Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

Program vypíše:

```
a=Text 1
b=
c=Text 1
Pokračujte stisknutím libovolné klávesy...
a=Text a
b=Text 1
c=Text c
Pokračujte stisknutím libovolné klávesy...
a=a: Text a
b=b: Text 1
c=c: c
Pokračujte stisknutím libovolné klávesy...
a=a: Text ab: Text 1c: c
b=b: Text 1
c=c: c
Pokračujte stisknutím libovolné klávesy...
a[1]=;
a=a; Text ab: Text 1c: c
Pokračujte stisknutím libovolné klávesy...
str=a; Text ab: Text 1c: c
Pokračujte stisknutím libovolné klávesy...
Zadej retezec: C++
d=C++
Pokračujte stisknutím libovolné klávesy...
```

7.3 Mělká a hluboká kopie

Pochopit správně rozdíly mezi mělkou a hlubokou kopií je velmi důležité, provedeme tedy ještě toto shrnutí.

Mělká kopie znamená, že se instance kopíruje po složkách. Čili atributy tak jak jsou, se ze zdrojové instance zkopírují do cílové instance. Takové chování je v pořádku, pokud jsou v atributech jednoduché datové typy. Dokonce se korektně zkopíruje i pole (připomeňme, že normálně není kopie pole možná a musí být provedena po jednotlivých prvcích). Velké problémy nastanou, pokud je některý z atributů ukazatel. Adresa uložená v ukazateli se zkopíruje a oba atributy v různých instancích ukazují na stejná data. Pokud se změní jedna instance, promítnou se změny i do druhé. Při destrukci jedné instance začne ukazatel ve druhé instanci odkazovat na již neexisující data. Bohužel, standardní verze operátoru přiřazení a kopírovacího konstruktoru provádí právě mělkou kopii.

Hluboká kopie zohledňuje přítomnost ukazatelů tak, že v cílové instanci vytváří stejná dynamická data znovu a získanou adresu ukládá do příslušného ukazatele. Mezi zdrojovou a cílovou instancí neexistuje žádná vazba. Každá instance pracuje se svou kopií dat. Ovšem hlubokou kopii musíme sami předepsat tím, že implementujeme vlastní verze operátoru přiřazení a kopírovacího konstruktoru.

7.4 Další možná rozšíření třídy TString

Regulérní řetězcová třída obvykle obsahuje další operace s řetězci. Mezi obvyklé operace patří zejména: stanovení délky řetězce, vkládání podřetězce, mazání části řetězce, kopírování podřetězce.

Takové operace není možno zavést ve formě operátorů, ale je nutno definovat nové metody.

V následujícím textu se seznámíme s algoritmy, které tyto operace realizují.

Metoda Length – stanovení délky řetězce

Tato metoda nejdříve vynuluje počítadlo.

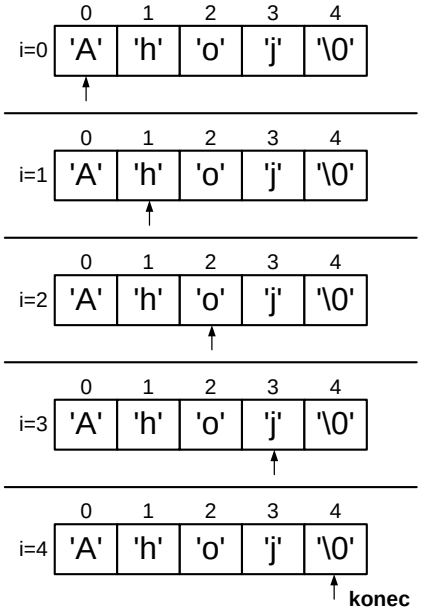
Potom prochází řetězcem tak dlouho, dokud nenalezne zarážku.

Za každý průchod si zvýší počítadlo o 1. Počet znaků před zarážkou (což je vlastně délka řetězce) je pak uložen v tomto počítadle.

Počítadlo je možno současně použít jako index právě procházeného prvku uvnitř řetězce.

Tato metoda patří mezi nejjednodušší řetězcové operace. Z hlediska standardní řetězcové knihovny ji zajišťuje funkce **strlen**.

Při realizaci této metody nemusíme testovat žádné chybové stavy, to je v podstatě dáno i tím, že metoda nemá vstupní parametry!



