají v jazyce C++. Doporučujeme ponechat výchozí nastavení (vše vybrané) a poté pokračovat stiskem tlačítka **Další**. V této verzi je pro instalaci třeba asi 330 MB diskového prostoru.

Nakonec je třeba vybrat adresář, do kterého bude instalace provedena, viz obrázek 1.6. Doporučujeme ponechat výchozí volbu a pokračovat stiskem tlačítka **Instalovat**.



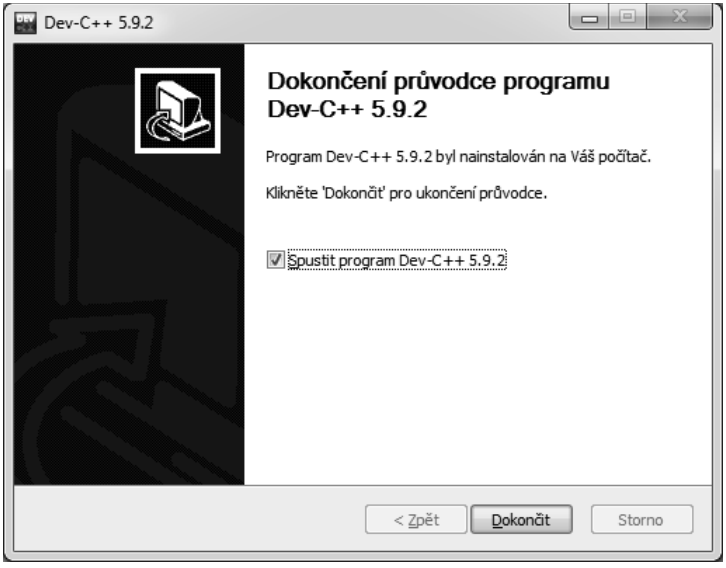
Obrázek 1.6. Volba adresáře pro provedení instalace

Poté se již rozbíhá instalace, jak dokumentuje obrázek 1.7.



Obrázek 1.7. Průběh vlastní instalace

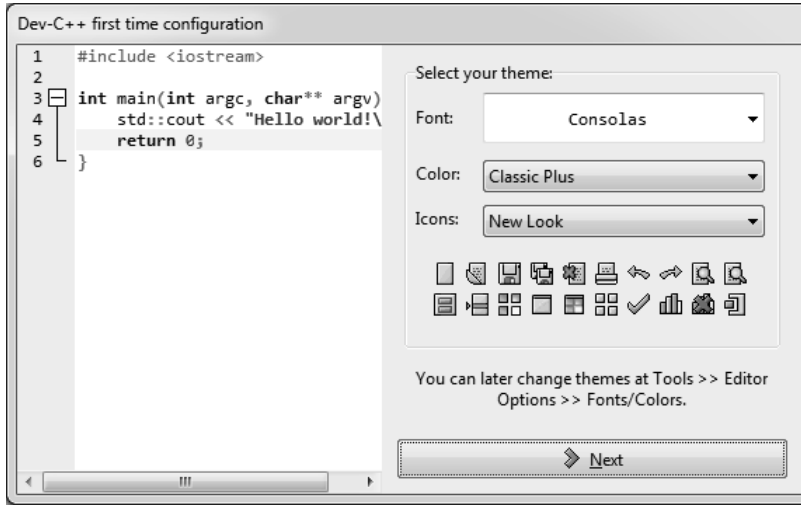
Na závěr se zobrazí dialog dle obrázku 1.8, který umožňuje přímé spuštění vývojového prostředí. Volbu **Spustit program Dev-C++** necháme aktivní a stiskneme tlačítko **Dokončit**.



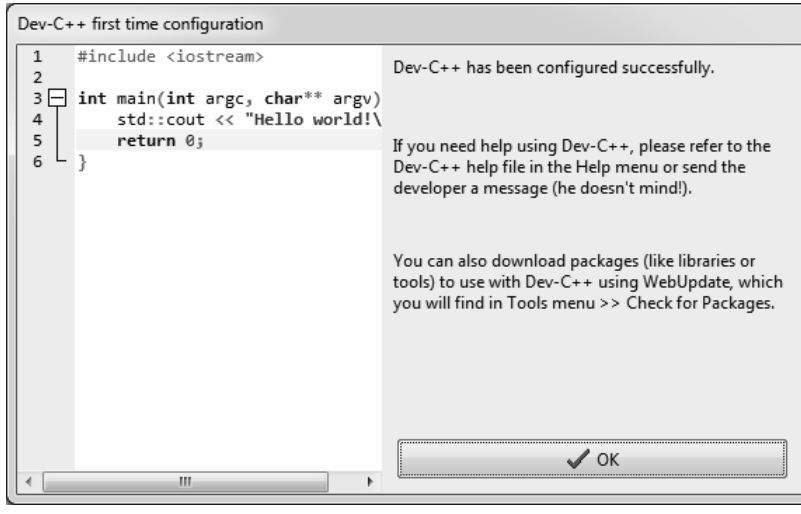
Obrázek 1.8. Dokončení instalace

Následně se objeví tři velmi podobné dialogy dle obrázků 1.9 až 1.11, které slouží pro konfiguraci prostředí. Jedná se o možnost změny jazyka instalace, schématu a nástrojů. Poslední dialog potvrzuje dokončení konfigurace.

Obrázek 1.9. Volba jazyka



Obrázek 1.10. Volba schématu



Obrázek 1.11. Dokončení konfigurace

Následně již nabíhá prostředí.

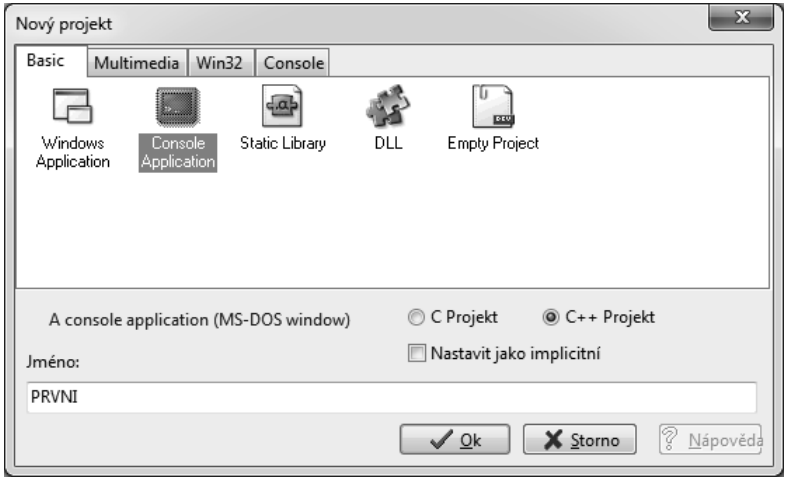
První program

Nyní tedy můžeme vytvořit svůj první program v jazyce C++.

Spustíme vývojové prostředí Dev-C++ (pokud již není spuštěno) a pomocí příkazu nabídky **Soubor → Nový → Projekt** (viz obrázek 1.12) vyvoláme dialog volby typu projektu dle obrázku 1.13.

Obrázek 1.12. Založení nového projektu

V dialogu volby typu projektu dle obrázku 1.13 se přepneme na kartu **Basic** a jako typ projektu zvolíme **Console Application**. Konzolová aplikace je typ aplikace pro operační systém Windows, která pracuje v textovém režimu (jako MS-DOS okno). Dále vybereme projekt typu **C++ Projekt**. V textovém poli **Jméno** zadáme název projektu **PRVNI**. Pokračujeme stiskem tlačítka **Ok**.

