Střední průmyslová škola elektrotechnická Brno, Kounicova 16

Název projektu:

Lékařská elektronika s podporou ICT

Registrační číslo:

CZ.1.07/1.1.02/03.0005

Tento produkt je spolufinancován
Evropským sociálním fondem a státním rozpočtem České republiky.

Střední průmyslová škola elektrotechnická Brno, Kounicova 16

Název: # Programovaní v jazyce C++ pro potřeby LE

Ověřovací verze č. 1

Zpracovala:

Ing. Dagmar Holková

Tento produkt je spolufinancován
Evropským sociálním fondem a státním rozpočtem České republiky.

Učební text neobsahuje úplný popis všech možností jazyka C++. Text je přizpůsoben výuce programování na SPŠE Brno dle ŠVP.
Řešené úlohy a zadání příkladů k řešení najdete v pracovních listech.

Tento produkt je spolufinancován
Evropským sociálním fondem a státním rozpočtem České republiky.

Obsah

1. Úvod

Učební text je určen pro seznámení se základy práce v jazyku C++ a nepředpokládá předchozí znalosti programování. Text je doplněn sadou cvičení s vyřešenými příklady a zadáním příkladů k samostatnému řešení.

Texty spolu s příklady mají pomoci žákům při samostatném studiu v případě potřeby doplnění učiva.

Důležité pojmy jsou vyznačeny ***tučnou kurzívou***.

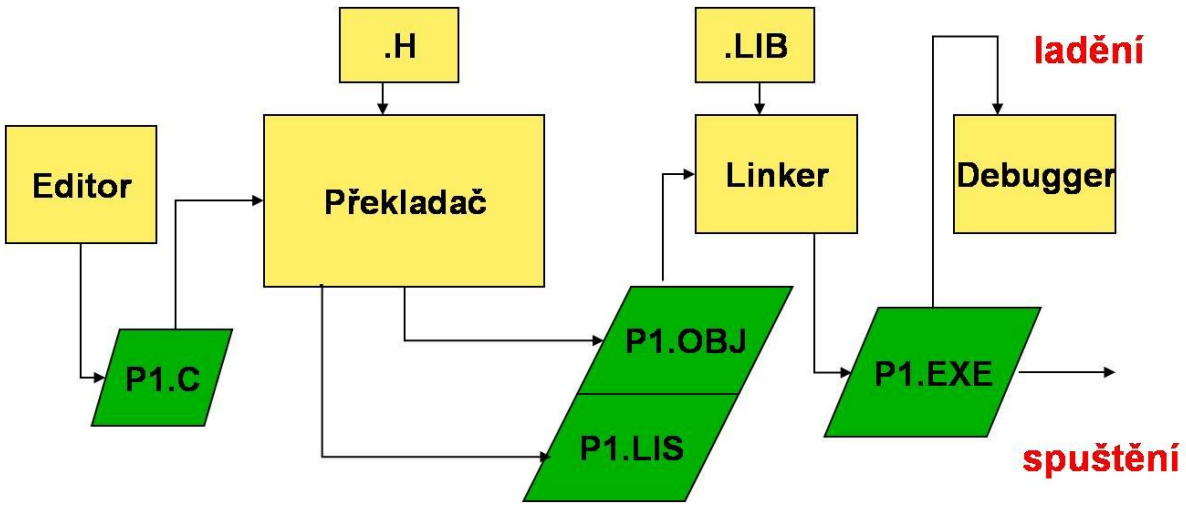
2. Vývoj jazyka C++

Na počátku byl programovací jazyk C autorů Briana W. Kernighana a Denise M. Ritchieho (kolem roku 1980). Když se objektově orientovaná analýza, návrh a programování stávaly stále více populární, vzal Bjarne Stroustrup nejoblíbenější jazyk pro vývoj komerčního softwaru – jazyk C a rozšířil ho o nezbytné vlastnosti umožňující objektově orientované programování. Jazyk C++ je tedy nadmnožinou jazyka C a každý platný program v jazyce C je platným programem i v jazyce C++. Oba jazyky mají hodně společného, ale liší se přístupem k řešení problému: jazyk C vychází ze strukturovaného programování, zatímco jazyk C ++ je založen na objektově orientovaném programování.

Jazyk C++ nabízí nástroje pro objektově orientované programování a výkon na úrovni systémového jazyka. Programovací jazyk C++ se stal jedním z nejvýznamnějších programovacích jazyků devadesátých let 20. století a jeho obliba pokračuje i na počátku 21. století.

3. Zpracování programu

Průběh zpracování programu v C++ můžeme schematicky znázornit takto:



Textový procesor – píšeme a opravujeme v něm program

Překladač (kompilátor) – provádí překlad zdrojového souboru (našeho programu) do kódu počítače. Vznikají dva soubory *.OBJ je náš přeložený program a *.LIS protokol o překladu, který obsahuje informace o chybách nalezených překladačem

Linker (sestavovací program) – přidělí relativním adresám adresy absolutní a provede všechny odkazy (např. na funkce obsažené v knihovnách). Výsledkem je spustitelný program *.EXE

Debugger (ladící program) – hledá chyby při běhu programu. Po nalezení chyby se celý proces opakuje tak dlouho, dokud program není bez chyb.

4. Struktura programu, základní příkazy

4.1. Postup při vytváření programu

Pokud již známe zadání úlohy a pochopili jsme požadavky na vstupy do programu a výstupy z programu, můžeme přejít k řešení úlohy. Prvním krokem při řešení úlohy by měl být návrh postupu jednotlivých kroků např. formou zápisu algoritmu ve tvaru vývojového diagramu. Tento postup není nutný u jednodušších úloh, ale určitě se vyplatí při řešení těch složitějších.

4.2. Pravidla zápisu programu

Zápis programu musí odpovídat pravidlům jazyka C++. V jazyku C++ se všechno řeší pomocí funkcí.

Příklad jednoduchého programu:

```
/* vypis textu na obrazovku */                    //poznámka  – zadání programu
#include <iostream>                                //  připojení vstupů a výstupů
```

```
using namespace std;                                    //  připojení příkazů v/ v
int main()                                              //  hlavní funkce
{                                                       //  začátek těla funkce
        cout << "Muj prvni program! ";                  //  výpis textu na obrazovku
        cout << "\n";                          // ukončeni výpisu řádku a přechod na nový řádek
        system  "pause";                       // podrží text na obrazovce do stisku klávesy
        return 0;                                 // ukončení programu a vrácení hodnoty 0
}                                                       //  konec těla funkce
```

#include <iostream> - jazyk C ++ neobsahuje sám prostředky pro vstup a výstup dat. Abychom mohli pracovat se vstupy a výstupy, je nutné připojit soubor ***iostream***, který všechny tyto funkce obsahuje. # - první znak je signál pro preprocesor. Preprocesor se spustí při každém spuštění kompilátoru. Projde si zdrojový kód a najde řádky, které začínají symbolem #. Příkaz include je instrukce pro preprocesor, která říká: To, co následuje je jméno souboru. Najdi tento soubor a vlož do tohoto místa v programu. Lomené závorky kolem názvu souboru říkají, že se má soubor hledat v nastavených adresářích

int main() - je celočíselný datový typ; u názvu funkce main() říká, že funkce bude vracet hodnotu typu celé číslo

using namespace std; - umožní připojení příkazů pro vstup a výstup z programu

cout << " Muj prvni program! "; - příkaz výstupu - vypíše text v uvozovkách na obrazovku, kurzor je na staven za textem

cout << "\|n"; - ukončení výpisu řádku a přechod na nový řádek

system "pause"; - příkaz zastaví výpis programu na obrazovce, abychom si ho mohli prohlédnout

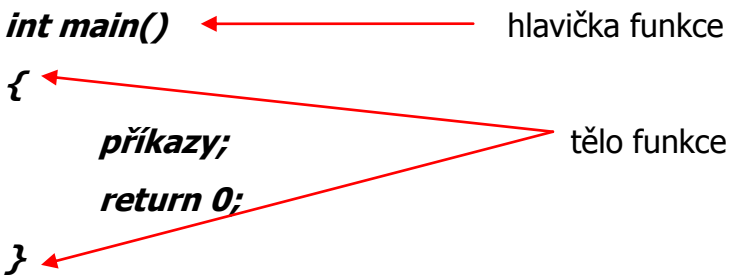
return 0; - poslední příkaz programu ; vrací jako výsledek hodnotu 0 - pro operační systém je hodnota 0 signálem normálního ukončení programu.

Závorky { a ***}*** uzavírají hlavní program nebo složený příkaz. Za každou závorkou { mohou být definice proměnných.

4.2.1. Funkce main()

Hlavní program představuje funkce ***main()*** , která musí být v programu vždy uvedena. Je to první funkce, která je v programu volána a je nutné udat její typ(= hodnota, kterou funkce vrací). V každém programu může být uvedena jen jednou.

Ukázka základní struktury funkce main():



Hlavička funkce bude stejná u všech programů. Tělo funkce tvoří instrukce neboli příkazy. Tyto příkazy se budou měnit podle zadání, které budeme řešit.

4.2.2. Příkazy programu

Tělo programu bude tvořeno příkazy. Příkaz představuje kompletní počítačovou instrukci. Každý příkaz musí být ukončen středníkem, aby překladač poznal, kde jednotlivá instrukce začíná a končí.

4.2.3. Komentáře

Komentáře slouží k uvedení poznámek ve vašem programu. Jazyk C++ používá komentáře do konce řádku:

// tento komentář platí jen do konce řádku

Komentář může stát samostatně na řádku nebo může být umístěn na konec řádku s příkazem, který vysvětluje. Překladač poznámky přeskakuje. Poznámky slouží programátorovi k připomenutí postupu při tvorbě programu. Oceníme je zvláště u

složitějších programů, nebo pokud se vracíme k programu po delší době a chceme si jej připomenout.

Doporučení: Každý program bude začínat komentářem, který poskytne stručný popis programu.

Jazyk C++ rozumí i komentářům jazyka C:

/ toto je příklad použití
víceřádkového komentáře ****/***

4.2.4. Hlavičkový soubor iostream

Jazyk C++ neobsahuje sám prostředky pro vstup a výstup dat. Abychom mohli pracovat se vstupy a výstupy, je nutné připojit soubor ***iostream***, který všechny tyto funkce obsahuje. # - první znak je signál pro preprocesor. Preprocesor se spustí při každém spuštění kompilátoru. Projde si zdrojový kód a najde řádky, které začínají symbolem #. Příkaz ***include*** je instrukce pro preprocesor, která říká: To, co následuje je jméno souboru. Najdi tento soubor a vlož do tohoto místa v programu. Lomené závorky kolem názvu souboru říkají, že se má soubor hledat v nastavených adresářích. Příkazy

```
#include <iostream>
using namespace std;
```

umožní používat příkaz vstupu a výstupu v programu.

Ve starších programech se můžete setkat se zápisem:

```
#include <iostream.h>
```

Podle nového stylu C++ se nepoužívají žádné přípony u hlavičkových souborů.

4.2.5. Složené závorky

Složené závorky ohraničují začátek a konec programu. Používají se také k ohraničení složeného příkazu.

4.2.6. Příkazy vstupu a výstupu

Příkazy vstupu (***cin***) a výstupu (***cout***) jsou součástí standardní knihovny – std. Příkaz ***cout*** použijeme pro výstup na obrazovku a používáme takto: Napíšeme slovo cout a za něj operátor přesměrování výstupu **<<**. Cokoli následuje za operátorem přesměrování výstupu, objeví se na obrazovce. Jestliže chceme vypsat text, je nutné ho napsat do uvozovek:

cout << " Muj prvni program! ";

Pokud chceme přejít na nový řádek, použijeme příkaz:

cout << "|n";

I když píšeme dva znaky, přečtou se jako jeden znak. Kdykoliv je použijeme ve výstupním proudu, způsobí přechod kurzoru na nový řádek:

Př.:

cout << " Prvni radek.|n Druhy radek.|n";

Příkaz vypíše text ve tvaru:

První radek.