Obr. 7.1. Naznačení průchodu řetězcem pro stanovení jeho délky

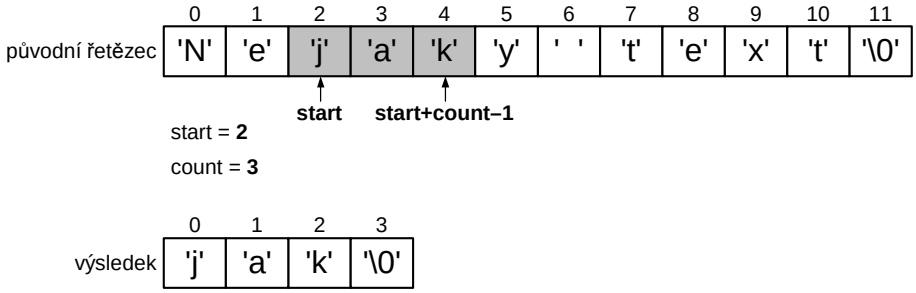
Doporučená hlavička:

```
unsigned Length( ) const;
```

Metoda SubStr – získání podřetězce

Parametry této metody jsou startovací pozice (**start**) a počet kopírovaných znaků (**count**), které definují podřetězec. Výsledkem je podřetězec zkopírovaný z původního řetězce.

Nejdříve se musí vytvořit nová instance řetězce, do kterého se podřetězec nakopíruje. Celá operace je pak proveditelná cyklem **for**, který v původním řetězci používá index od **start** do **start+count–1** a v cílovém řetězci se pak jedná o indexy **0** až **count–1**. Na pozici **count** ve výsledku musí být umístěna zarážka.



Obr. 7.2. Vysvětlení úlohy parametrů start a count, odpovídající podřetězec

Podmínky úspěšného provedení operace:

- cílový řetězec musí mít fyzickou délku vyšší než je dáno parametrem **count** (lze vyřešit při jeho vytváření, je to plně v režii této metody),
- indexy **start** i **start+count–1** musí padnout do rozsahu platných indexů řetězce (musí být nižší než délka řetězce),
- **count** musí být nezáporné číslo (je-li count = 0, je výsledkem prázdný řetězec).

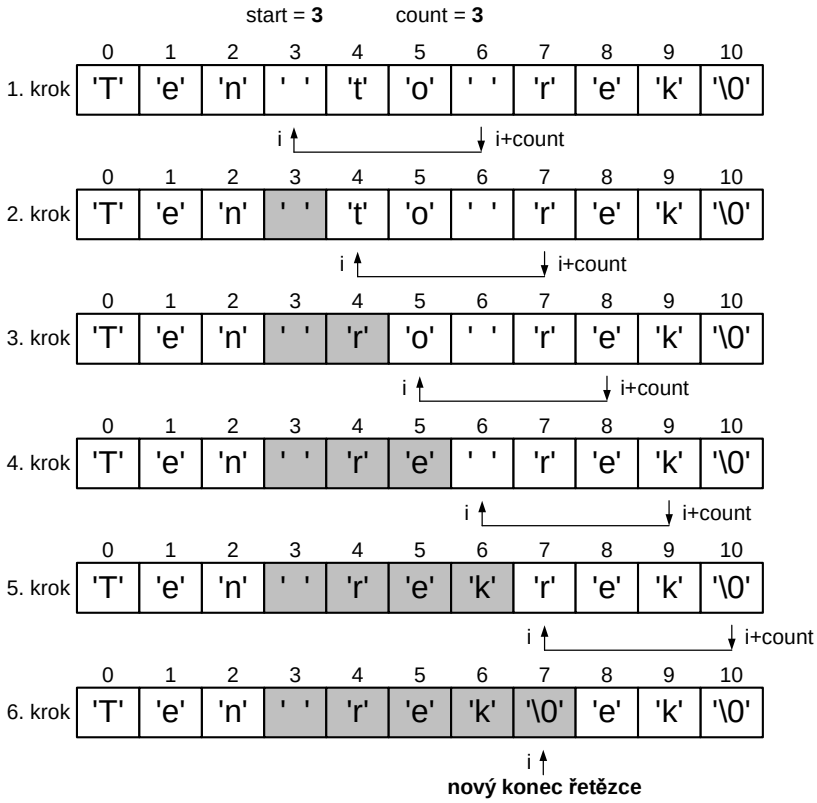
Doporučená hlavička:

```
TString SubStr(int start, int count) const;
```

Metoda Delete – rušení podřetězce

Parametry této metody jsou obvykle startovací pozice (**start**) a počet rušených znaků (**count**).