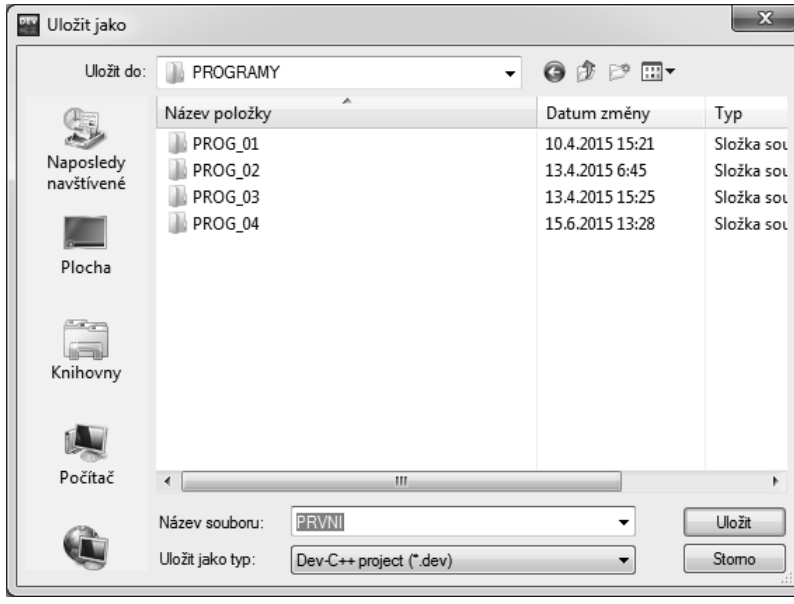


Obrázek 1.13. Volba typu a názvu projektu

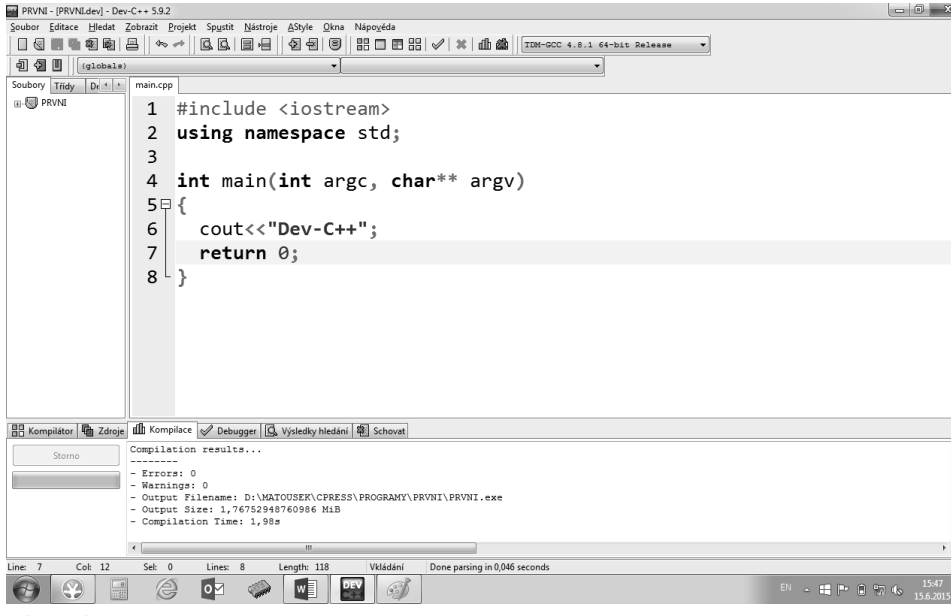
Následně se zobrazí dialog dle obrázku 1.14, což je klasický dialog volby adresáře pro uložení projektu. Zvolíme příslušnou složku a pokračujeme tlačítkem **Uložit**.

Zápis našeho prvního programu dokumentuje obrázek 1.15. Soubor byl standardně pojmenován jako **MAIN.CPP** (main = hlavní program, CPP značí zkratku pro „cé plus plus“). Všimněte si, že textový editor rozlišuje syntaxi pomocí barev a stylů písma. Příkazy preprocesoru (povíme si o nich podrobněji v průběhu našeho seznamování s jazykem C++) se značí zeleně,

tučně jsou označena klíčová slova jazyka C++, červeně speciální znaky, modře textové řetězce a fialově zápisy čísel.



Obrázek 1.14. Volba adresáře pro uložení souborů projektu



Obrázek 1.15. Zápis programu v textovém editoru

Výpis programu je ještě uveden v textové formě níže.