Druhy radek.

V některých příkladech se nám může hodit i další funkce pro výstup printf(). Tato funkce umožní naformátovat výstup podle typu proměnných, případně omezit jejich velikost:

Př.:

int cislo;

cin>> cislo; *// načtení hodnoty 25 do proměnné cislo*

printf(" Vase cislo je %d!",cislo);

Příkaz printf() vypíše text ***Vase cislo je*** a místo formátovaného výstupu ***%d*** doplní hodnotu uloženou v proměnné ***cislo***: ***Vase cislo je 25!***

Stručný přehled formátů výstupu:

Celá čísla	***%d***
Desetinná čísla	***%f***
Řetězec znaků(slovo)	***%s***

Příkaz ***cin*** využijeme pro vstup dat z klávesnice:

int cislo;

cin>> cislo; *// načtení hodnoty 25 do proměnné cislo*

Napíšeme slovo ***cin,*** za něj operátor přesměrování vstupu **>>** a jméno proměnné, do které hodnotu načítáme **cislo**. Funkce ***cin*** převede skupinu znaků na vstupu do tvaru odpovídajícímu typu vaší proměnné – v našem příkladě celočíselné proměnné.

4.2.7. Styl zápisu programu

Základní pravidla zápisu programu:

Jeden příkaz na řádek.

Složené závorky uvozující začátek a konec programu píšeme na samostatné řádky.

Klíčová slova jazyka C++ píšeme malými písmeny.

Příkazy v těle funkce píšeme odsazeně od složených závorek.

4.2.8. Deklarace proměnných

V programech budeme pracovat s hodnotami, ať už načtenými nebo vypočítanými. Počítač od vás bude požadovat upřesnění informací o každé hodnotě – její jméno a typ (celé číslo, desetinné číslo …), aby věděl, kolik paměti bude potřebovat na její uložení.

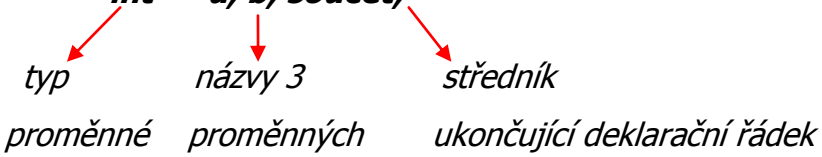
> ***Jméno proměnné*** je název, kterým budeme hodnotu označovat. Měl by vystihovat její obsah nebo použití např. suma, pocet, prumer, max, min, nazev apod. Snažte se vyhnout nic neříkajícímu označení a, b, c, d …

Pro název proměnné platí tato ***pravidla***:

- ***jako název proměnné nelze použít klíčové slovo jazyka C++***
- ***rozlišují se velká a malá písmena (např. POM, Pom, pom jsou tři různé proměnné!)***
- ***název může obsahovat alfabetické znaky, číslice a podtržítka, ale nesmí začínat číslicí!***
- ***délka názvu není omezena, ale pokud zvolíte dlouhý název, musíte jej vždy správně zapsat! Překladač vás upozorní na chybu a budete muset opravovat a opravovat …***
- ***zápis jména proměnné bez české diakritiky***
- ***každé jméno proměnné musí být jedinečné - > nemůžete jedním jménem označit více proměnných***
- ***můžeme deklarovat více proměnných stejného typu jedním deklaračním příkazem – používáme tzv. vícenásobnou deklaraci***

Deklaraci proměnných provádíme na začátku programu. Tím si usnadníme hledání a kontrolu jmen proměnných. Doporučuji deklarovat na jednom řádku proměnné stejného typu:

např. ***int a, b, soucet;***



nebo

int a,b;

float soucet, prumer;

4.2.9. Příkaz přiřazení

Příkaz přiřazení je nejčastěji používaným příkazem. Obecně jej můžeme popsat:

proměnná = výraz;

např. *i = 3;*

d = ‘z ‘;

s = 3.14*r*r;

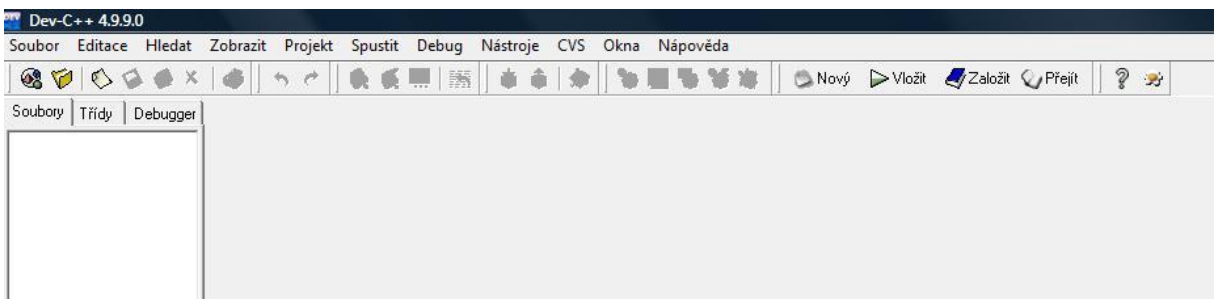
Pro porovnání používáme v C++ příkaz ==

Vícenásobné přiřazení: i = j = k = 2; se vyhodnotí zprava doleva i =(j =(k = 2));

5. Prostředí Dev-C++

5.2. Spuštění Dev-C++

Jako první se seznámíme s pracovním prostředím. Po spuštění programu se objeví následující obrazovka:



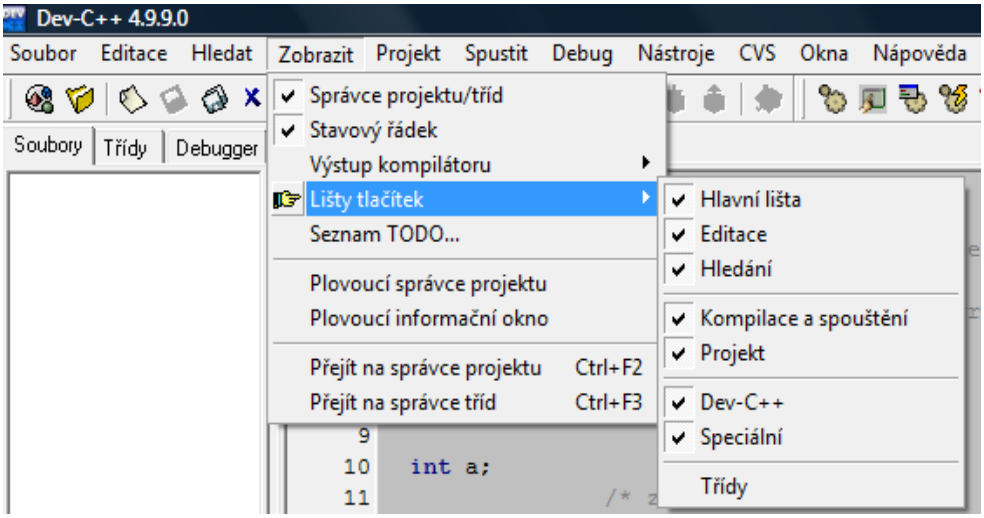
Možnosti hlavní nabídky:

Soubor - vytvoření, otevření nebo uložení souboru.

Editace – zprostředkuje práci s textem – např. krok zpět, vyjmout, kopírovat, vložit atd.

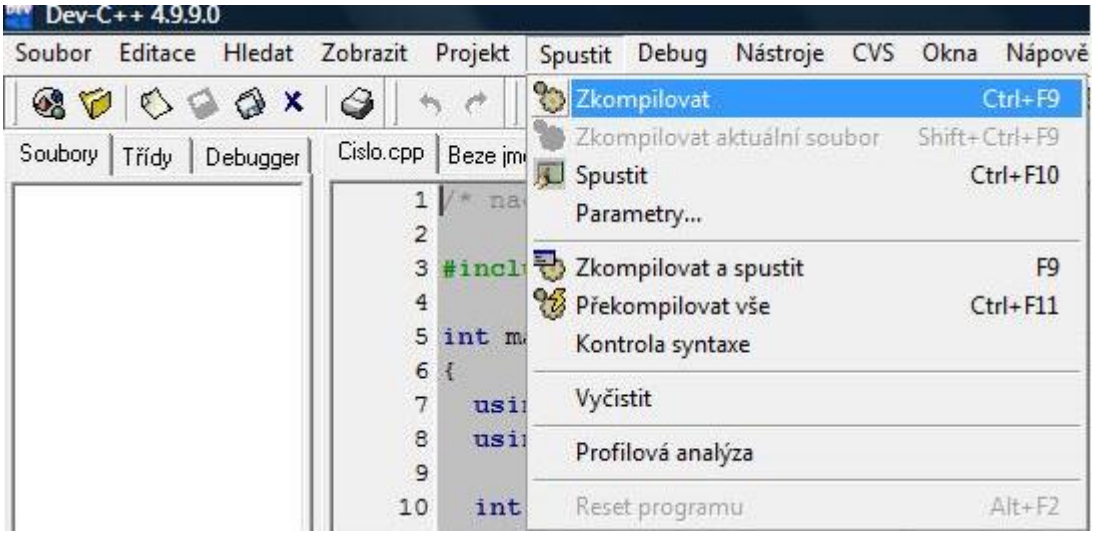
Hledat – obsahuje nástroje pro vyhledání pojmu, rychlé hledání, nahrazení výrazu jiným atp.

Zobrazit – umožní nastavení lišty tlačítek, zobrazení výstupu překladače atd.