Celá operace je pak proveditelná cyklem, který startuje od indexu **start** a prochází řetězcem až do jeho konce (včetně). Pokud použijeme řídicí index zavedený jako **i**, bude se z pravé části řetězce (pozice **i+count**) kopírovat jeden znak do levé části řetězce (pozice **i**). Index se bude zvyšovat až do nalezení zarážky. I ta se však ještě zkopíruje z pravé části nalevo.



Obr. 7.3. Naznačení operace rušení podřetězce

Podmínky úspěšného provedení operace:

- indexy **start** i **start+count–1** musí padnout do rozsahu platných indexů řetězce (musí být nižší než délka řetězce),
- **count** musí být nezáporné číslo (je-li count = 0, nezruší se žádný znak).

Doporučená hlavička:

```
void Delete(int start, int count);
```

Metoda Insert – vložení podřetězce

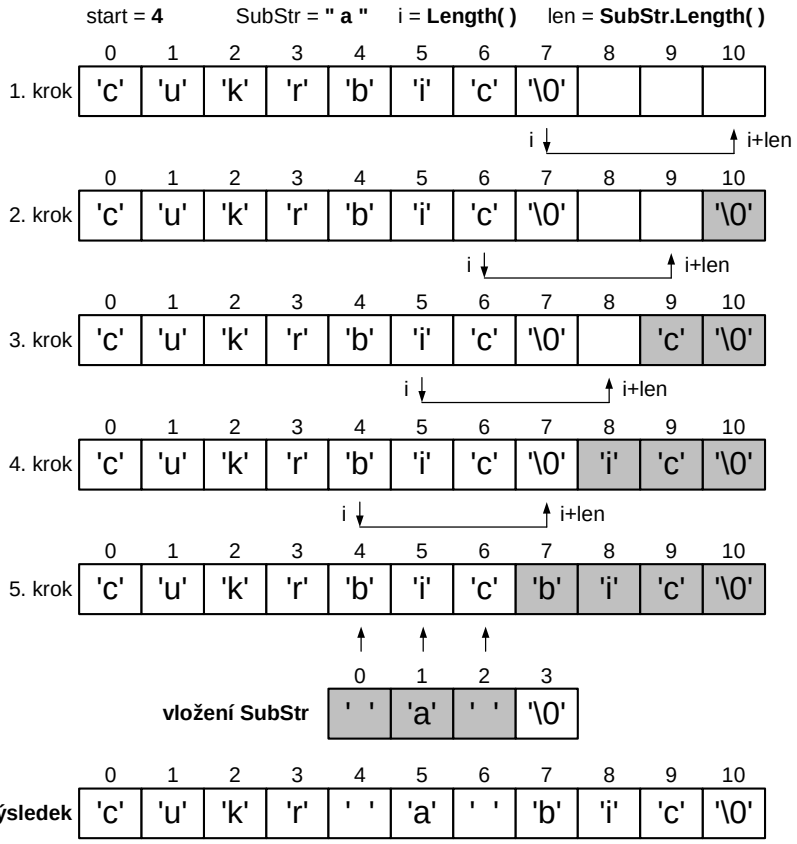
Parametry této metody jsou obvykle startovací pozice (**start**) a vkládaný řetězec (**SubStr**).

Operaci musíme rozdělit na dvě části:

- Nejdříve vytvoříme místo pro vkládaný řetězec. Takže znaky počínaje indexem **start** musí být odsunuty směrem ke konci řetězce o tolik pozic, kolik udává délka vkládaného řetězce. Znaky je třeba kopírovat od konce a to

včetně zarážky. Uložíme-li si do proměnné **i** délku původního řetězce a do proměnné **len** délku vkládaného podřetězce, budeme vždy kopírovat jeden znak zleva (index **i**) napravo (index **i+len**). Poslední kopie proběhne pro prvek s indexem **start**.