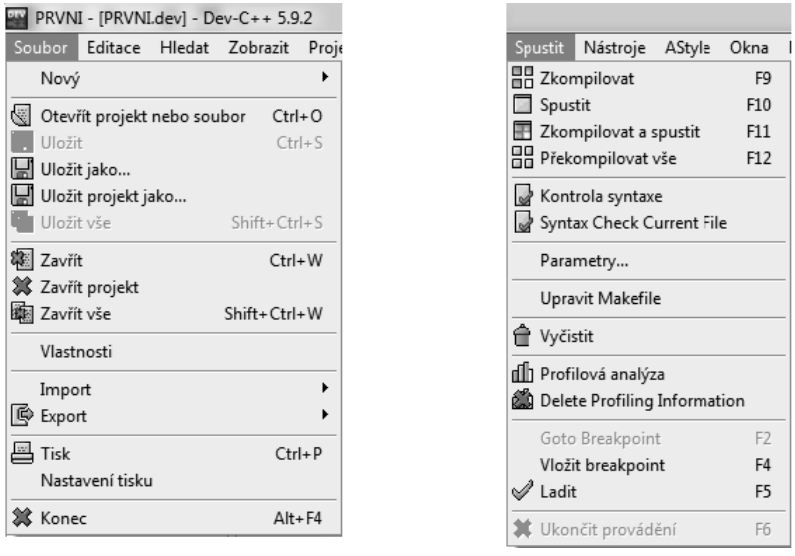
MAIN.CPP

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
  cout<<"Dev-C++";
  return 0;
}
```

Klíčové položky nabídky

Na začátku naší práce s vývojovým prostředím Dev-C++ vystačíme pouze s několika málo příkazy z nabídek **Soubor** a **Spustit**, pro informaci jsou tyto dvě nabídky uvedeny formou obrázku 1.16.

Nabídka **Soubor** obsahuje známé příkazy. Nejčastěji budeme používat příkaz **Nový** buď pro založení nového projektu, nebo pro vložení nového zdrojového souboru do stávajícího projektu. Příkaz **Zavřít projekt** zavře aktuální projekt, ale neukončí vývojové prostředí. Příkaz **Zavřít** zavře aktuální zdrojový soubor otevřený v editoru. Více nyní vědět nemusíme. Příkazem **Otevřít projekt nebo soubor** otevřeme dříve vytvořený projekt nebo zdrojový soubor (který jsme třeba nedopatřením zavřeli).



Obrázek 1.16. Nabídky Soubor a Spustit

Nabídka **Spustit** obsahuje čtyři klíčové příkazy, které jsou svými názvy velmi podobné, proto vysvětlíme rozdíly (viz obrázek 1.16):

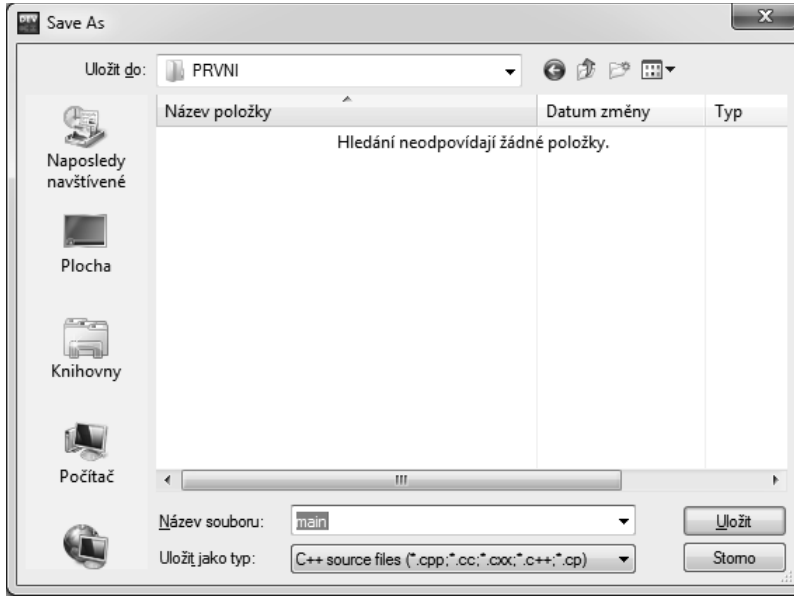
- **Zkompilovat F9** – přeloží aktuální projekt, vznikne cílový spustitelný soubor s koncovkou EXE,
- **Spustit F10** – spustí dříve vytvořený cílový soubor (nový překlad se neprovádí),
- **Zkompilovat a spustit F11** – pokud došlo ve zdrojovém souboru ke změnám, provede nový překlad; pokud je překlad v pořádku, spustí následně program,
- **Překompilovat vše F12** – provede opětovný překlad všech zdrojových souborů v projektu.

Překlad programu

Nyní již víme, že nejsnazší způsob pro překlad a následné spuštění programu je použít zkratkovou klávesu **F11** (vyvolá se příkaz **Zkompilovat a spustit**).

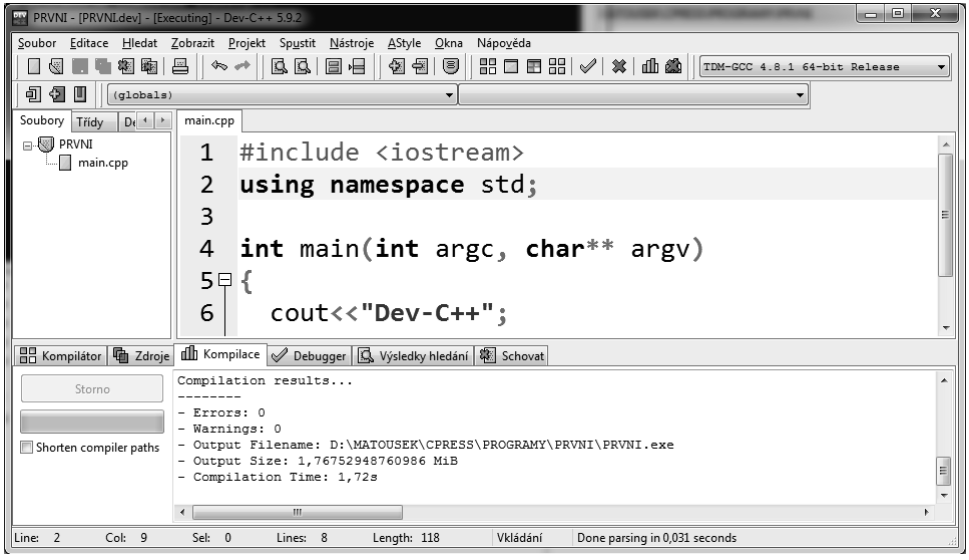
Před prvním překladem budeme dotázáni na název zdrojového souboru proto, aby vývojové prostředí mohlo pod tímto názvem soubor uložit. Výchozí název **MAIN.CPP** lze změnit, obvykle však tento název ponecháme. Viz obrázek 1.17.

Při opětovných překladech již vývojové prostředí ukládá zdrojové soubory automaticky. Situaci po úspěšném překladu zachycuje obrázek 1.18, následně je přeložený program spuštěn v konzoli, jak dokladuje obrázek 1.19.



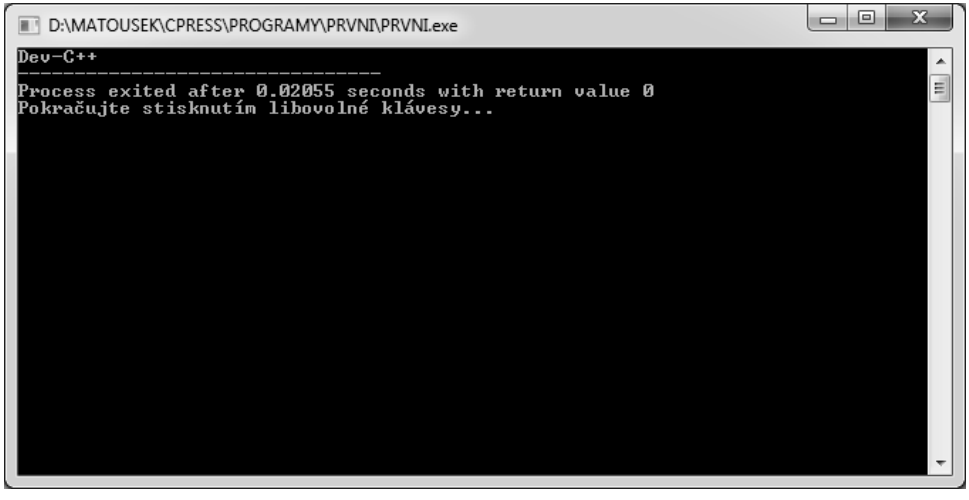
Obrázek 1.17. Dialog pro volbu názvu zdrojového souboru pro jeho uložení

Je možné, že se vám nepodaří zdrojový text opsat správně. Hlášení chyb je sice v angličtině, ale chybu lze rozeznat pomocí čísla řádku. Dále víme, že například klíčová slova jsou psána tučně (uděláme-li chybu při zápisu klíčového slova, nebude příslušné slovo vyobrazeno tučně).



Obrázek 1.18. Situace po překladu, v dolní části jsou informace o výsledcích překladu

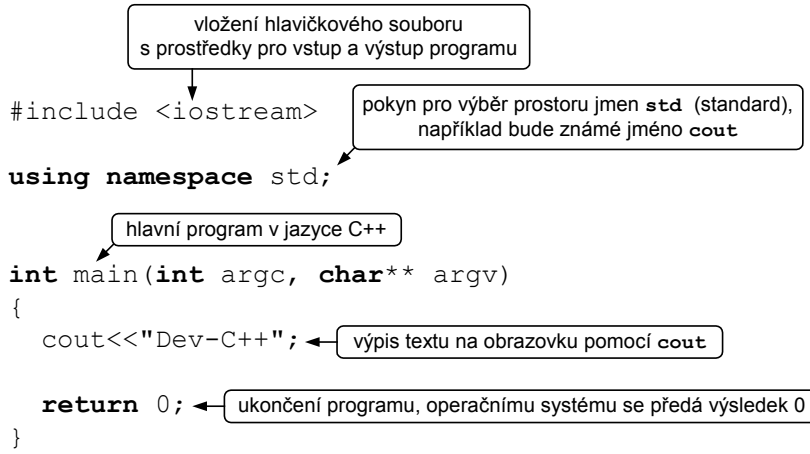
Výsledkem běhu programu je výpis textu Dev-C++. Prostředí samo dále vypíše hlášení ohledně času běhu programu (zhruba 20 ms) a návratové hodnotě 0. Rovněž vloží čekání na stisk klávesy, aby bylo možné si vypsané výsledky prohlédnout. Viz obrázek 1.19.



Obrázek 1.19. Běh programu v konzoli

Stručné vysvětlení zápisu programu

V této chvíli není možné zcela vysvětlit konstrukce, které jsou v zápisu tohoto jednoduchého programu použity. Pokusme se o to alespoň stručně pomocí obrázku 1.20.



Obrázek 1.20. Rozbor programu

Na začátku zdrojového textu se obvykle uvádí tzv. direktivy neboli příkazy preprocesoru. Tyto příkazy rozeznáme podle toho, že začínají symbolem # (mřížka). Příkaz `#include <iostream>` slouží pro vložení zdrojového souboru s názvem iostream (input/output stream, tedy vstupně/výstupní proudy; proudy jsou v jazyce C++ prostředky pro vstup a výstup programu; například načtení dat z klávesnice nebo výpis na obrazovku, již v kapitole 2 se dozvíme, jak se proudy používají).

Příkaz `using namespace std;` vybere k použití prostředky poskytované tzv. jmenným prostorem `std` (standard). Do tohoto prostoru patří zejména prostředky `cin` (vstupní konzolový proud, používá se pro načtení dat z klávesnice) a `cout` (výstupní konzolový proud, používá se pro výpis dat na obrazovku).

Jazyk C++ používá (podobně jako většina jiných programovacích jazyků) blokovou strukturu. Program lze rozdělit do určitých celků, tzv. podprogramů, které se v případě jazyka C++ nazývají funkce. Hlavní program je pak zapsán jako funkce `main` (česky hlavní). Každá funkce může mít návratovou hodnotu a parametry (o tom později v kapitole 7).

V hlavním programu je zapsán příkaz výstupu, který do proudu `cout` odešle uvedený textový řetězec. V programu to způsobí vypsání příslušného řetězce na obrazovce.

Program je zakončen příkazem `return 0;` (návrat), který ukončí jeho běh a operačnímu systému předá návratový kód. Zde zvolená hodnota 0 se většinou používá jako znamení, že program doběhl normálně (bez hlášení chyby). Různými hodnotami návratových kódů můžeme operačnímu systému sdělovat různé chyby, které při své činnosti program zjistil.

Pár zajímavostí

Na tomto místě připojíme pár zajímavostí, které se mohou hodit při běžném psaní programů.

Komentáře neboli poznámky

Pokud potřebujeme určitou část zdrojového textu označit poznámkou (neboli napsat k ní komentář), lze použít dvě varianty:

1. Jedná-li se o krátký komentář, který se vejde na jeden řádek, použijeme dvě lomítka a za ně zapíšeme komentář. Například:

```
cout<<"Dev-C++"; //vypíše Dev-C++
```

2. Jedná-li se o delší komentář na více řádků, začneme kombinací lomítko – hvězdička, poté zapíšeme příslušný text a komentář ukončíme opačnou kombinací, tedy hvězdička – lomítko. Například:

```
/*toto je komentář,
který zabírá více řádků*/
```

Komentáře slouží pouze pro orientaci ve zdrojovém textu. Všechny komentáře překladač ignoruje.

Proto také komentáře často používáme k tomu, abychom určitou část zdrojového textu „vypustili“. Říkáme, že tuto část textu „zakomentujeme“. Překladač pak tuto část nepřeloží (tedy „vynechá“).

Pomocné nástroje na Internetu

Na internetu lze nalézt mnoho užitečných informací k programování. Doporučujeme zejména stránky:

- **www.cplusplus.com** – zde je k dispozici řada informací k jazyku C++, například podrobnější výklad klíčových slov a knihoven, nápověda a tutoriály. Text je v angličtině.
- **cpp.sh** – zajímavé stránky obsahující tzv. C++ Shell. Jedná se o online překladač jazyka C++. Můžete ho používat zejména pro testování krátkých programů, a přitom nepotřebujete mít v počítači nainstalovaný překladač jazyka C++.

Rozdělení základních datových typů

V následujících kapitolách se postupně seznámíme s datovými typy pro uložení různých hodnot. To nám dovolí vytvářet programy, které budou například schopny provádět jednoduché matematické operace.

Datový typ určuje obor hodnot (v jakém rozsahu proměnná uchovává hodnoty), interpretaci obsahu (číslo, znak apod.) a množinu přípustných operací (například čísla lze sčítat nebo porovnávat). Mezi základní datové typy patří:

- *celočíselné (v pevné řádové čárce):* například **int** (viz kapitolu 2),
- *reálné (v pohyblivé řádové čárce):* například **float** (viz kapitolu 3),
- *logické:* **bool** (viz kapitolu 4),
- *znakové (pro uchování jednoho znaku):* **char** (viz kapitolu 10),
- *odvozené datové typy (jsou odvozeny z číselných nebo znakových typů):* například pole (viz kapitolu 6), ukazatel (viz kapitoly 8 a 9), struktura (viz kapitolu 12), třída (viz kapitolu 16) atd.

KAPITOLA 2

Celočíselné datové typy

V této kapitole:
- Celá čísla se znaménkem a bez znaménka
- Základní vstupně/výstupní operace
- Pokročilejší operace s proměnnými a proudy
- Aritmetické operace s celými čísly
- Zadávání číselných literálů v různých soustavách

Paměť počítače ani jeho výpočetní výkon nejsou neomezené, proto používáme různé datové typy přiměřené pro konkrétní použití. Základní charakteristiky datových typů:

- **Obor hodnot** určuje omezení rozsahu hodnot uchovávaných v proměnné příslušného datového typu. Má-li být rozsah hodnot větší, musí proměnná zabírat větší prostor v paměti a také provádění operací zabírá obvykle více času.
- **Interpretace obsahu** určuje, jakým způsobem se nahlíží na obsah proměnné. Na hardwarové úrovni je obsah proměnné reprezentován „shlukem jedniček a nul". Obsah proměnné můžeme ale chápat jako číslo nebo znak či mnoha jinými způsoby.
- **Množina operací** určuje, které operace lze vykonávat nad proměnnými daného datového typu, případně jak tyto operace probíhají. Například dělení dvou celých čísel je prováděno celočíselně. Kdežto dělení dvou reálných čísel je prováděno v reálné aritmetice. Sčítat můžeme proměnné číselných typů, kdežto pole sčítat nelze. Atd.

Nejjednodušší skupinu datových typů představují celá čísla, proto touto skupinou začneme. Níže se dozvíme, že celočíselné proměnné mohou být bez znaménka (kladné hodnoty včetně nuly) nebo se znaménkem. Dále se seznámíme se základními vstupně/výstupními operacemi (naučíme se načíst hodnotu čísla z klávesnice do proměnné a vypsat obsah proměnné na obrazovku). Poté se seznámíme s aritmetickými operacemi poskytovanými jazykem C++.

Celá čísla se znaménkem a bez znaménka

Základní rozdělení celých čísel je na *čísla bez znaménka* (kladné hodnoty včetně nuly) a *čísla se znaménkem* (kladné i záporné hodnoty včetně nuly). Pro rozlišení těchto dvou skupin používáme klíčová slova `unsigned` (označuje proměnnou bez znaménka) a `signed` (označuje proměnnou se znaménkem).

Pro deklaraci proměnné typu celé číslo používáme datový typ `int`. Je to zkratka anglického označení *integer*, která odpovídá českému pojmu celé číslo. Proměnná tohoto typu zabírá v paměti počet bajtů, který je obvykle určen typem překladače. *V současnosti je obvyklé, že překladače berou základní velikost celého čísla v rozsahu 32 bitů, tedy jako 4bajtové číslo.*

Je-li třeba, lze původní rozsah typu `int` upravit buď zkrácením na polovinu rozsahu v bajtech, takový typ označujeme jako `short` (česky „krátký"). Případně je možné původní rozsah v bajtech rozšířit na dvojnásobek, takový typ označujeme jako `long` (česky „dlouhý"). Dokonce existuje ještě „širší" typ označovaný jako `long long`.

Celá čísla bez znaménka

Začněme nejdříve typem `unsigned int`, který odpovídá celému číslu bez znaménka v rozsahu 4 bajtů. Do proměnné tohoto typu lze uložit hodnoty 0 až $2^{32} - 1$. Počet odlišných čísel kódovaných ve dvojkové soustavě ve 32 bitech odpovídá 2^{32}. Vzhledem k tomu, že začínáme od nuly, je maximální hodnota vždy o jedničku nižší než 2^{32}, tedy maximální hodnota je $2^{32} - 1$.

Datový typ `unsigned short int` má rozsah v bajtech poloviční proti předchozímu případu (tedy 2 bajty). Do proměnné tohoto typu lze uložit hodnoty 0 až $2^{16} - 1$.

Datový typ `unsigned long int` má obvykle rozsah shodný jako typ `unsigned int`.

Datový typu `unsigned long long int` má proti typu `unsigned int` rozsah v bajtech dvojnásobný (tedy 8 bajtů). Do proměnné tohoto typu lze uložit hodnoty 0 až $2^{64} - 1$.

Zvláštností je pak datový typ `unsigned char`. Později se dozvíme (viz kapitolu 10), že se používá pro znakové proměnné. Nicméně ho lze použít i pro celá čísla a rozsah odpovídá jednomu bajtu, číselně 0 až $2^8 - 1$.

Rychlý přehled o celočíselných datových typech bez znaménka získáme z tabulky 2.1. Najdeme zde příslušný datový typ, jeho zkrácený název (nemusíme tak například zapisovat `unsigned int`, stačí zapsat pouze `unsigned`), velikost proměnné v bajtech (tedy informaci o místu, které zabere v paměti), číselný rozsah a příponu. Přípona se používá v případech, kdy je nutné zapisovat číselné hodnoty ve vyšším rozsahu, než odpovídá (znaménkovému) datovému typu `int`. Příponu uvádíme také v případě, že je nutné zdůraznit, jaký má být typ číselné konstanty.

Tabulka 2.1. Datové typy pro celá čísla bez znaménka

Datový typ	Zkrácený název	Velikost	Číselný rozsah	Přípona
`unsigned int`	`unsigned`	4 bajty	0 až 2^{32}-1	`UL`
`unsigned long int`	`unsigned long`		(0 až 4 294 967 295)	
`unsigned short int`	`unsigned short`	2 bajty	0 až 2^{16}-1	není
			(0 až 65 535)	

Datový typ	Zkrácený název	Velikost	Číselný rozsah	Přípona
`unsigned long long int`	`unsigned long long`	8 bajtů	0 až 2^{64}-1 (0 až 18 446 744 073 709 551 615)	`ULL`
`unsigned char`	není	1 bajt	0 až 2^{8}-1 (0 až 255)	není

Celá čísla se znaménkem

Celá čísla se znaménkem používají jeden bit pro uložení znaménka, takže rozsah čísel je pak o jeden bit užší. Záporných hodnot je stejný počet jako kladných hodnot včetně nuly.

Pro 4bajtové typy `signed int` a `signed long int` je rozsah -2^{31} až 2^{31} - 1. Pro 2bajtový typ `signed short int` je rozsah -2^{15} až 2^{15} - 1. Pro 8bajtový typ `signed long long int` je rozsah -2^{63} až 2^{63} - 1. Pro 1bajtový typ `signed char` je rozsah -2^{7} až 2^{7} - 1.

Tabulka 2.2. Datové typy pro celá čísla se znaménkem

Datový typ	Zkrácený název	Velikost	Číselný rozsah	Přípona
`signed int` `signed long int`	`int` `long`	4 bajty	-2^{31} až 2^{31}-1 (-2 147 483 648 až +2 147 483 647)	není `L`
`signed short int`	`short`	2 bajty	-2^{15} až 2^{15}-1 (-32 768 až +32 767)	není
`signed long long int`	`long long`	8 bajtů	-2^{63} až 2^{63}-1 (-9 223 372 036 854 775 808 až +9 223 372 036 854 775 807)	`LL`
`signed char`	`char`	1 bajt	-2^{7} až 2^{7}-1 (-128 až +127)	není

Charakteristiky celočíselných datových typů

Chceme-li snadno zjistit rozsahy jednotlivých číselných typů v dané implementaci překladače, stačí použít soubor **LIMITS.H**. Tento soubor nalezneme v adresáři instalace vývojového prostředí.

V tabulce 2.3 jsou uvedeny nejpoužívanější symboly a jejich význam (v závorce je pak uvedena hodnota, která platí pro případ překladačů, které jako základní typ uvažují 32bitové celé číslo). Podobně jsou definovány symboly `LONG_MIN`, `LONG_MAX`, `ULONG_MAX`, `LLONG_MIN`, `LLONG_MAX`, `ULLONG_MAX`.

Tabulka 2.3. Vybrané symboly ze souboru LIMITS.H

Symbol	Význam
`CHAR_BIT`	počet bitů pro reprezentaci proměnné typu `char` (8)
`SCHAR_MIN`	minimální hodnota proměnné typu `signed char` (-128)
`SCHAR_MAX`	maximální hodnota proměnné typu `signed char` (+127)
`UCHAR_MAX`	maximální hodnota proměnné typu `unsigned char` (255)
`SHRT_MIN`	minimální hodnota proměnné typu `signed short` (-32 768)
`SHRT_MAX`	maximální hodnota proměnné typu `signed short` (+32 767)
`USHRT_MAX`	maximální hodnota proměnné typu `unsigned short` (65 536)
`INT_MIN`	minimální hodnota proměnné typu `signed int` (-2 147 483 648)
`INT_MAX`	maximální hodnota proměnné typu `signed int` (+2 147 483 647)
`UINT_MAX`	maximální hodnota proměnné typu `unsigned int` (4 294 967 295)

Základní vstupně/výstupní operace

Vstupně/výstupní operace, tedy načítání hodnot proměnných a jejich výpis, se v jazyce C++ provádí pomocí tzv. proudů (anglicky streams). Proudy poskytují velmi přehledný a výkonný prostředek pro vstupně/výstupní operace.

Standardní (tzv. konzolové) proudy se nazývají `cin` (vstupní) a `cout` (výstupní). Pro jejich použití je třeba zapsat pokyn (použití symbolů z tzv. prostoru jmen s označením `std`, tím se nám zjednoduší zápisy: místo plné formy `std::cin` a `std::cout` budeme zapisovat pouze `cin` a `cout`):