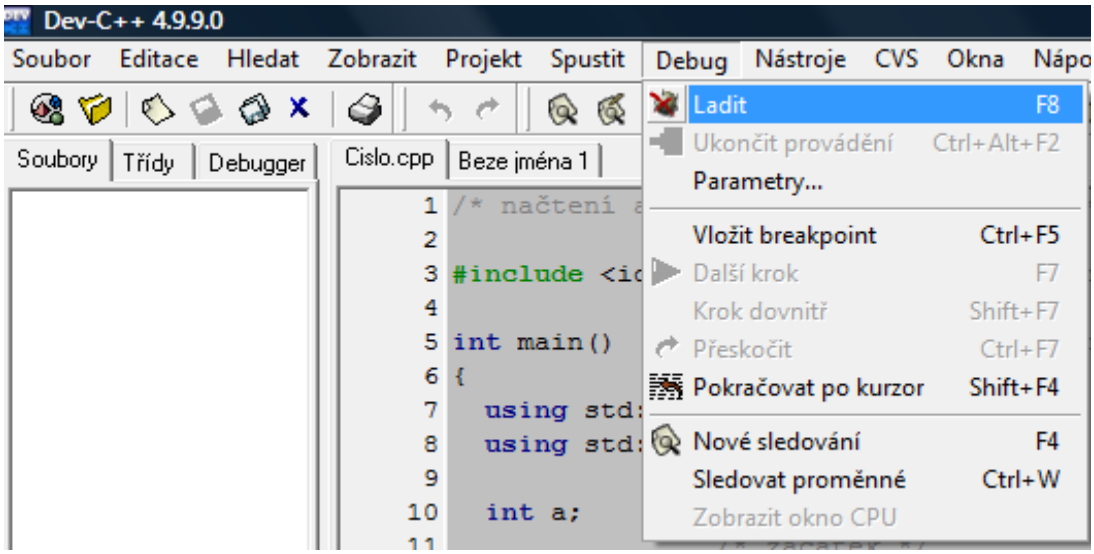


Projekt – nabídka pro práci s projektem

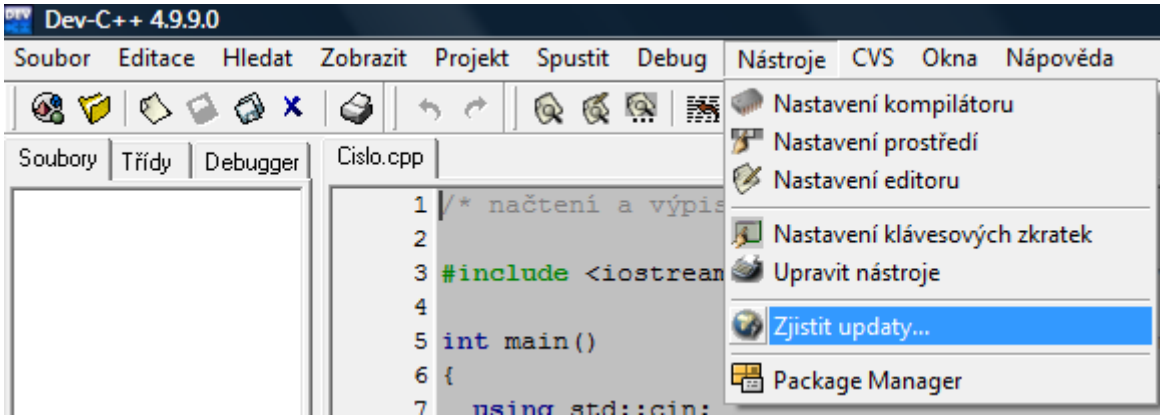
Spustit – pomocí této nabídky můžeme zkontrolovat správnost zápisu programu a jeho následné spuštění (F9) – lze provést i kliknutím na příslušnou ikonu v liště tlačítek



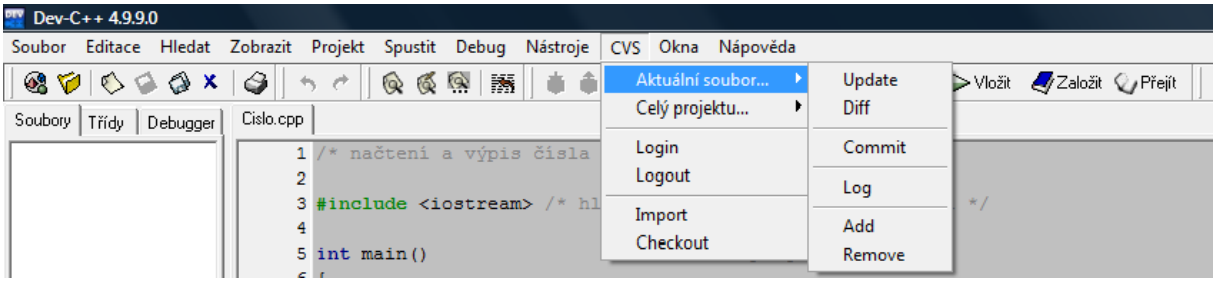
Debug – ladění programu



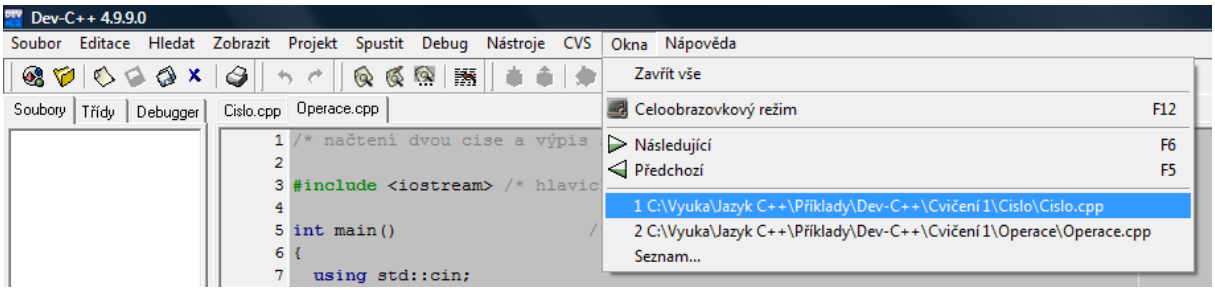
Nástroje



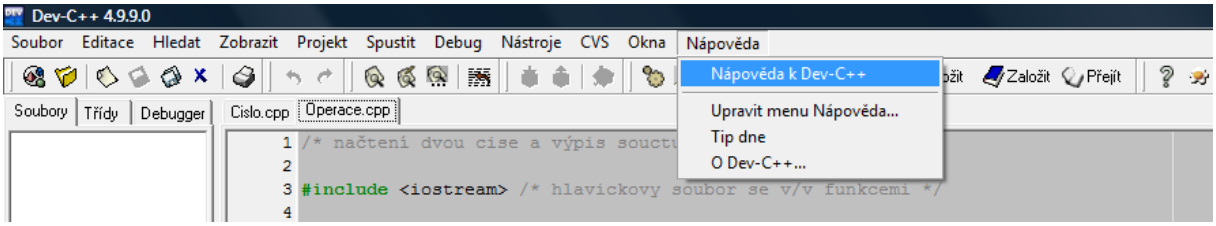
CVS



Okna

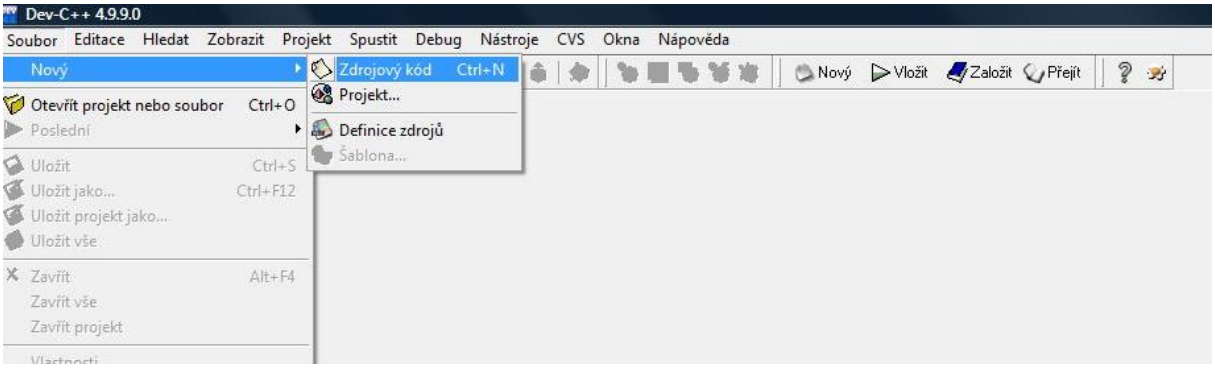


Nápověda

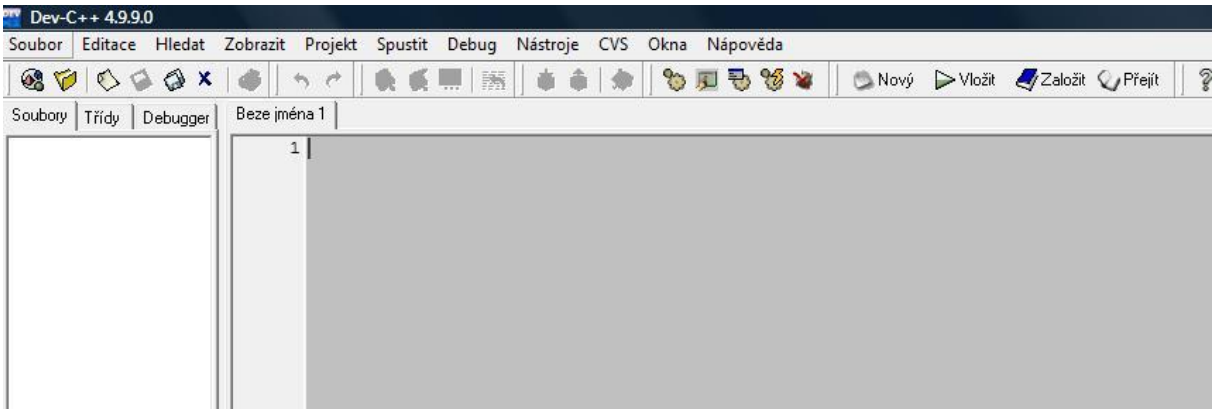


5.2. Zápis nového programu

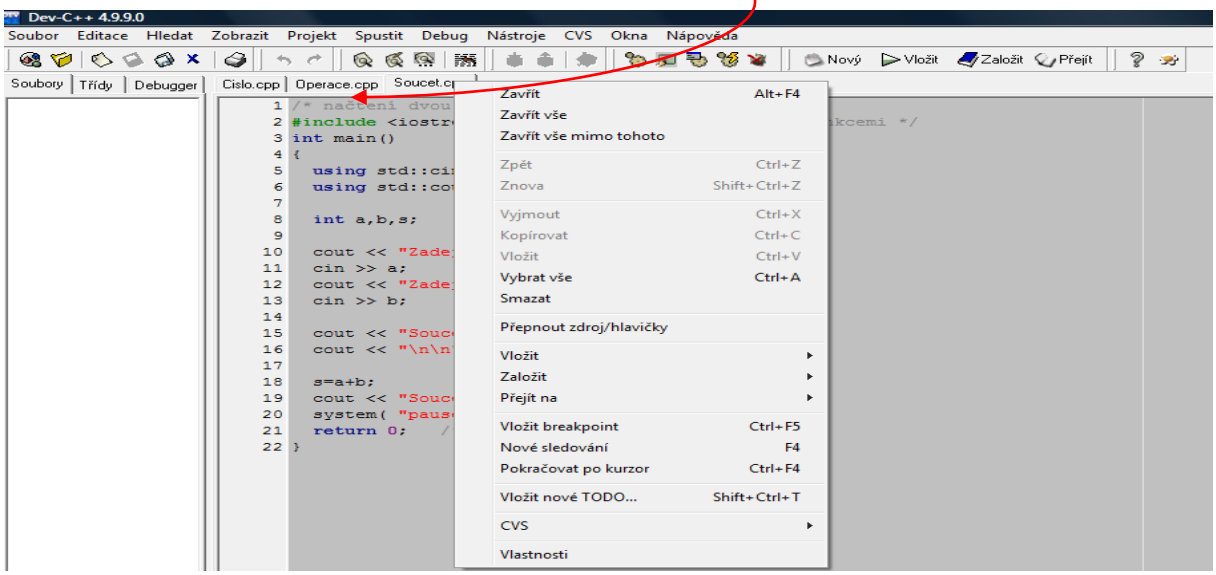
Z hlavní nabídky vybereme možnost ***Soubor***, dále ***Nový*** a pak ***Zdrojový kód(CTRL+N)***. Otevřeme tak nový soubor v textovém editoru a můžeme začít psát první řádek programu:



Potvrdíme volbu a pokračujeme zápisem programového kódu:



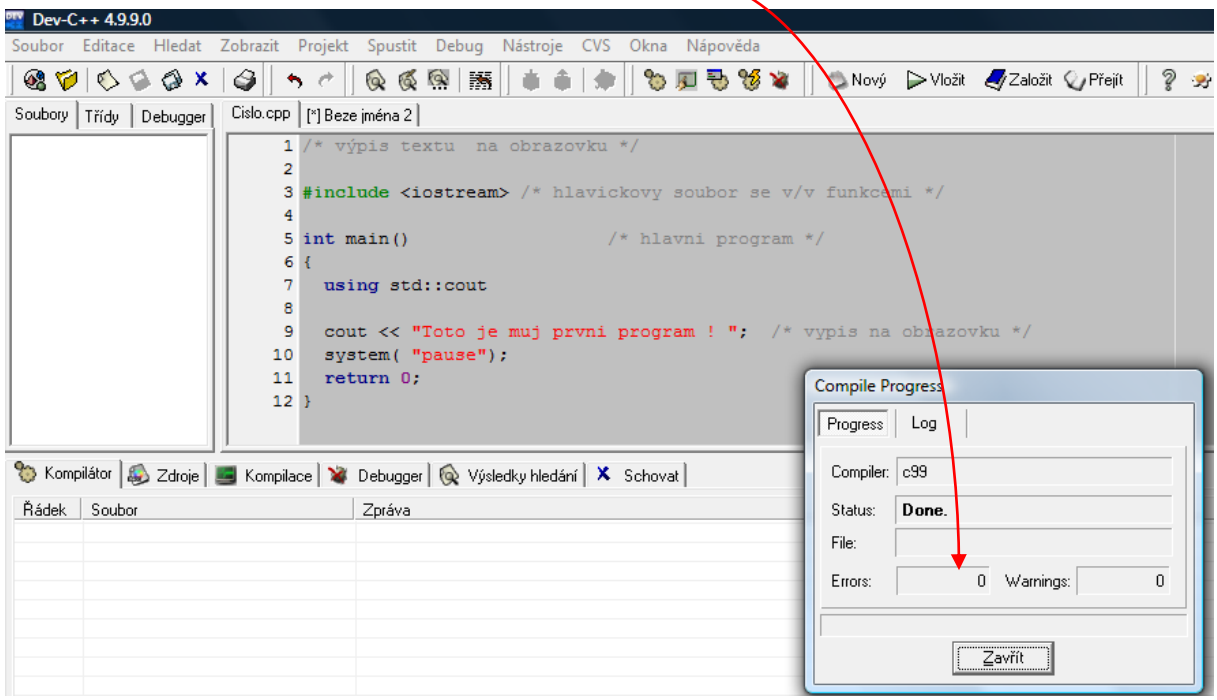
Pokud již pracujeme v prostředí Dev-C++, můžeme současně otevřít více programů. Jejich názvy se umístí do záložek listů nad pracovní plochou a my můžeme využít pomocnou nabídku pro práci s programem:



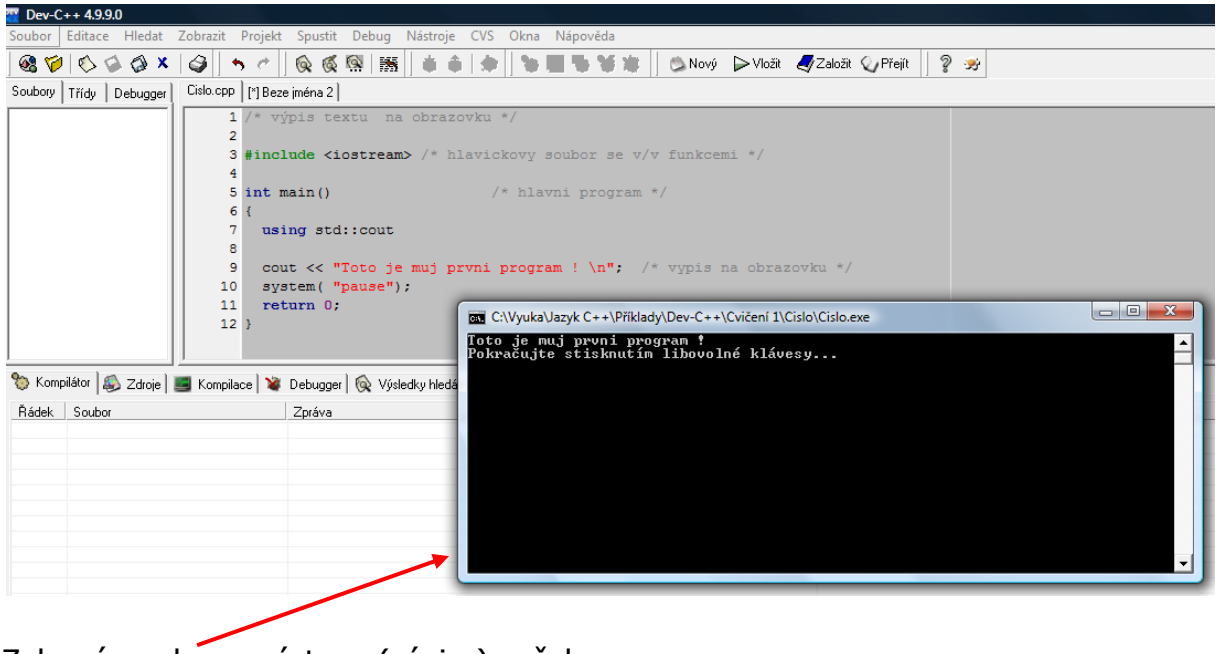
Vyvoláme ji nastavením na příslušnou záložku a stisknutím pravého tlačítka na myši.

5.3. Překlad a spuštění

Pokud jsme dokončili zápis programu, spustíme překlad. Překladač zkontroluje správnost zápisu programu:



Pokud program neobsahuje syntaktické chyby (rozuměj - je zapsán podle pravidel C++), můžeme program spustit: Nabídka ***Spustit –>Spustit*** nebo ***Ctrl+F10***



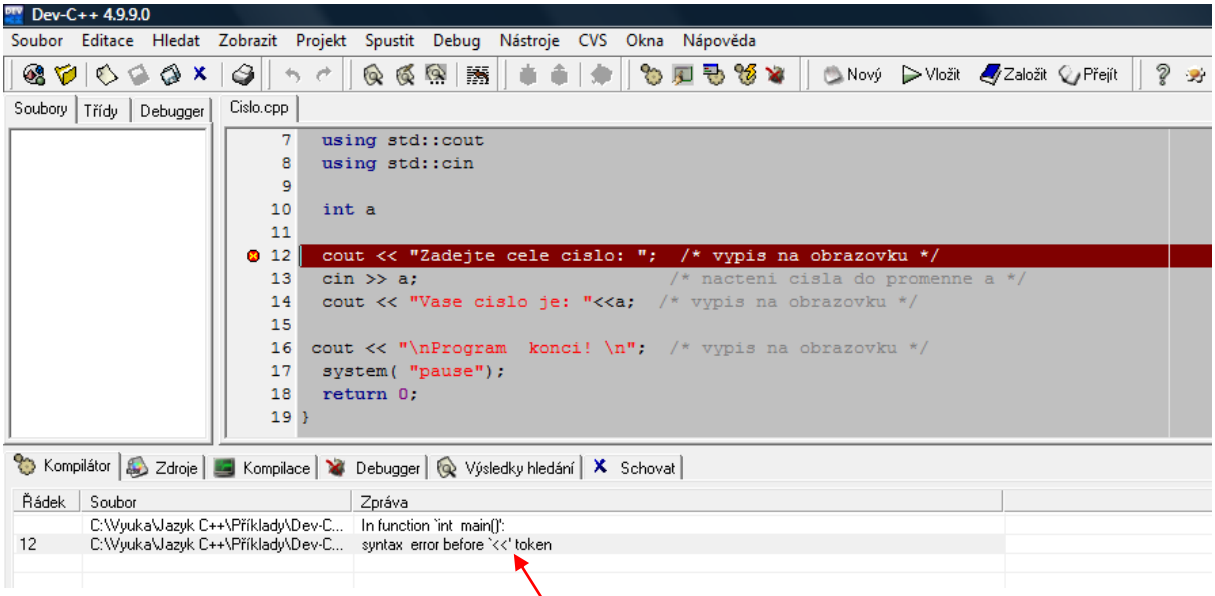
Zobrazí se okno s výstupy (výpisy) našeho programu

Můžeme také využít klávesy ***F9***, která provede ***překlad a spuštění***.

5.4. Chyby v zápisu programu

Program se nám podaří napsat bez syntaktických chyb většinou jen výjimečně.

Častěji se budeme setkávat s hlášením chyb:



Pod editačním oknem se objeví výpis chyb, zjištěných při překladu:

Číslo řádku	Cesta s názvem programu	Popis chyby
12	C:\Vyuka\Jazyk C++\Příklady\Dev-C++\Cvičení 1\Cislo\Cislo.cpp	syntax error before `<<' token

Když opravujete chyby, tak opravte nejprve tu první. Pokud ji nemůžete najít na vyznačeném řádku, hledejte na předchozím. Pokračujeme opravou textu programu tak dlouho, dokud zápis programu není bez chyb. Bezchybný překlad umožní spuštění programu a testování na vstupních datech. Jestliže výsledky programu neodpovídají požadavků zadání úlohy, znamená to, že náš program obsahuje logické chyby – chyby v řešení úlohy. Tyto chyby si musí programátor najít v programu sám, kompilátor je nedokáže odhalit!

6. Jednoduché datové typy

Ve většině programů budeme potřebovat proměnnou, do které bychom mohli uschovat načtenou nebo vypočítanou hodnotu. Máme na výběr ze 4 jednoduchých datových typů: celočíselný

desetinné číslo

znakový

boolean

Přehled typů proměnných:

Typ	**Velikost**	**Hodnoty**
bool	1 bajt	true *nebo* false
unsigned short int	2 bajty	0 až 65 535
short int	2 bajty	-32 768 až 32767

unsigned long int	4 bajty	0 až 4 294 967 295
long int	4 bajty	-2 147 483 648 až 2 147 483 647
int (16 bitů)	2 bajty	-32 768 až 32767
int (32bitů)	4 bajty	-2 147 483 648 až 2 147 483 647
unsigned int (16 bitů)	2 bajty	0 až 65 535
unsigned int (32 bitů)	4 bajty	0 až 4 294 967 295
char	1 bajt	256 znakových hodnot
float	4 bajty	1,2e-38 až 3,4e38
double	8 bajtů	2,2e-308 až 1,8e308

Poznámka:

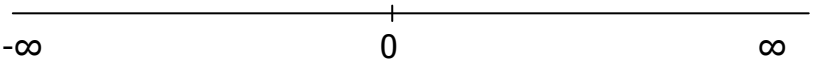
Velikost proměnných se může lišit v závislosti na typu kompilátoru a počítače. Aktuální velikosti zjistíme pomocí funkce ***sizeof(proměnná)****, která vrací velikost typu nebo proměnné v bajtech. Ukážeme si ve cvičení.*

Název proměnné by měl vystihovat její použití v programu (např. soucet, suma, počet, pocitadlo, soucin apod.) a měl by odpovídat pravidlům jazyka C++ (uvedena v kap. 4.2.8).

Doporučuje se vhodně volit název proměnné tak, aby odpovídal jejímu použití v programu. Např. názvy a,aa, b, atd. vám po delší době těžko připomenou, proč jste je v programu použili a jaké hodnoty jste do nich ukládali. Takové názvy volíme v případě indexů např. i,j apod.

6.1. Celočíselný datový typ

Celá čísla obecně představují nekonečně mnoho čísel:



Paměť počítače je však omezena a tím je dána i podmnožina požitých celých čísel. Základní celočíselné typy seřazené podle velikosti jsou ***short***, ***int*** a ***long***.

Příklad deklarace:

```
int     cislo1,cislo2, soucet;

        cislo1 = 3;
        cislo2 = 8;
        soucet = cislo1 + cislo2;
```

V ukázce je použita ***vícenásobná deklarace***, tj. současně zavádíme více proměnných stejného typu. Vzájemně je oddělujeme čárkou a deklaraci ukončíme středníkem. Následně můžeme proměnné použít v programu – načíst do nich hodnoty nebo uložit výsledek výpočtu. Většinou vystačíme s typem ***int***.