- Druhou částí operace je pak kopírování znaků z řetězce **SubStr** (indexy **0** až **len–1**) do cílového řetězce (indexy **start** až **start+len–1**).



Obr. 7.4. Naznačení operace rušení podřetězce

Podmínky úspěšného provedení operace:

- cílový řetězec musí mít fyzickou délku vyšší než je dáno součtem délek obou řetězců,
- index **start** musí padnout do rozsahu platných indexů cílového řetězce (musí být nižší nebo roven délce řetězce, pokud je start roven délce řetězce, přidá se vkládaný řetězec na konec).

Doporučená hlavička:

```
void Insert(int start, const TString& SubStr);
```

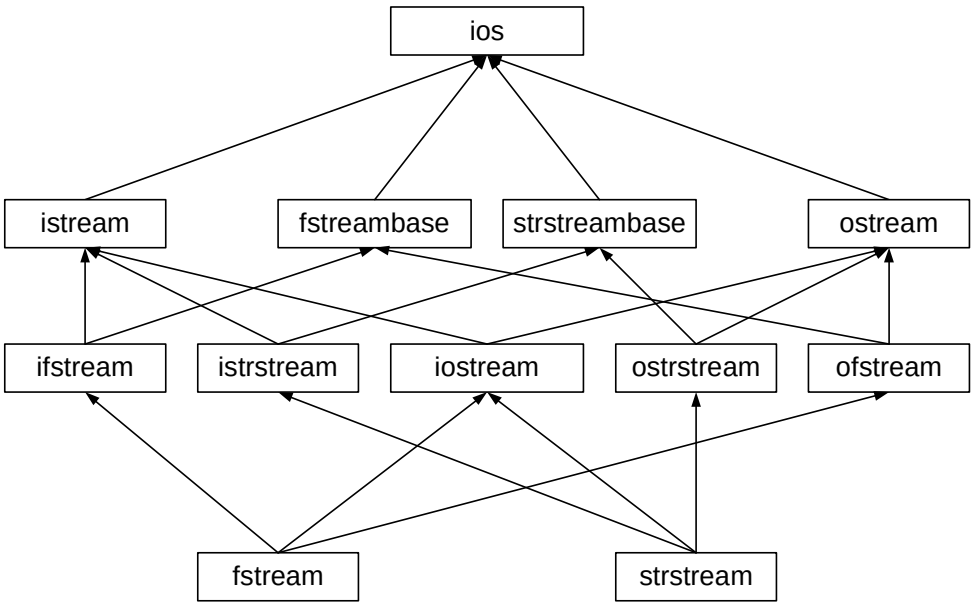
8 Proudová knihovna

Základní informace o proudech byly uvedeny v kapitole 1.3. Proudům se budeme nyní věnovat podrobněji.

8.1 Hierarchie proudů

Na obr. 8.1 je uvedena hierarchie proudových tříd. Na vrcholu stojí třída **ios**, přímými následníky jsou třídy **iostream** (vstupní proud) a **ostream** (výstupní proud).

Z těchto tříd jsou pak posléze odvozeny souborové a řetězcové proudy.



Obr. 8.1. Hierarchie tříd proudové knihovny

8.2 Standardně zavedené proudy

Prostor jmen **std** zavádí tři standardní instance proudů, viz tab 8.1.

Tab. 8.1 Standardně definované proudy

Proud	Směr	Význam
cout	výstup	standardní výstupní proud
cin	vstup	standardní vstupní proud
cerr	výstup	standardní výstup chybových zpráv

8.3 Základní třídy

Připomeňme krátce význam základních tříd hierarchie.

Tyto základní proudové třídy jsou definovány v hlavičkovém souboru **iostream**.

Výstup – ostream

Třída **ostream** je definována tak, aby operátor **<<** zajistil výstup *vestavěných typů*. Umožňuje tedy výstup datových typů: **char***, **char**, **short**, **int**, **long**, **double**, **void***. Třída je přímým následníkem **ios**.

Vstup – istream

Třída **istream** je definována tak, aby operátor **>>** zajistil vstup vestavěných typů. Třída je přímým následníkem **ios**.

8.4 Souborové proudy

C++ definuje tři souborové proudy, viz tab. 8.2. Souborové proudy jsou definovány v hlavičkovém souboru **fstream**.