```
using namespace std;
```

Základní výstupní operace

Výstupní proud se nazývá `cout`, což je zkratka od anglického označení *console output* (výstupní konzole). Jako výstupní konzole pracuje většinou obrazovka počítače. Hodnoty odeslané do výstupního proudu `cout` uvidíme tedy vypsané na obrazovce.

Pro odeslání hodnoty do výstupního proudu používáme operátor <<, který se odborně označuje jako *insertor* (insert – vložit, vkládá hodnoty do proudu). Symbol << představuje myšlenou šipku, která říká, že hodnota je nasměrována do proudu. Operátor << se používá v této formě zápisu:

```
výstupní_proud << výraz;
```

Je-li třeba odeslat do proudu (provést výpis) více výrazů, lze operátor << použít v daném příkazu opakovaně, například:

```
výstupní_proud << výraz1 << výraz2 << výraz3;
```

Chceme-li vypsat text, musíme jej uvést v rámci uvozovek:

```
výstupní_proud << "text";
```

Pro ukončení řádku při výpisu používáme *manipulátor* `endl` (zkratka od anglického označení *end line*):

```
výstupní_proud << endl;
```

Po tomto krátkém výkladu můžeme již předvést výstupní operace pomocí proudu `cout` na krátkém příkladu. Vypíšeme na obrazovku základní vlastnosti výše popsaných datových typů `int` a `unsigned`. Pro získání informace o velikosti proměnné nebo datového typu používáme operátor `sizeof` (vychází z anglického označení „*size of* “, píše se však bez mezery). Tento operátor se používá ve formě zápisu:

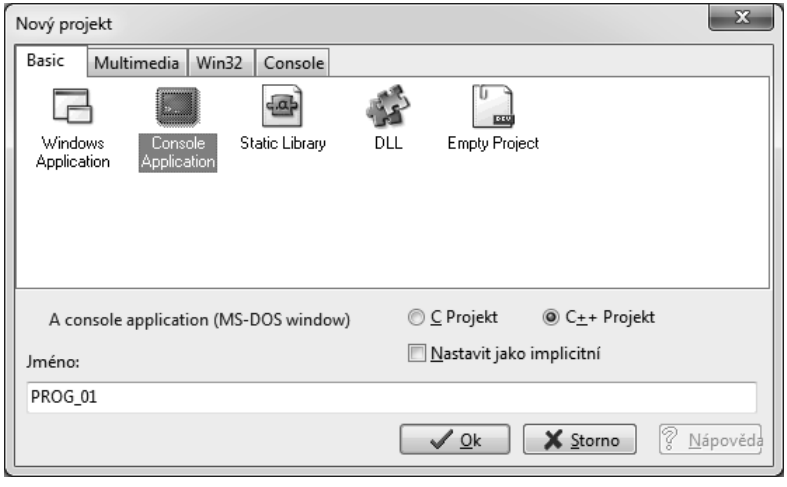
```
sizeof(datový_typ|proměnná)
```

Výsledkem použití operátoru `sizeof` je celé číslo, které určuje počet bajtů potřebných pro uložení proměnné příslušného datového typu do paměti.

Nyní spustíme vývojové prostředí Dev-C++. Pomocí položky nabídky **Soubor → Nový → Projekt**, viz obrázek 2.1, vyvoláme dialog volby typu projektu dle obrázku 2.2.

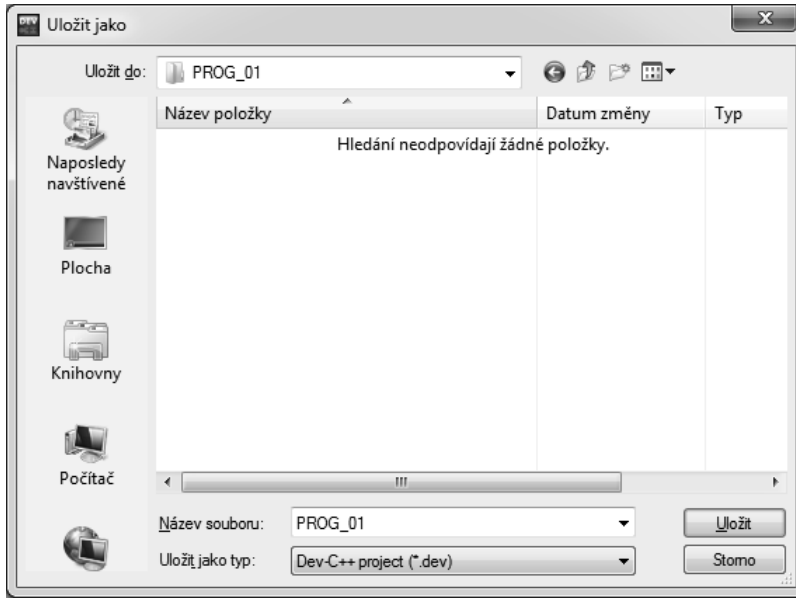
Obrázek 2.1. Založení nového projektu

V dialogu volby typu projektu dle obrázku 2.2 se přepneme na kartu **Basic** a jako typ projektu zvolíme **Console Application**. Konzolová aplikace je typ aplikace pro operační systém Windows, která pracuje v textovém režimu (jako MS-DOS okno). Dále vybereme projekt typu **C++ Projekt**. V textovém poli **Jméno** zadáme název projektu **PROG_01**. Pokračujeme stiskem tlačítka **Ok**.

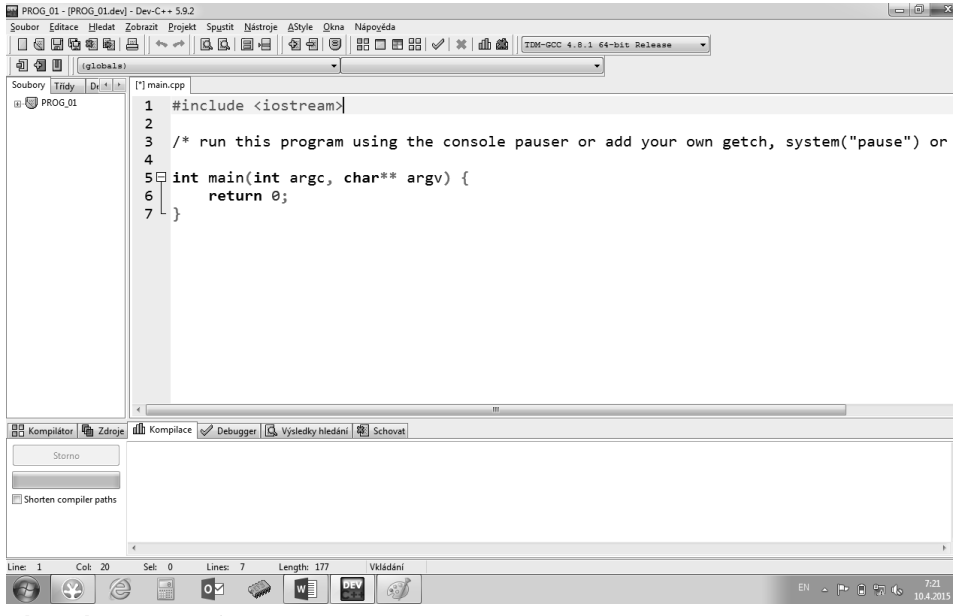


Obrázek 2.2. Volba typu projektu

Následně se zobrazí dialog dle obrázku 2.3, což je klasický dialog volby adresáře pro uložení projektu. Zvolíme příslušnou složku a pokračujeme tlačítkem **Uložit**.



Obrázek 2.3. Volba adresáře pro uložení souborů projektu



Obrázek 2.4. Nyní můžeme začít psát program

Po těchto krocích je již vývojové prostředí připraveno na vlastní programování. Situace je zřejmá z obrázku 2.4.

Vývojové prostředí samo vloží hlavičkový soubor **iostream**, který je potřebný pro použití proudů. Pro vložení externího zdrojového souboru se používá direktiva `#include` (*include* – zahrnout) v této formě:

```
#include <název_souboru>
```

Následně požadujeme použití jmenného prostoru `std` a sami požadujeme vložení hlavičkového souboru **limits.h**, který obsahuje informace o celočíselných datových typech.

Hlavní program zapisujeme do funkce `main`.

Program pomocí proudu `cout` vypíše informace o velikosti datového typu `int` v bajtech a minimální a maximální hodnotu proměnné typu `int`.

Následně jsou podobně vypsány informace o datovém typu `unsigned`.

PROG_01:

```
#include <iostream>
using namespace std;
#include <limits.h>
int main(int argc, char** argv)
{
  cout<<"velikost typu int: "<<sizeof(int)<<endl;
  cout<<"minimalni hodnota: "<<INT_MIN<<endl;
  cout<<"maximalni hodnota: "<<INT_MAX<<endl;
  cout<<endl;
  cout<<"velikost typu unsigned: "<<sizeof(unsigned)<<endl;
  cout<<"minimalni hodnota: "<<0<<endl;
  cout<<"maximalni hodnota: "<<UINT_MAX<<endl;
  return 0;
}
```

Výpis programu v konzoli:

```
velikost typu int: 4
minimalni hodnota: -2147483648
maximalni hodnota: 2147483647
velikost typu unsigned: 4
minimalni hodnota: 0
maximalni hodnota: 4294967295
```

Základní vstupní operace

Vstupní proud se nazývá `cin`, což je zkratka anglického označení *console input* (vstupní konzole). Jako vstupní konzole pracuje většinou klávesnice počítače. Hodnoty pro načtení do proměnných zadáváme tedy pomocí klávesnice.

Pro načtení hodnoty ze vstupního proudu používáme operátor >>, který se odborně označuje jako *extraktor* (extract – vyjmout, vyjímá hodnoty z proudu). Symbol >> představuje myšlenou šipku, která říká, že hodnota je nasměrována do proměnné. Operátor >> se používá v této formě zápisu:

```
vstupní_proud >> proměnná;
```

Příklad použití vstupní operace (načtení čísla z klávesnice) bude proveden níže, po vysvětlení dalších detailů.

Pokročilejší operace s proměnnými a proudy

Před dalšími příklady se musíme seznámit s dalšími operacemi, které je potřebné znát při práci s proměnnými a vstupně/výstupními proudy.

Deklarace proměnné

Deklarace proměnné představuje pro překladač nahlášení názvu (přesněji *identifikátoru*) proměnné a jejího datového typu.

Překladač musí zejména kontrolovat, zda proměnná daného názvu již dříve neexistuje (název proměnné se rovněž nesmí shodovat se dříve zavedenými symboly nebo klíčovými slovy programovacího jazyka). Při zpracování programu lze používat pouze proměnné, které byly předem nahlášeny (deklarovány). Pokud je ve výrazu použit symbol, který překladač nezná, hlásí chybu.

Překladač dále kontroluje, zda všechny nahlášené proměnné může uložit do paměti, nebo zda byla překročena její velikost. V případě překročení velikosti dostupné paměti hlásí chybu.

Deklarace proměnné začíná uvedením datového typu a následuje název proměnné:

```
<datový_typ> <identifikátor>;
```

Pro volbu identifikátoru platí tato pravidla:

- je doporučeno používat písmena anglosaské abecedy (nelze používat diakritiku), případně číslice (identifikátor však nesmí začínat číslicí),
- je možné používat speciální znaky (např. _), některé znaky jsou však vyhrazené pro operátory jazyka (např. +, -, *, /, # atd.),
- identifikátor nesmí obsahovat mezeru ani jiné tzv. bílé znaky,
- překladač rozlišuje výšku písmen (case sensitive).

Deklarace proměnné může být spojena s definicí její výchozí hodnoty, která se uvede za rovnítkem:

```
<datový_typ> <identifikátor>=<výchozí hodnota>;
```

V rámci jednoho řádku lze deklarovat více proměnných daného datového typu s tím, že některé mohou mít definovánu výchozí hodnotu a jiné ne. Příklad:

```
int pocet=10,x,y,konec=0;
```

Výše jsme deklarovali čtyři proměnné typu celé číslo se znaménkem. Proměnné `pocet` a `konec` mají definovány výchozí hodnoty. Proměnné `x` a `y` mají nedefinovanou („náhodnou“) výchozí hodnotu. Obvykle se deklarace proměnné bez definice výchozí hodnoty nedoporučuje, programovací jazyk C++ ji však umožňuje.

Výstupní manipulátory dec, hex, oct

Výstupní proudy podporují přepnutí výpisu čísel do desítkové, šestnáctkové (hexadecimální) nebo osmičkové (oktalové) soustavy. Pro tyto případy se používají manipulátory `dec`, `hex` a `oct`.

Připomeňme, že standardní způsob výpisu je prováděn v desítkové soustavě, takže manipulátor `dec` používáme obvykle jen v případě, že chceme výpis přepnout ze šestnáctkové nebo osmičkové soustavy zpět do desítkové soustavy.

Níže je uveden jednoduchý příklad. Do proměnné `x` (je typu `int`) načteme z klávesnice celé číslo. Jeho hodnotu pak zobrazíme desítkově, šestnáctkově a osmičkově.

PROG_02:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
  int x;  //deklarace proměnné
  cout<<"Zadej cele cislo: ";
  cin>>x; //načtení hodnoty z klávesnice
  cout<<"desitkove: "<<dec<<x<<endl;     //desítkový výpis
  cout<<"sestnactkove: "<<hex<<x<<endl;  //šestnáctkový výpis
  cout<<"osmickove: "<<oct<<x<<endl;     //osmičkový výpis
  return 0;
}
```

Výpis programu v konzoli (zadali jsme 31):

```
Zadej cele cislo: 31
desitkove: 31
sestnactkove: 1f
osmickove: 37
```

Aritmetické operace s celými čísly

Aritmetické operátory provádí základní matematické operace s čísly, jejich zápis je velmi podobný jako v matematických výrazech.

Základní aritmetické operátory

Základní aritmetické operátory předepisují výpočet operací součtu, rozdílu, součinu, podílu (celočíselného) a zbytku po celočíselném dělení. Tyto operátory zapisujeme podobně jako v matematice, symbol operátoru je zapsán mezi dva operandy. Operátory, které mají dva operandy, označujeme odborně jako *binární*.

Tabulka 2.4. Základní aritmetické operátory

Operátor	Příklad zápisu	Význam
+	x+y	součet dvou čísel uložených v proměnných x a y
–	x–y	rozdíl dvou čísel uložených v proměnných x a y
*	x*y	součin dvou čísel uložených v proměnných x a y
/	x/y	celočíselný podíl dvou čísel uložených v proměnných x a y
%	x%y	zbytek po dělení (modulo) dvou čísel uložených v proměnných x a y

Níže uvedený příklad demonstruje výpočty s operátory dle tabulky 2.4 na dvou proměnných a = 11 a b = 5. Hodnoty obou proměnných jsou nejdříve kontrolně vypsány a následně se zobrazují výsledky získané použitím operátorů nad těmito operandy.

PROG_03:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
  int a=11,b=5;
  cout<<"a="<<a<<", b="<<b<<endl<<endl;
  cout<<"a+b="<<a+b<<endl;
  cout<<"a-b="<<a-b<<endl;
  cout<<"a*b="<<a*b<<endl;
  cout<<"a/b="<<a/b<<endl;
  cout<<"a%b="<<a%b<<endl;
  return 0;
}
```

Výpis programu v konzoli:

```
a=11, b=5
a+b=16
a-b=6
a*b=55
a/b=2
a%b=1
```

Unární aritmetické operátory

Zvláštní postavení mají tzv. *unární operátory*. Tyto operátory pracují pouze s jedním operandem, viz tabulku 2.5. Jelikož jsou některé symboly operátorů podobné jako v předchozí tabulce 2.4, musíme někdy rozlišovat například binární – (ve významu rozdílu dvou čísel) a unární – (ve významu záporného znaménka čísla).

Operátory + a – označují znaménko čísla a zapisují se vždy před operand. Používání unárního + není příliš časté, unární – slouží pro změnu znaménka čísla (odpovídá násobení –1).

Tabulka 2.5. Unární aritmetické operátory

Operátor	Příklad zápisu	Význam
+	+x	kladné znaménko čísla
–	–y	záporné znaménko čísla (násobení –1)
++	x++ *nebo* ++x	x++ postinkrement, ++x preinkrement
––	x–– *nebo* ––x	x–– postdekrement, ––x predekrement

Operátory ++ a –– se označují jako *inkrement* (zvýšení o 1) a *dekrement* (snížení o 1). Existují dvě formy zápisu, před proměnnou a za proměnnou (viz také tabulku 2.5):

- Je-li operátor zapsán před proměnnou, má význam *preinkrementu* (++) resp. *predekrementu* (––). Předpona *pre* značí „před". Hodnota proměnné je tedy nejdříve zvýšena resp. snížena a jako taková vstupuje do výrazu.
- Je-li operátor zapsán za proměnnou, má význam *postinkrementu* (++) resp. *postekrementu* (––). Předpona *post* značí „po". V rámci výrazu se uvažuje původní hodnota proměnné, ke zvýšení resp. snížení dochází až po určení hodnoty výrazu.

V níže uvedeném programu jsou tyto operátory předvedeny na několika výpočtech:

1. Nejdříve je na proměnné x = 11 a y = -5 použit operátor unární –, takže dostáváme výsledek -11 a 5.
2. Následně je sestaven složitější výraz vysl=++x * y--. Vzhledem k zápisu si musíme uvědomit, že ++x značí preinkrement (již v rámci vyhodnocování výrazu se uvažuje „nová" hodnota x = 12) a y–– značí postdekrement (v rámci vyhodnocování výrazu se bere „sta-

rá“ hodnota y = -5). Výsledek je tedy -60, vedlejším efektem je změna obou proměnných: x = 12, y = -6.

3. Pro lepší přehled jsou hodnoty obou proměnných nastaveny zpět: x = 11, y = -5 a je vyhodnocován výraz vysl=x++ * --y. Zde x++ značí postinkrement (uvažuje se x = 11) a −−y značí predekrement (uvažuje se y = -6). Výsledek je tedy -66, vedlejším efektem je změna obou proměnných: x = 12, y = -6.
4. Vidíme, že po vyhodnocení těchto dvou komplikovaných výrazů jsou hodnoty proměnných x a y stejné, ale výsledky výrazů se liší.

PROG_04:

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
  int x=11,y=-5,vysl;
  cout<<"x="<<x<<", y="<<y<<endl;
  cout<<"-x="<<-x<<", -y="<<-y<<endl;
  cout<<endl;
  vysl=++x * y--;
  cout<<"vysl=++x * y--: "<<vysl<<", x="<<x<<", y="<<y<<endl;
  x=11;
  y=-5;
  cout<<endl;
  vysl=x++ * --y;
  cout<<"vysl=x++ * --y: "<<vysl<<", x="<<x<<", y="<<y<<endl;
  return 0;
}
```

Výpis programu v konzoli:

```
x=11, y=-5
-x=-11, -y=5
vysl=++x * y--: -60, x=12, y=-6
vysl=x++ * --y: -66, x=12, y=-6
```

Priorita a asociativita

Priorita určuje pořadí zpracování operandů při použití různých operátorů. Změny priority docílíme pomocí závorek `( )`.