6.2. Desetinná čísla

Někdy se objeví požadavek na práci s desetinnými čísly např. výpočty objemu, obsahu geometrických obrazců nebo těles, případně výpočty odmocnin atd. V těchto případech musíme použít proměnné typu desetinné číslo, protože do celočíselné proměnné by se nám hodnota nevešla. Desetinné typy seřazené podle velikosti jsou ***float*** a ***double***. Většinou budeme používat typ ***float***.

Příklad deklarace:

```
float   strana,obvod, obsah;

        strana = 4.5;
        obvod= 4 * strana;
        obsah= strana *  strana;
```

6.3. Znakový datový typ

Znakový datový typ je určen pro ukládání znaků např. písmen a číslic. Obecně lze do takovéto proměnné uložit jakýkoliv znak z ASCII tabulky.

ASCII tabulka – znaková sada, se kterou pracujeme. Má rozsah 0-255, ale většinou pracujeme jen s dolní polovinou tabulky znaky od 0 do 127. Horní polovina je vyhrazena pro znaky národních abeced.

Dolní část ASCII obsahuje:

řídící znaky	0		31	*neviditelné*
mezera	' '		32	
pomocné znaky	33	'! '	47	'/ '
číslice	48	'0 '	57	'9 '
pomocné znaky	58	': '	64	'@'
velká písmena	65	'A '	90	'Z'
pomocné znaky	91	'['	96	'' '
malá písmena	97	'a '	122	'z '
pomocné znaky	123	'{ '	126	'~ '

6.3. Typ boolean

Jazyk C dříve vyjadřoval nenulovou hodnotu jako *pravda* a nulovou hodnotu jako *nepravda*. Nyní můžeme zapisovat ***pravdu*** jako ***true*** a ***nepravdu*** jako ***false***.

6.4. Konstanty

Konstantou budeme v programu chápat specifickou proměnnou, která bude obsahovat po celou dobu trvání programu stejnou hodnotu. Celočíselné konstanty mohou vyjadřovat číslo v soustavě:

Desítkové	první číslice v rozsahu 1-9	např. 34
Osmičkové	první číslice je 0, další v rozsahu 1-7	např. 023
Šestnáctkové	první dva znaky 0x (0X), pak A-F,0-9	např. 0xA2

Příklad deklarace:

const float pi = 3.14;

float r,obvod, obsah;

r = 7.5;
*obvod= 2 *pi*r;*
*obsah= pi * r*r;*

7. Aritmetické a logické operátory

7.1. Aritmetické operátory

Mezi základní aritmetické operace řadíme sčítání, odčítání, násobení, dělení, zbytek po celočíselném dělení. Všechny operace používají dva operandy a jednu operaci:

Sčítání ***soucet = a + b***

Odčítání ***rozdil = a – b***

Násobení ***soucin = a * b***

Dělení ***podíl = a / b***

Zbytek po celočíselném dělení ***zbytek = a % b***

Pokud jsou oba operandy (a,b) celočíselné, je i výsledek celé číslo. Jestliže je alespoň jeden operand desetinné číslo, výsledek bude desetinné číslo!

Zkrácené zápisy a jejich význam:

a += 5	***a = a + 5***
a -= 5	***a = a – 5***
a *= 5	***a = a * 5***
a /= 5	***a = a / 5***

a %= 5 ***a = a % 5***

7.2. Priorita operátorů

Aritmetické operátory dodržují obvyklé aritmetické pořadí: násobení, dělení a zbytek po celočíselném dělení se provádějí před operacemi sčítání a odčítání.

Př.

*c = b * a - 2*

Pokud si nejste jisti pořadím aritmetických operací, použijte závorky:

Př.

*b = b * (a-2)*

7.3. Speciální operátory

inkrement ++ (zvýšení o hodnotu 1)

dekrement -- (snížení o hodnotu 1)

Rozlišujeme jejich použití podle zápisu:

++výraz - zvětšení před použitím

výraz++ - zvětšení po použití

Např. *a = 2, b = 7*

a++; *a = 3*

b = ++a; *a = 3* *b = 3*

b = a++; *a = 3* *b = 2*

c = --b + 5; *c = --b + 5 = 6 +5 = 11* *a = 2* *b = 6* *c = 11*

Upozornění:

Inkrementujte nebo dekrementujte stejnou proměnnou pouze jednou v jednom příkazu! Jinak by se zápisy mohly stát nejednoznačnými a nepřesnými! (např. b = 3 * a++ * (5- ++a))

Př. *a = 5 b = 8*

b += a *b= b + a = 8 +5 = 13*

b /= --a *b = b / --a = 8 / 4 = 3*

*b *= a – 2* *b = b * (a-2) = 8 * (5 - 2) = 24*

7.4. Relační a logické operátory

Relační operátory používáme při porovnávání čísel nebo proměnných:

Operátor	zápis	
==	rovná se	(! Často se zaměňuje za dosazovací příkaz =)
!=	nerovná se	
>	větší	
>=	větší nebo rovno	
<	menší	
<=	menší nebo rovno	

Logické operátory používáme k vytváření složených podmínek:

&&	logický součin
||	logický součet
!	negace

Př.

if (a > 10) && *(a <= 55)* *dotaz, jestli a leží v intervalu (10, 55>*

7.5. Vyhodnocování logických výrazů

Uvedeme si priority vyhodnocování logických výrazů:

! ++ -- - +	zprava doleva
**** / %***	zleva doprava
+ -	zleva doprava
< <= >= >	zleva doprava
== !=	zleva doprava
&&	zleva doprava
||	zleva doprava
? :	zprava doleva
= += -= *= atd.	zprava doleva
'	zprava doleva

V jazyce C mají aritmetické operátory vyšší prioritu než logické! Pokud si nejste jisti, doporučuji použít závorky!!

8. Podmíněné příkazy

8.1. Podmíněný příkaz if

Často se v programech setkáme s potřebou rozhodnout se na základě vyhodnocení určité podmínky. K tomu slouží příkaz if. Zápis příkazu if:

```
if (podmínka)
    příkaz 1;
else
    příkaz 2;
```

Na základě vyhodnocení podmínky se provede příkaz 1, pokud je podmínka splněna; v opačném případě příkaz 2.

Př.

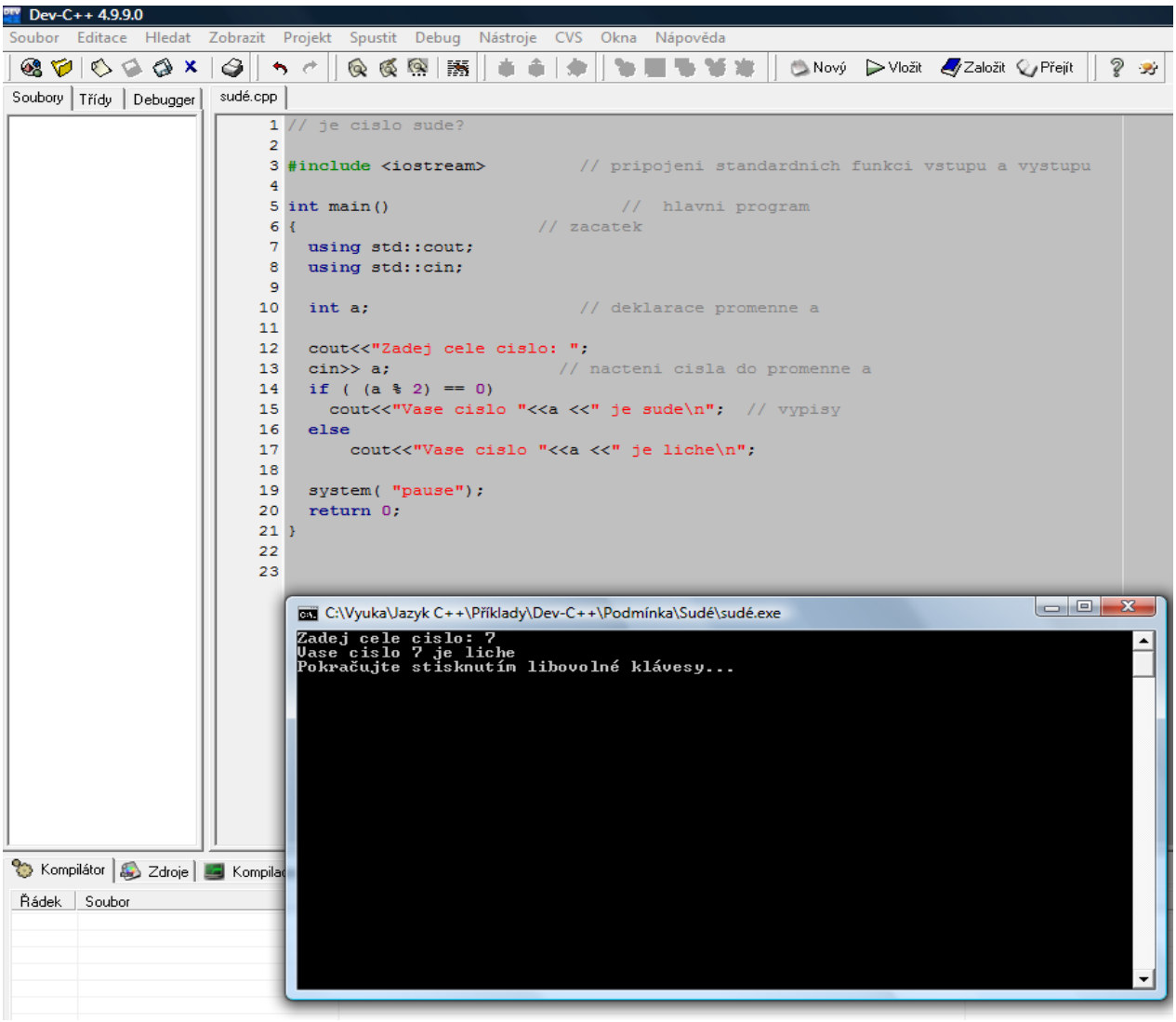
a) neúplná podmínka

```
if (a > b)          // závorky jsou povinné
    c = a;
```

b) úplná podmínka

```
if  (a > b)         // závorky jsou povinné
    max = a;
else
    max = b;
```

Použití v příkladu:



Pokud potřebujete v kterékoliv větvi příkazu ***if*** zapsat více příkazů, použijte ***složené závorky*** pro označení složeného příkazu!

Př.

```
if ( a > b)
   {
     max = a;
     c = c + a;
   }
else
  {
    max = b;
   c = c + b;
  }
cout <<" Vetsi je "<< max<< "|n";
```

8.2. Přepínač switch

Zvláštním případem podmíněného příkazu je přepínač. Používáme ho k vícenásobnému větvení programu.