Tab. 8.2 Souborové proudy

Třída	Význam
ifstream	proud pro vstupní soubor
ofstream	proud pro výstupní soubor
fstream	proud pro čtení i zápis do souboru

Otevření souboru

Každý soubor, se kterým chceme pracovat, je třeba nejdříve otevřít pomocí metody **open**. Hlavička:

```
void open(
  const char* filename,
  int openmode,
  int protect=filebuf::openprot
);
```

jméno souboru

způsob otevření

ochrana souboru

*Tab. 8.3 Režimy otevření proudu definované v **ios** (bitové pole)*

Režim	Význam
ios::in	otevři pro čtení
ios::out	otevři pro zápis
ios::ate	otevři a přesuň se na konec souboru
ios::app	přidej na konec (append)
ios::trunc	zkrať délku souboru na 0 (přepis)
ios::binary	otevři jako binární (ne textový) soubor

Zavření souboru – close

Soubor zavíráme metodou **close**, hlavička:

```
void close();
```

Test úspěšnosti poslední operace

Úspěšnost poslední operace s proudem lze testovat tak, že proud použijeme v logickém výrazu. Proběhla-li poslední operace korektně, chápe se stav proudu jako pravdivý (**true**). Pokud došlo k chybě (například neúspěšné otevření souboru nebo dosažení konce souboru), chápe se stav proudu jako nepravdivý (**false**).

8.4.1 PROG_08-01 – zápis do textového souboru pomocí proudu

Zadání: Napište program, který do souboru **DATA.TXT** zapíše 1000 náhodně vygenerovaných celých čísel v rozsahu 1 až 100. Každé číslo bude zapsáno na zvláštním řádku.

Nejdříve deklarujeme proměnnou typu **ofstream** s identifikátorem **vystup**.

Tento proud metodou **open** napojíme na soubor s názvem **DATA.TXT**.

Úspěšné otevření ověříme tak, že proměnnou **vystup** použijeme v podmíněném příkazu. Pokud se otevření souboru podařilo, je výraz brán jako pravdivý a provedou se další příkazy.

Vlastní ukládání čísel je řešeno klasicky, operátorem **<<**.

Nakonec musíme zavřít soubor pomocí metody **close**.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <stdlib.h>
using namespace std;

int main()
{
  ofstream vystup;
  vystup.open("data.txt",ios::out);

  if(vystup)
  {
    for(int i=1;i<=1000;i++)
      vystup<<1+rand()%100<<endl;
    vystup.close();

    cout<<"Data zapsana"<<endl;
  }
  else
    cout<<"Soubor nelze otevrit"<<endl;
}
```

výstupní proud

otevření souboru

test úspěšnosti otevření

uložení čísla

zavření souboru

8.4.2 PROG_08-02 – čtení z textového souboru pomocí proudu

Zadání: Napište program, který přečte celá čísla uložená v souboru **DATA.TXT** a vypočítá jejich součet a průměr.

Pro tento příklad musíme deklarovat jednak instanci třídy **ifstream** (označíme ji identifikátorem **vstup**) a dále proměnnou na načtení jednoho čísla (**i**), proměnnou na výpočet součtu (označíme ji **soucet** a vynulujeme) a proměnnou na počítání čísel (označíme ji **pocet** a vynulujeme ji).

Následně otevřeme soubor pomocí metody **open**.

Kontrola úspěšného otevření je opět provedena testováním stavu proudu v podmíněném příkaze.

Pokud se otevření souboru podaří, je spuštěn cyklus **while**. Ten nejdříve přečte jedno číslo ze souboru a okamžitě zkontroluje, zda bylo čtení úspěšné.

Úspěšně přečtené číslo zvýší proměnnou **pocet** o 1 a jeho hodnota je přičtena do součtu čísel představovaných obsahem proměnné **soucet**.

Po přečtení všech čísel zavřeme soubor metodou **close**.

Pokud byl počet čísel nenulový (a tedy lze provést výpočet průměru), zobrazí se jednak součet čísel, jejich počet a dále průměr všech čísel (vzpomeňte si, že pro vyvolání reálného podílu je nutné celočíselné hodnoty přetypovat na typ **double**).