V tabulce 2.6 jsou uvedeny dosud probrané aritmetické operátory. Nejvyšší prioritu tedy mají unární operace. Operátory uvedené ve stejném řádku mají stejnou prioritu. Pokud se v daném výrazu použijí operátory se stejnou prioritou (a nejsou použity závorky), budou automaticky seskupeny v pořadí určeném **asociativitou**.

Tabulka 2.6. Priorita a asociativita aritmetických operátorů

Priorita	Operátory	Asociativita
1	++, −−, *unární* +, −	←
2	*, /, %	→
3	*binární* +, −	→

Uvažme, jak se vyhodnotí výraz: `3*5/3*8`. Operátory `*` a `/` mají stejnou prioritu, o výsledku tedy rozhodne asociativita. Automatické seskupení proběhne tedy takto: `((3*5)/3)*8`. Nejdříve se tedy stanoví součin `3*5`, tedy 15. Následuje dělení `15/3`, tedy 5. Nakonec proběhne násobení `5*8`, celkový výsledek pak bude 40.

Pokud použijeme závorky, může být výraz vyhodnocen jinak a vede k jinému výsledku, například: `(3*5)/(3*8)`. Nyní bude výsledek 0 (celočíselné dělení `15/24`).

Zadávání číselných literálů v různých soustavách

Přímé uvedení číselné hodnoty se označuje jako *literál*. Jazyk C++ umožňuje zapisovat literály v desítkové, šestnáctkové a osmičkové soustavě.

Desítková soustava je výchozí (platné číslice jsou 0 až 9).

Pro zápis číselného literálu v šestnáctkové soustavě musíme použít „předponu“ `0x` nebo `0X` (platné číslice jsou 0 až 9 a dále A až F, případně místo velkých písmen použijeme malá písmena a až f). Například literál 0x10f představuje šestnáctkovou hodnotu 10F (tedy desítkovou hodnotu 271).

Zápis číselného literálu v osmičkové soustavě začíná nulou (0), proto pozor! Například literály 123 a 0123 představují odlišné hodnoty. První zápis je desítkový literál 123, druhý je osmičkový literál 0123 (desítkově 83).

Je-li třeba určit datový typ literálu, používají se přípony: `U` (unsigned), `L` (long), `LL` (long long) a `ULL` (unsigned long long). Pokud není přípona určena, rozumí se automaticky, že literál je zapsán dle typu `int`. Viz tabulky 2.1 a 2.2.

KAPITOLA 3

Datové typy pro reálná čísla

V této kapitole:
- Vlastnosti datových typů pro reálná čísla
- Vstupně/výstupní operace z pohledu reálných čísel
- Aritmetické operace s reálnými čísly
- Implicitní a explicitní typové konverze
- Priorita a asociativita dosud probraných operátorů

V kapitole 2 jsme se seznámili s celočíselnými datovými typy. Pro uložení číselných hodnot ve velkém rozsahu obvykle nepožadujeme tak velkou přesnost, jakou nám poskytují celá čísla. Obvykle vystačíme s přesností například 6 desítkových číslic.

Pro reprezentaci reálných čísel pak používáme datové typy obecně označované jako *čísla v plovoucí řádové čárce* (anglicky *float-point*). Takové číslo je reprezentováno dvěma složkami: mantisou a exponentem. *Mantisa* (M) je hodnota čísla, *exponent* (E) určuje řád. Viz tento předpis (pro desítkovou soustavu):

reálné číslo = $M \cdot 10^E$

Příklady zápisů čísel v tomto formátu: $1{,}602 \cdot 10^{-19}$ (elementární náboj); $6{,}023 \cdot 10^{23}$ (Avogadrova konstanta).

Vlastnosti datových typů pro reálná čísla

K dispozici jsou celkem tři typy: `float`, `double` a `long double`, které se liší oborem hodnot. Tyto typy se také liší velikostí v bajtech a přesností, která je určena počtem platných cifer (číslic).

Formát zápisu reálných literálů v desetinném tvaru se vyznačuje používáním tečky místo obvyklé desetinné čárky, například: `1.2345`. Reálné literály lze také zapisovat v semilogaritmickém tvaru, například: `1.602E-19`.

Tabulka 3.1. Vlastnosti datových typů pro reálná čísla

Typ	Přibližný obor hodnot
`float` velikost: 4 bajty, přesnost: 6 platných desítkových cifer	$\pm 1{,}2 \cdot 10^{-38}$ až $\pm 3{,}4 \cdot 10^{+38}$
`double` velikost: 8 bajtů, přesnost: 15 platných desítkových cifer	$\pm 2{,}2 \cdot 10^{-308}$ až $\pm 1{,}8 \cdot 10^{+308}$
`long double` velikost: 16 bajtů, přesnost: 18 platných desítkových cifer	$\pm 3{,}4 \cdot 10^{-4932}$ až $\pm 1{,}2 \cdot 10^{+4932}$

Přesné charakteristiky těchto datových typů lze získat pomocí symbolů z hlavičkového souboru **FLOAT.H**.

Níže je doplněn krátký program, který pro každý z uvedených typů vypíše postupně: velikost v bajtech, počet platných desítkových číslic, minimální a maximální hodnotu. Můžete tyto údaje porovnat s tabulkou 3.1.

PROG_01:

```
#include <iostream>
#include <float.h>
using namespace std;
int main(int argc, char** argv)
{
  cout<<"sizeof(float)="<<sizeof(float)<<endl;
  cout<<"FLT_DIG="<<FLT_DIG<<endl;
  cout<<"FLT_MIN="<<FLT_MIN<<endl;
  cout<<"FLT_MAX="<<FLT_MAX<<endl;
  cout<<endl;
  cout<<"sizeof(double)="<<sizeof(double)<<endl;
  cout<<"DBL_DIG="<<DBL_DIG<<endl;
  cout<<"DBL_MIN="<<DBL_MIN<<endl;
  cout<<"DBL_MAX="<<DBL_MAX<<endl;
  cout<<endl;
  cout<<"sizeof(long double)="<<sizeof(long double)<<endl;
  cout<<"LDBL_DIG="<<LDBL_DIG<<endl;
  cout<<"LDBL_MIN="<<LDBL_MIN<<endl;
  cout<<"LDBL_MAX="<<LDBL_MAX<<endl;
  cout<<endl;
  return 0;
}
```

Výpis programu v konzoli:

```
sizeof(float)=4
FLT_DIG=6
FLT_MIN=1.17549e-038
FLT_MAX=3.40282e+038
sizeof(double)=8
DBL_DIG=15
DBL_MIN=2.22507e-308
DBL_MAX=1.79769e+308
sizeof(long double)=16
LDBL_DIG=18
LDBL_MIN=3.3621e-4932
LDBL_MAX=1.18973e+4932
```

Vstupně/výstupní operace z pohledu reálných čísel

Informace ke vstupně/výstupním operacím z kapitoly 2 můžeme nyní doplnit o přehled dalších manipulátorů a rozšířit informacemi, které platí pro reálná čísla. Při výpisu hodnoty můžeme nastavit parametry, které určí, jak přesně výpis proběhne:

- **šířka** – určuje minimální počet znaků, které se mají vypsat. Uvažujme například výpis čísla 56 na 5 míst. V tomto případě je třeba přidat další 3 znaky, aby celková šířka výpisu byla 5 znaků. Znak, který se použije jako výplň, je obvykle mezera (je možné jej ale změnit pomocí parametru *výplňový znak*). Pokud uvažujeme výpis čísla 56 v šířce 0, vypíše se prostě `56` (výpis hodnoty nesmí být zkreslen).
- **přesnost** – určuje počet číslic za desetinnou tečkou, které se zobrazí. Například výpis čísla π (3,14159265 …) při šířce 5 a přesnosti 3 vypadá takto: `3.142` (při výpisu probíhá nezbytné zaokrouhlení); platí pro formáty `fixed` a `scientific`.
- **výplňový znak** – určuje znak, který se použije jako výplň při výpisu hodnoty ve větší šířce, jako výchozí výplňový znak je použita mezera.
- **zarovnávání** – určuje způsob zarovnání výpisu hodnoty a místo, kam se budou vkládat výplňové znaky. Rozlišujeme (uvažujme výpis čísla -1 při šířce 5):
  - zarovnání doleva (`left`) – výpis hodnoty je zarovnán doleva, výplň se vkládá zprava: `-1□□□`,
  - zarovnání doprava (`right`) – výpis hodnoty je zarovnán doprava, výplň se vkládá zleva: `□□□-1`,
  - mezi znaménko a hodnotu (`internal`) – výpis znaménka je vlevo, následují výplňové znaky a nakonec vypisovaná hodnota: `-□□□1`.

Přehled běžně používaných manipulátorů pro výpis uvádí tabulka 3.2. Na začátku jsou připomenuty dříve popsané manipulátory `endl`, `dec`, `oct` a `hex`.