Nejdříve si vysvětlíme příkaz break:

break – ukončuje větev, okamžitě opouští příkaz switch, který ho vyvolal a pokračuje se dalšími příkazy v programu

Zápis příkladu switch:

```
switch (a)  {
            case  hodnota1 : příkaz1; break;
            case  hodnota2 : příkaz2; break;
                     default : příkaz ;break;
           }
```

Při jeho zápisu musíme dodržet následující pravidla:

- ***výraz, podle kterého se přepínáme musí být celé číslo***
- ***nelze zapsat více hodnot pro jednu větev***
- ***každá větev musí být ukončena příkazem break***
- ***je možné použít větev default, kterou budeme procházet, pokud se neprovede žádná výše uvedená větev***
- ***větev default nemusí být uvedena a pokud je uvedena, není nutné jí zapisovat jako poslední.***

Př.:

```
switch (a)   {
                case  `+` : cout <<" Soucet je "<< x + y; break;
                case  `-` : cout <<" Rozdil je "<< x – y; break;
                default : cout<<"Chyba ! \n";break;
              }
```

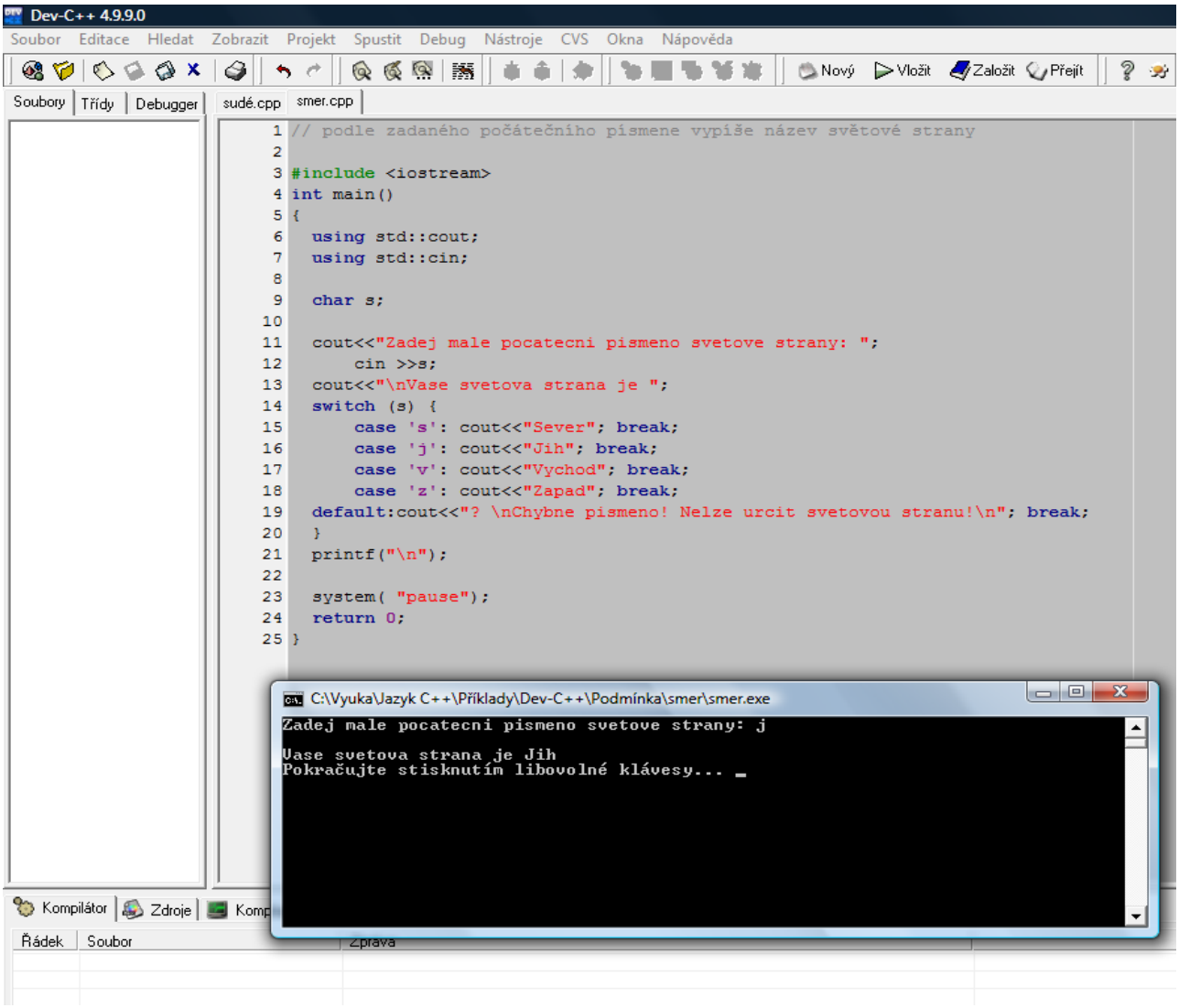
Pokud potřebujeme zapsat více hodnot, pro které se provedou příkazy v jedné větvi, je možné použít tento zápis:

```
switch ( x )
{
    case  hodnota1 :
    case  hodnota2 :
    case  hodnota3 :
    case  hodnota4 : příkaz1; break;
    case  hodnota5 : příkaz2; break;
    default : cout<<"Chyba ! \n";break;
}
```

Pro x = hodnota1 nebo hodnota2 nebo hodnota3 nebo hodnota4 se provede příkaz1.

Pro x= hodnota5 s provede příkaz2.

Použití v příkladu:



9. Cykly

9.1. Pojem cyklus

Používáme je k opakování jednoho nebo více příkazů, na základě vyhodnocení určité podmínky. Často potřebujeme v programu opakovaně načítat nebo vypisovat hodnoty. V jazyce C++ rozlišujeme 3 typy cyklů: ***for, while, do-while***

Při používání cyklů je důležité si podle zadání vybrat vhodný typ cyklu – počet průchodů cyklem; jestli může nebo nemůže cyklus proběhnout alespoň jednou!

Nepoužívejte řídící proměnnou cyklu jako pomocnou proměnnou pro výpočty!

Pokud cyklus provádí jen jeden příkaz, nemusíme používat složené závorky!

9.2. Příkazy break a continue

Doplníme a vysvětlíme příkazy break a continue, které ovlivňují provádění všech cyklů:

break – ukončuje (nejvnitřnější) smyčku, okamžitě opouští cyklus

continue – skočí na konec (nejvnitřnější) smyčky a tím si vynutí nové vyhodnocení podmínky cyklu. Neopouští cyklus!

9.3. Cyklus for

Cyklus ***for*** je zvláštním případem cyklu se vstupní podmínkou – nejprve se testuje podmínka a pokud je splněna, cyklus se provede. U cyklu ***for*** musíme znát počet průchodů cyklem! Zápis cyklu for lze rozdělit do tří částí vzájemně oddělených středníky for (i=1;i<=5;i++). První část je inicializační (i=1;) a proběhne na začátku

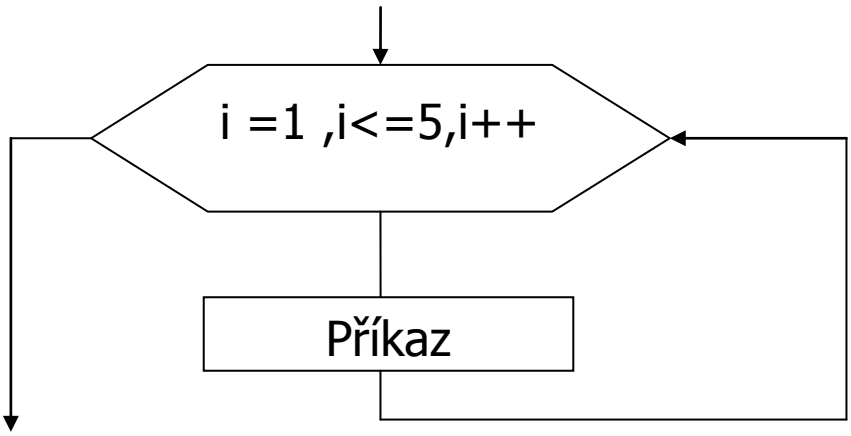
cyklu. Následuje druhá část – podmínka proveditelnosti cyklu (i<=5;), která se testuje před každým opakováním těla cyklu. Třetí část tvoří ošetření řídící proměnné cyklu (i++), které se provede jako poslední příkaz těla cyklu. Výhoda cyklu for je v tom, že si tuto úpravu řídící proměnné provede sám.

Obecný zápis:

for (výraz-začátek;výraz-konec;krok)

příkaz;

Vývojový diagram cyklu **for**:



počáteční nastavení řídící proměnné cyklu — *podmínka pro opakování cyklu* — *ošetření řídící proměnné cyklu*

for (i=1;i<=5;i++) …***i je řídící proměnná cyklu for***

Při provádění tohoto cyklu se nejprve nastaví řídící proměnná cyklu na hodnotu 1 (i=1), pak se otestuje podmínka proveditelnosti cyklu (i<=5) a pokud je splněna,

provedou se příkazy v těle cyklu (např. cout<<i<<" ";). Jako poslední krok cyklu se provede úprava řídící proměnné cyklu podle zápisu cyklu (i++). Pro snadnější pochopení cyklu for znázorníme jeho postupné provádění:

```
for (i=1;i<=5;i++)
        cout<<"cislo je "<<i<<"\n ";
```

Výpis programu:

cislo je 1
cislo je 2
cislo je 3
cislo je 4
cislo je 5

Pomocí popisu průběhu programu si vysvětlíme průběh cyklu for:

pro i=1 , 1 <=5 … *ano, podmínka je splněna*
cislo je 1 … *výpis na obrazovku*
i=i+1=1+1=2 … *úprava řídící proměnné*
pro i=2 , 2 <=5 … *ano, podmínka je splněna*
cislo je 2
i=i+1=2+1=3 … *úprava řídící proměnné*
pro i=3 , 3 <=5 … *ano, podmínka je splněna*
cislo je 3
i=i+1=3+1=4 … *úprava řídící proměnné*
pro i=4 , 4 <=5 … *ano, podmínka je splněna*
cislo je 4
i=i+1=4+1=5 … *úprava řídící proměnné*
pro i=5 , 5<=5 … *ano, podmínka je splněna*
cislo je 5
i=i+1=5+1=6 … *úprava řídící proměnné*
pro i=6 , 6 <=5 … *podmínka není splněna,*
cyklus končí a v i je hodnota 6

Př.:

```
for (i=0;i<5;i++)
    x++;
```

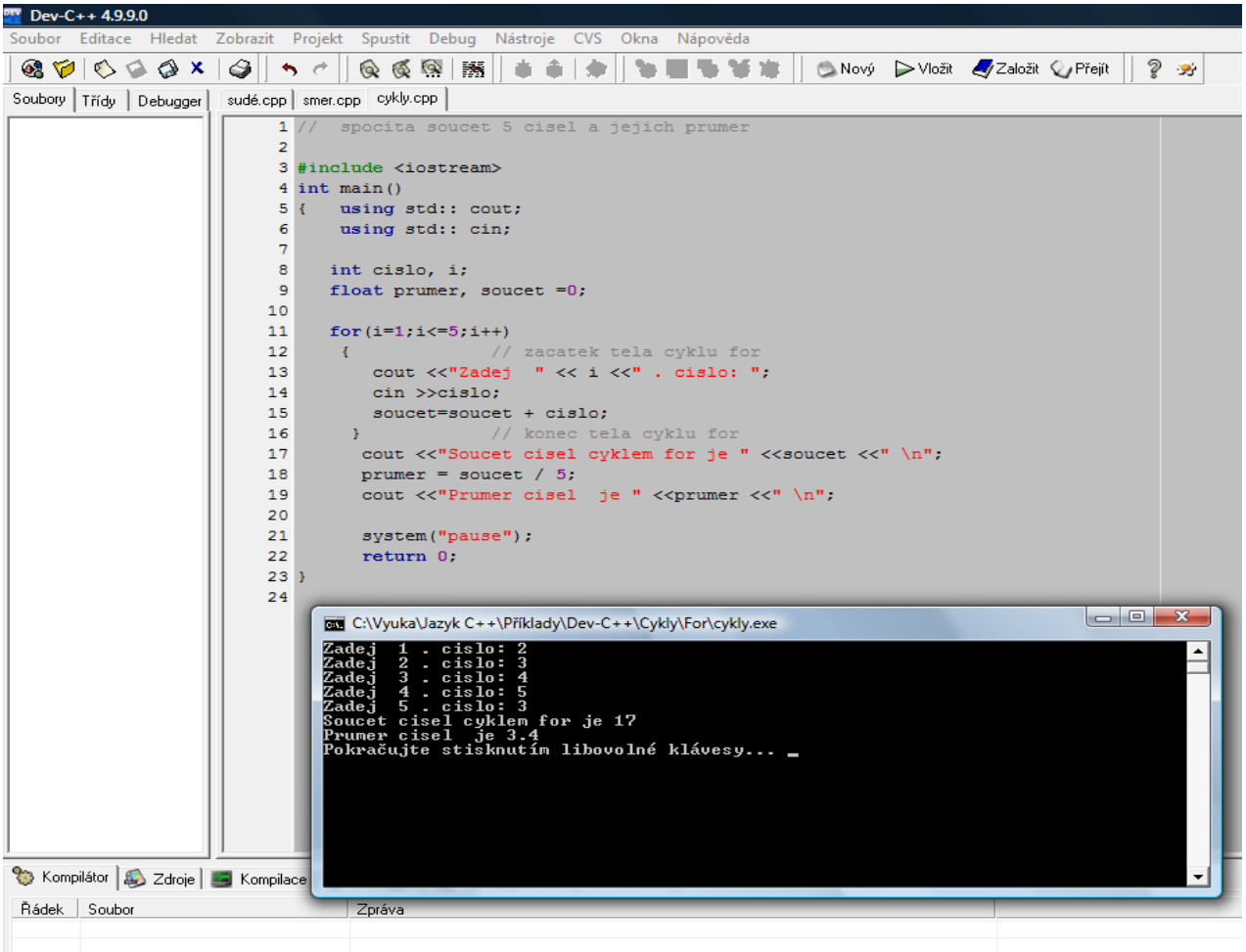
nebo

```
for (i=1;i<=5;i++)
    x++;
```

nebo

```
for (i=5;i>0;i--)
    x++;
```

Použití v příkladu:



Parametry cyklu for: **výraz-začátek;výraz-konec;krok** spolu nemusí souviset a dokonce nemusí být uvedeny, ale je nutné uvést středník, kterým je chybějící výraz oddělen!

Např. Mohou dokonce chybět všechny parametry a tím vytvoříme nekonečný cyklus, který ale odpovídá pravidlům zápisu příkazu v C++ a překladač jej neoznačí jako chybný:

```
for ( ; ; )
```

Upozornění: **Pozor na zápis středníku za podmínku cyklu - vede k ukončení příkazu for a tím se oddělí tělo cyklu a stane se samostatným příkazem**

```
for (i=1;i<=5;i++);                    ...cyklus proběhne 5x
        cout<<"cislo je "<<i<<"\n ";    ... pak se vypíše text
```

Výpis programu:

cislo je 5

Často se při zápisu cyklu for využívá i operátor čárka – pozor na vhodné použití!

Např.

```
for (i= 1, sum=0; i<=10; i++)
```

provede se na začátku cyklu

provede se automaticky jako poslední příkaz cyklu

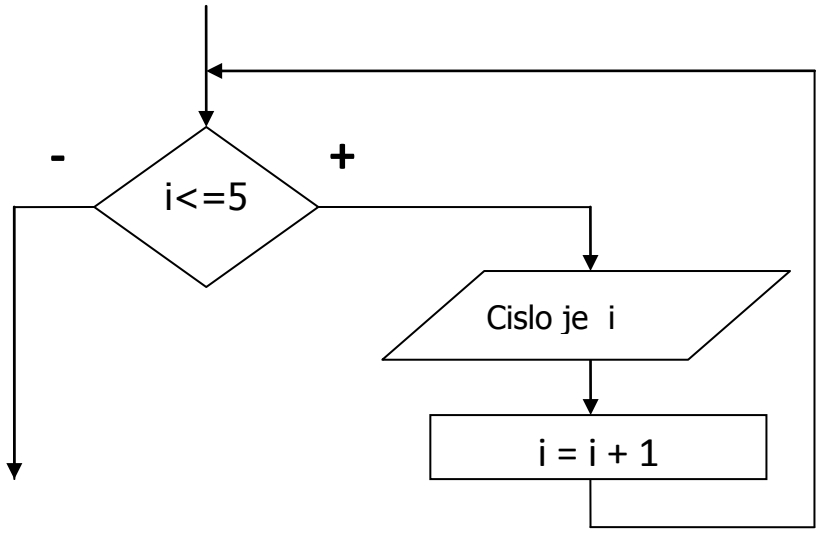
9.2. Cyklus while

Cyklus **while** je cyklem se vstupní podmínkou: cyklus while napřed testuje podmínku provádění cyklu, teprve pokud je splněna, postupně vykoná příkazy těla cyklu – **pokud není podmínka splněna, příkazy v těle cyklu nemusí proběhnout ani jednou!**

Obecný zápis:

```
while (podmínka)
{
        příkaz1;
        příkaz2;
}
```

Vývojový diagram cyklu **while**:



Př.:

```
i=1;
while (i<=5)
    {
      cout<<"cislo je "<<i<<"\n ";
      i++;
     }
```

Výpis programu:

cislo je 1
cislo je 2
cislo je 3
cislo je 4
cislo je 5

Cyklus while je stejně jako cyklus for cyklem se vstupní podmínkou. Má však jisté odlišnosti:

- **řídící proměnná cyklu (v tomto příkladu i) se musí nastavit před cyklem**
- **v těle cyklu musíme upravit obsah řídící proměnné cyklu (i++)**

Upozornění: **Pozor na zápis středníku za podmínku cyklu - vede k ukončení příkazu while a tím k vytvoření nekonečného cyklu**

while (i<=5);

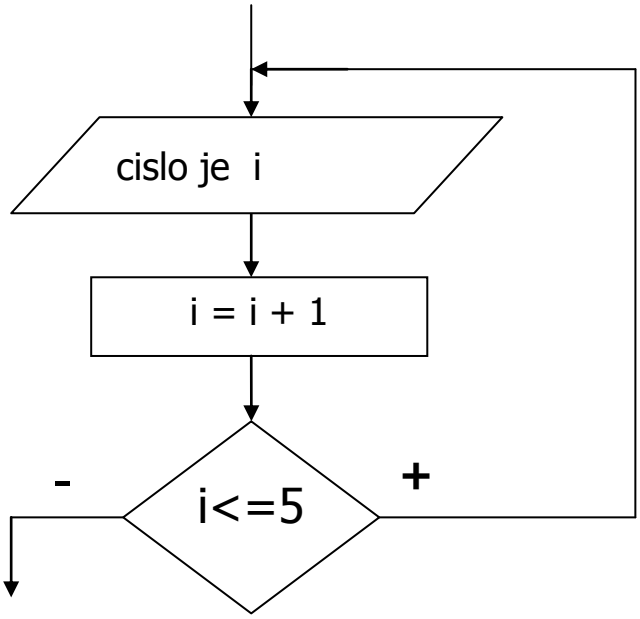
9.3. Cyklus Do-while

Cyklus **do-while** testuje podmínku pro opakování cyklu po průchodu cyklem – **cyklus vždy proběhne alespoň jednou!**

Zápis:

```
do {
    příkaz1;
    příkaz2;
} while (podmínka) ;
```

Vývojový diagram cyklu **do-while**:



Př.: ***i=1;***

```
do
  {
    cout<<"cislo je "<<i<<"\n ";
    i++;
  } while (i<=5) ;
```

Výpis programu:

cislo je 1
cislo je 2
cislo je 3
cislo je 4
cislo je 5

Cyklus do-while je cyklem s výstupní podmínkou. Má své charakteristické vlastnosti:

- **příkazy v těle cyklu se provedou vždy alespoň jednou, dojde tedy ke změně obsahu sumátorů a jiných proměnných**
- **v těle cyklu musíme upravit obsah řídící proměnné cyklu (i++)**

Upozornění: Za příkazem **do** se nepíše středník – celý příkaz končí zápisem podmínky pro ukončení cyklu. Středník píšeme až za ni.

10. Pole

10.1. Jednorozměrné pole

Doposud jsme pracovali s jednoduchými proměnnými – v jedné proměnné byla uložena jedna hodnota. V praxi se při řešení příkladů setkáme s požadavkem uložit větší počet hodnot stejného typu – naměřené průměrné denní teploty (365 desetinných čísel), počet zameškaných hodin za jeden pracovní týden (5 celých čísel) apod. Tento typ hodnot budeme ukládat do proměnné typu pole. Každá hodnota se ukládá do samostatného prvku pole a všechny tyto prvky jsou uloženy v paměti za sebou.

Ukázka pole:

23	11	64	48	15

Pole představuje jednu proměnnou, která obsahuje určitý počet hodnot stejného typu – označujeme ji jako strukturovanou proměnnou s homogenními prvky. Abychom se dostali k jednotlivým prvkům pole, musíme použít název pole a ***index pole***, což je pořadové číslo prvku pole. Jednotlivé prvky pole můžeme považovat za jednoduché proměnné.

Definice:

Pole je strukturovaný datový typ, který se skládá z prvků stejného typu. K jednotlivým prvkům pole přistupujeme pomocí indexů. V C++ je pole vždy indexováno od 0 !

Deklarace:

typ název [počet] ;

(počet – hodnotu musíme znát už při překladu programu)

Př.:

```
int   pole [ 5 ];
```

pole					
	23	11	-56	108	-5
index	*0*	*1*	*2*	*3*	*4*

Do proměnné ***pole*** v uvedeném příkladu můžeme uložit 5 hodnot; první bude mít index 0 (pole [0]), poslední bude mít index 4 (pole [4]) !

Deklarace a inicializace pole

Zvláštním případem deklarace je tzv. deklarace s inicializací, kdy v jednom kroku pole deklarujeme (zavádíme jako proměnnou) a současně ho plníme uvedenými hodnotami:

int pole [3] = {1,2,3} … jedním krokem deklarujeme pole v délce 3 prvků a rovnou ho plníme čísly 1 2 3

int pole [ ] = {102,9,11,-3,58} … deklarujeme pole a uvedením jeho hodnot definujeme i délku pole, kterou si překladač spočítá podle počtu uvedených hodnot

Přístup k prvkům pole:

pole [0] = 23 ;

pole [1] = 11 ;

pole[2] = -56 ;

pole[3] = 108 ;

pole[4] = -5 ;

Jazyk C++ nekontroluje dolní a horní meze polí! Pokud by v programu mohla nastat situace podtečení nebo přetečení mezí pole, je nutné, abychom to sami ošetřili v programu!

Typ pole závisí na typu hodnot, které chceme do pole ukládat. Budeme používat již uvedené typy u jednoduchých datových typů : int, float a char.

Velikost pole souvisí s počtem zpracovaných hodnot. Není omezen žádnou konstantou, snad jen velikostí paměti.

Pro práci s polem se nejčastěji používá cyklus for.