Tabulka 3.2. Přehled manipulátorů

Manipulátor	Použitelnost pro čísla	Význam
`endl`	celá/reálná	ukončí řádek a nastaví kurzor na začátek následujícího řádku
`dec`	celá	přepne zobrazení celých čísel do desítkové soustavy
`oct`	celá	přepne zobrazení celých čísel do osmičkové soustavy
`hex`	celá	přepne zobrazení celých čísel do šestnáctkové soustavy
`showpos`	celá/reálná	zajistí zobrazení znaménka i pro nezáporná čísla (kladná čísla včetně nuly)
`noshowpos`	celá/reálná	vypne zobrazení znaménka pro kladná čísla (znaménko se zobrazuje pouze pro záporná čísla)
`left`	celá/reálná	přepne na zarovnávání doleva (výplň zprava)
`right`	celá/reálná	přepne na zarovnávání doprava (výplň zleva)
`internal`	celá/reálná	přepne na vyplnění mezi znaménkem a hodnotou
`fixed`	reálná	přepne výpis reálných čísel do tvaru s pevnou pozicí desetinné tečky
`scientific`	reálná	přepne výpis reálných čísel do exponenciálního tvaru
`uppercase`	celá/reálná	přepne výpis čísel tak, že znaky se zobrazují jako velká písmena (týká se výpisu celých čísel v šestnáctkové soustavě a reálných čísel v exponenciálním tvaru)
`nouppercase`	celá/reálná	přepne výpis čísel tak, že znaky se zobrazují jako malá písmena (týká se výpisu celých čísel v šestnáctkové soustavě a reálných čísel v exponenciálním tvaru)
`setw(w)`	celá/reálná	nastaví šířku následujícího výpisu dle w, chybějící znaky do požadovaného počtu jsou realizovány zvolenou výplní (pokud zadání šířky nezopakujeme před každým výpisem, bude mít následující výpis opět šířku 0)
`setfill(f)`	celá/reálná	nastaví výplňový znak na f (znakové literáry píšeme mezi apostrofy)
`setprecision(p)`	reálná	nastaví výpis reálných čísel na počet desetinných míst daných p

Poslední tři manipulátory jsou parametrické (jsou řízeny parametrem zapsaným v závorkách). Pro použití těchto manipulátorů je nutné do zdrojového textu vložit hlavičkový soubor **iomanip**:

```
#include <iomanip>
```

Pro lepší pochopení připojujeme několik příkladů použití výše popsaných manipulátorů pomocí tabulky 3.3.

Tabulka 3.3. Příklady formátování výpisu

Příklad	Výpis	Vysvětlení
`cout<<56;`	56	normální výpis kladné hodnoty (bez znaménka +)
`cout<<-56;`	-56	výpis záporné hodnoty musí vždy obsahovat znaménko –
`cout<<showpos<<56;`	+56	výpis kladné hodnoty včetně znaménka
`cout<<setw(5)<<left<<-56;`	-56□□	šířka 5, zarovnání doleva: po výpisu čísla se připojí 2 mezery
`cout<<setw(5)<<right<<-56;`	□□-56	šířka 5, zarovnání doprava: před výpisem čísla se vloží 2 mezery
`cout<<setw(5)<<internal<<-56;`	-□□56	šířka 5, zarovnání mezi: mezi znaménko a číslice se vloží 2 mezery
`cout<<fixed<<3.141519265;`	3.141519	výpis reálného čísla v desetinném tvaru
`cout<<fixed<<setprecision(3)` `<<3.141519265;`	3.142	zobrazí se 3 číslice za desetinnou tečkou (+zaokrouhlení)
`cout<<scientific<<setprecision(3)` `<<3.141519265;`	3.142e+000	výpis v exponenciálním tvaru, zobrazí se 3 číslice za desetinnou tečkou (+zaokrouhlení)

`cout<<setw(10)<<left<<setfill('0')`
`<<fixed<<setprecision(3)`
`<<3.141519265;`
	3.14200000	výpis v desetinném tvaru, 3 platné číslice za desetinnou tečkou, zbytek se doplní nulami do celkové šíře 10 znaků (zarovnání doleva)

`cout<<setw(10)<<right<<setfill('0')`
`<<fixed<<setprecision(3)`
`<<3.141519265;`
	000003.142	výpis v desetinném tvaru, 3 platné číslice za desetinnou tečkou, vlastní výpis předchází nuly do celkové šíře 10 znaků (zarovnání doprava)

Aritmetické operace s reálnými čísly

Pro reálná čísla lze používat běžné aritmetické operátory stejné jako pro celá čísla. Jsou zde však dva drobné rozdíly:

1. Operátor % (modulo, zbytek po celočíselném dělení) nemá pro aritmetiku reálných čísel smysl, jeho použití s reálným číslem je chápáno jako chyba.
2. Operátor / (dělení) má v reálné aritmetice smysl reálného dělení, tedy například výraz 5.0/2.0 má výsledek 2,5.

Přehled operátorů použitelných pro reálná čísla je uveden v tabulkách 3.4 a 3.5. V tabulce 3.4 jsou běžné binární operátory (mají levý a pravý operand), v tabulce 3.5 jsou unární operátory (mají pouze jeden operand).

Tabulka 3.4. Základní aritmetické operátory

Operátor	Příklad zápisu	Význam
+	x+y	součet dvou čísel uložených v proměnných x a y
–	x–y	rozdíl dvou čísel uložených v proměnných x a y
*	x*y	součin dvou čísel uložených v proměnných x a y
/	x/y	podíl dvou čísel uložených v proměnných x a y

Tabulka 3.5. Unární aritmetické operátory

Operátor	Příklad zápisu	Význam
+	+x	kladné znaménko čísla
–	–y	záporné znaménko čísla (násobení –1)
++	x++ *nebo* ++x	x++ postinkrement, ++x preinkrement
––	x–– *nebo* ––x	x–– postdekrement, ––x predekrement

Přípony pro rozlišení literálů reálných čísel

Podobně jako u celých čísel jsou literály pro reálná čísla typů `float`, `double` a `long double` označovány příponami `F`, `D` a `L`. Výchozí typ literálu (pokud nepoužijeme žádnou z přípon) je `float`.

Implicitní a explicitní typové konverze

Při práci s různými datovými typy se často nevyhneme konstrukci, kde v rámci jednoho výrazu použijeme proměnné nebo literály různých číselných typů. Vznikají pak tyto otázky:

- Jak překladač tento problém řeší?
- V jaké aritmetice se počítá?
- Může dojít k chybnému vyhodnocení výrazu?

Překladač může provádět dva základní typy převodů (konverzí) dat:

- **Implicitní typová konverze** – slovem implicitní označujeme případ, kdy k převodu dochází automaticky; překladač použije implicitní převod v případě, kdy je to nezbytně nutné.
- **Explicitní typová konverze** – slovem explicitní označujeme případ, kdy je převod vynucen programátorem tak, že ho v programu předepíše pomocí operátoru přetypování.

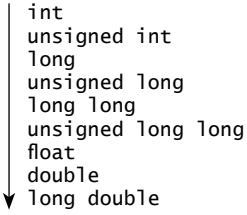
Implicitní typové konverze

K implicitní typové konverzi dochází v níže uvedených případech:

1. Konverze samostatných operandů: Před vyhodnocením výrazu se samostatné operandy převádějí takto:

- `char` nebo `short` se konvertují na `int`,
- `unsigned char` nebo `unsigned short` se konvertují na `int` (pokud `int` může reprezentovat jejich hodnotu, tzn. nepřeteče) nebo na `unsigned int` (pokud se nezdařila konverze na `int`).

2. Binární operace: Pokud mají operandy binární operace různý typ, konvertuje se typ operandu s nižší prioritou na typ operandu s vyšší prioritou. Priorita je pevně dána (typy char a short se nejdříve konvertují na int), viz obrázek 3.1 (typ int má nejnižší prioritu):



Obrázek 3.1. Priorita pro konverzi operandů binární operace

3. Převod vynucený přiřazením: V přiřazovacích příkazech je typ na pravé straně konvertován na typ na levé straně. Pokud překladač takovou konverzi nedokáže provést, označí daný řádek programu jako chybný.

Vysvětlující příklad – Uvažujeme tyto proměnné:

```
int a;
float b=3.3;
short c=2;
```

A nyní, jak se vyhodnotí výraz:

```
a=b*c;
```

Bude se postupovat v těchto krocích:

1. Použije se pravidlo pro samostatné operandy. Operand b je typu short, a proto se převede na hodnotu typu int. Operand c je typu float a převádět se nebude.
2. Operátorem * je předepsána binární operace. Operand c je typu float, a má tak vyšší prioritu. Proto se operand b dříve převedený na typ int převede ještě jednou, a to na typ float. Výsledkem operace je hodnota 6.6.
3. Nyní se výsledek výrazu přiřazuje do proměnné a, která je typu int. Použije se pravidlo pro přiřazení. Reálná hodnota 6.6 se převede na celé číslo tak, že se odsekne desetinná část (pozor: nezaokrouhluje se!). Výsledek je tedy 6 a je to hodnota typu int.

„Ale já bych chtěl při konverzi z reálného čísla na celé číslo provést zaokrouhlení!“ To není nic složitého, stačí k zaokrouhlovanému číslu přičíst hodnotu 0,5: `a=b*c+0.5;` Promyslete, proč to tak je!