Vynulování pole:

```
for ( i=0; i<5; i++)
        pole[ i ] = 0;
```

Načtení čísel do pole:

```
for ( i=0; i<5; i++)
{
        cout<<"Zadej "<<i+1<<". prvek";
       cin>>pole[i];
}
```

Výpis pole:

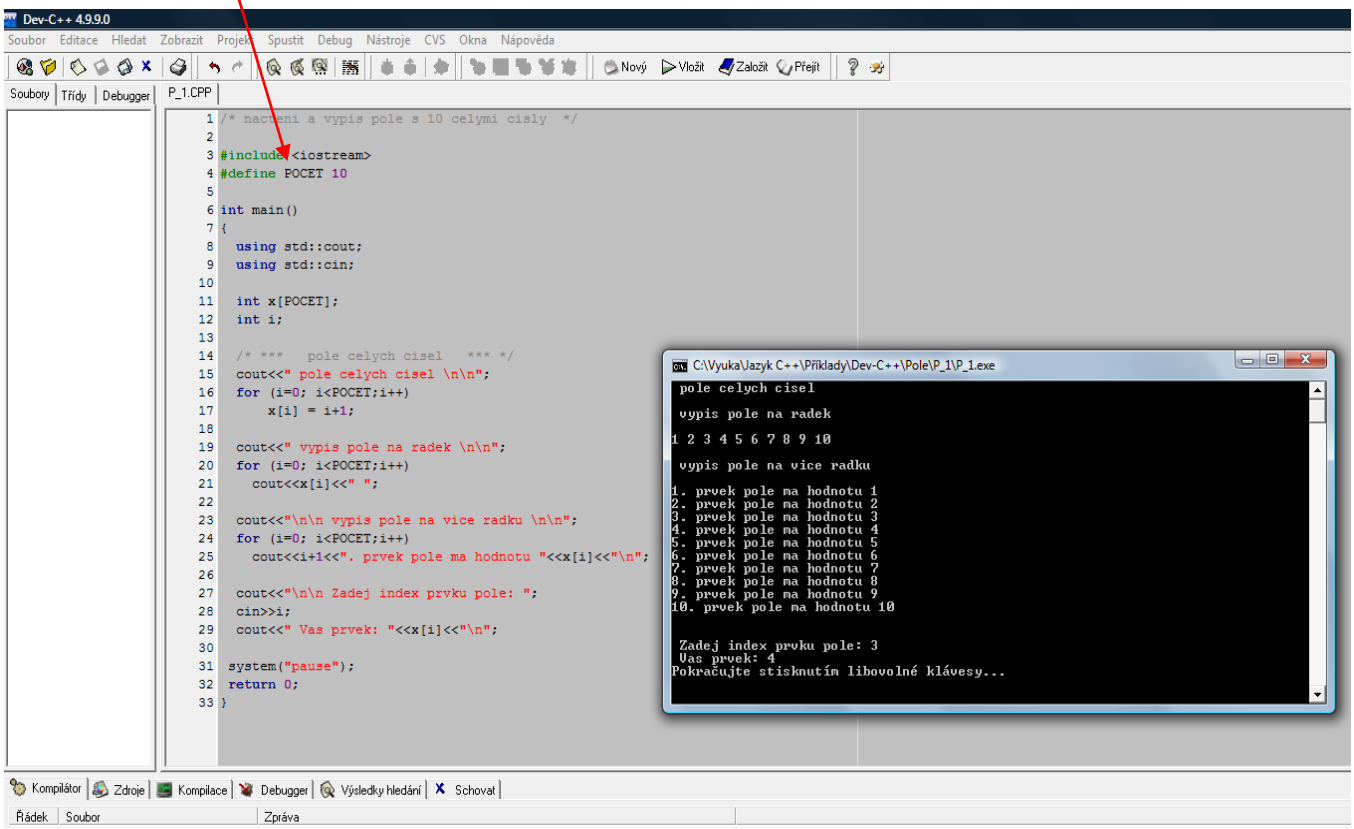
```
cout<<" Vase pole: ";
for ( i=0; i<5; i++)
{
       cout<< pole[i]<<"  ";
```

```
}
```

Při práci s polem je vhodné zavést konstantu pro velikost pole a tuto konstantu používat ve všech cyklech. Změnou hodnoty konstanty pak snadno upravíme program tak, aby místo 5 hodnot pracoval s 15, 30, 50 … hodnotami, aniž bychom museli dělat další úpravy v programu. Použijeme k tomu příkaz:

#define POCET 10

Použití v příkladu:



10.2. Generátor čísel

Pokud potřebujeme pole naplnit čísly a nemusíme je zadávat ručně, můžeme použít generátor náhodných čísel. Podle instalace programu je někdy nutné připojit knihovny ***<time.h>*** a ***<stdlib.h>***, které obsahují příkazy pro nastartování generátoru náhodných čísel i příkazy pro samotné generování.

V těle programu uvedeme příkaz pro inicializování (nastartování) generátoru:

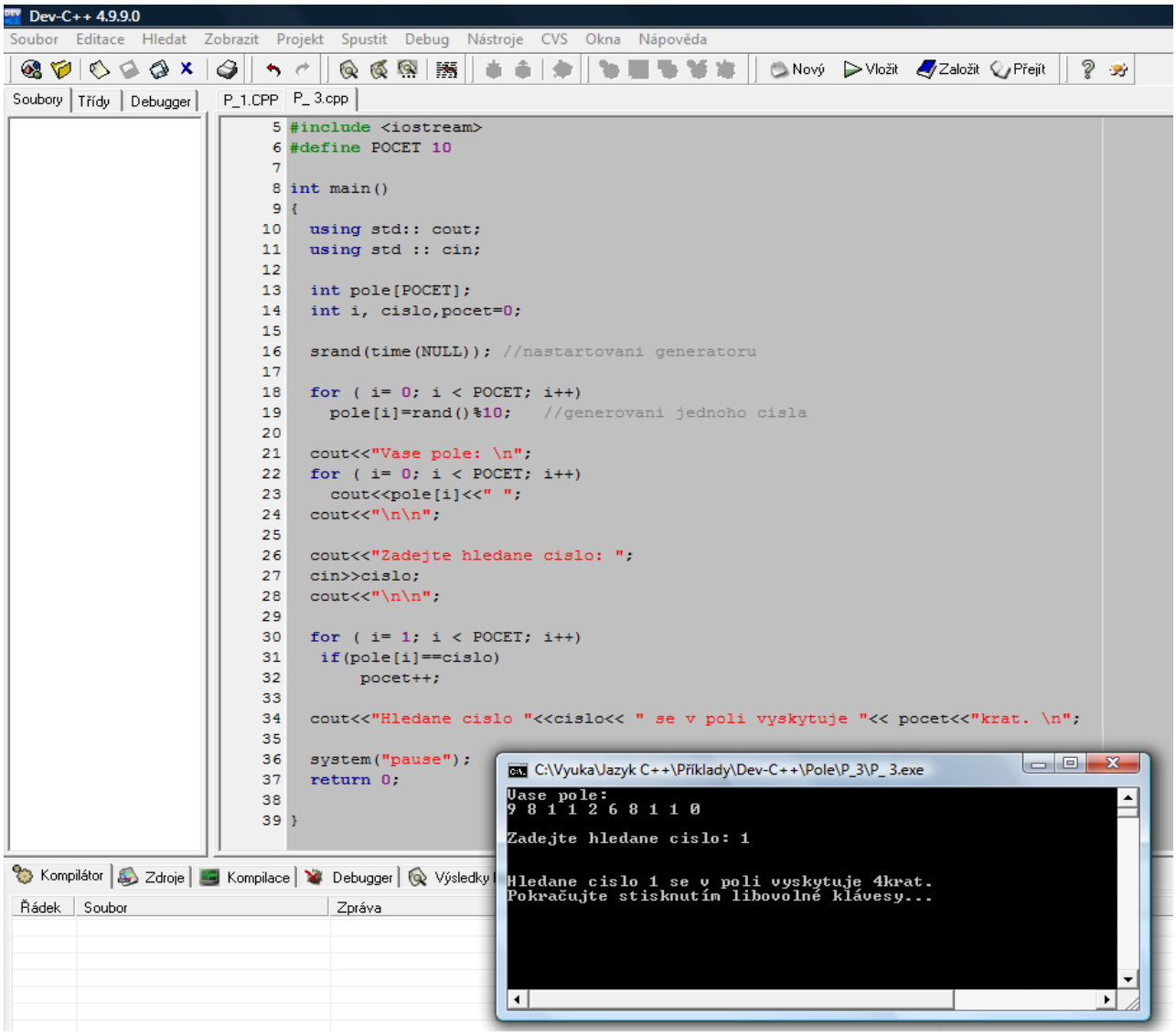
srand(time(NULL));

tento příkaz uvedeme v programu jen jednou, nejčastěji ho zapíšeme hned za deklaraci proměnných.

Ke generování čísel do pole použijeme příkaz ***rand***:

pole[i] = rand()%100; ... generuje čísla od 0 do 99 ...<0,100)

Použití v příkladu:



10.3. Třídění prvků v poli

Při řešení úloh se často setkáme s požadavkem seřadit prvky pole do určitého pořadí. Třídění (řazení) prvků v poli znamená jejich uspořádání vzestupně nebo sestupně. Existuje několik řadících algoritmů. Liší se postupem třídění prvků.

Bubble Sort

Je třídící metoda založená na porovnávání sousedních prvků. Pokud prvky vyhovují podmínce, porovnávají se další prvky. Jestliže prvky podmínce nevyhovují, dojde k jejich prohození (výměně pozice v poli).

Při postupném porovnání všech prvků pole dojde k zatřídění jednoho prvku. Pro uspořádání všech prvků v ***jednorozměrném poli***, musíme použít ***dva cykly*** :

```
for ( i=0; i<n-1; i++)
        for ( j=0; j<n-1; j++)
            if (pole[j] > pole[j+1] )
              {
                 pom=pole[j];
                 pole[j]=pole[j+1];
                 pole[j+1]=pom;
              }
```

Máme-li 5-ti prvkové pole, pak pokud zatřídíme 4 prvky pole, 5-tý prvek je zatříděn automaticky!

Upravený – rychlejší zápis Bubble Sortu – neporovnávají se znovu prvky, které už byly zatříděny:

```
for ( i=n; i>1; i--)
        for ( j=0; j<i-1; j++)
                if (pole[j] > pole[j+1] )
                   {
                      pom=pole[j];
                      pole[j]=pole[j+1];
                      pole[j+1]=pom;
                   }
```

Nejrychlejší zápis Bubble Sortu – obsahuje příznakovou proměnnou pro ukončení třídění, pokud nedošlo k prohození prvků!

```
do  {
          konec=0;
         for ( j=0; j<n-1; j++)
                 if (pole[j] > pole[j+1])
                  {
                    konec=1;
                    pom=pole[j];
                    pole[j]=pole[j+1];
                    pole[j+1]=pom;
                  }
      } while (konec==1);
```

10.4. Dvourozměrné pole

Při práci můžeme kromě jednorozměrného pole použít také vícerozměrné pole.
S každým dalším rozměrem souvisí i požadavek na další index (dvourozměrné pole = 2 indexy, trojrozměrné pole = 3 indexy atd.). Rozměr pole není omezen, snad pouze velikostí paměti. Problém nastává při zobrazení vícerozměrných polí. Dvourozměrné pole můžeme vypsat jako matici hodnot uspořádaných do řádků a sloupců (rovina). U trojrozměrného pole (můžeme si představit jako Rubikovu kostku) musíme pole

vypisovat po jednotlivých rovinách – maticích hodnot. Tím se trochu ztrácí efekt třetího rozměru (prostor). V našich příkladech nám postačí dvourozměrné pole.

Definice:

Dvourozměrné pole si můžeme představit jako matici prvků stejného typu. K jednotlivým prvkům matice přistupujeme pomocí indexu řádku a indexu sloupce.

Deklarace:

typ mat [řádky] [sloupce]

rozměry matice musíme znát už při překladu

Př.:

int mat [2] [3];

mat = *sloupec* *0* *1* *2*

řádek

0	12	23	-5
1	62	37	105

Přístup k prvkům matice:

mat [0] [0] = 12 ; *mat [1] [0] = 62 ;*

mat [0] [1] = 23 ; *mat [1] [1] = 37 ;*

mat [0] [2] = -5 ; *mat [1] [2] = 105 ;*

Načtení čísel do matice:

```
for ( i=0; i<2; i++)
      for ( j=0; j<3; j++)
        {
           cout<<"\nZadej prvek["<<i<<","<<j<<"]:";
          cin>>mat[i] [j];
        }
```

Výpis matice:

```
      cout<<"Vaše matice: \n";
      for ( i=0; i<5; i++)
        {
           for ( j=0; j<3; j++)
              cout<<mat[i] [j];
          cout<<"\n";
        }
```