Vstupy a výstupy v jazyce C++

- jazyk C++ dává možnost řešit vstup a výstup proměnných (na V/V zařízení) podstatně elegantněji než jazyk C. Tyto mechanizmy se postupně vyvíjejí, v poslední době využívají vlastnosti objektů, šablon, dědění i přetěžování funkcí. Existují společné vlastnosti operací s proměnnou a V/V, které jsou specializované pro standardní typy.
- přetížení (globálních) operátorů << a >>
- typově orientovány
- knihovní funkce (ne klíčová slova) <iostream> (ios pro char; wios pro wchar_t, …)
- vstup a výstup je zajišťován přes objekty, hierarchie tříd
- stream je abstrakce I/O zařízení, které je ”napojeno” na soubor, paměť, standardní (seriové) zařízení
- standardní objekty pro vstupy a výstupy (streamy) spojené s konzolami – cin, cout, cerr, clog pro char a wcin, wcout, wcerr, wclog pro wchar_t
- knihovny xxxstream (http://www.cplusplus.com/reference/iolibrary/)

<iostream> standardní vstup/výstup
<fstream> streamy pro soubory
<sstream> streamy pro řetězce (paměť)
<ios> definuje základní třídy a konstanty
<streambuf> definuje buffery (přístup na zařízení potom není přímý pro každou operaci)
<iomanip> standardní manipulátory

práce se streamy

- vstup a výstup základních typů přes konzolu
- formátování základních typů
- práce se soubory
- implementace ve třídách
- obecná realizace streamu

vstup a výstup přes konzolu

- přetíženy operátory << a >> pro základní typy
- výběr na základě typu proměnné
- předdefinován cout, cin, cerr, clog (+ w…)
- knihovna iostream
- zřetězení díky návratové hodnotě typu stream
- vyprázdnění (případného) bufferu – flush, endl (´\n´+flush), ends (´\0´+flush) V obecném kódu dávat přednost použití ´\n´ - buffery se vyprázdní po naplnění (např. pro disk výhodnější). Vyprázdnění bufferu může být výhodné pro standardní konzolu

```
cin >> i >> j >> k;
cout << i << ”text” << j << k << endl;

xstream & operator xx (xstream &, Typ& p) {  }
```

- načítá se proměnný počet znaků (get, getline, ignore, read) – **gcount** zjistí kolik
- vypouštění bílých znaků – přepínače **ws, noskipws, skipws** (přeskakuje BZ na začátku, nepřeskakuje, vynechá bílé znaky (default))
  bílý znak – (isspace()) tab, enter, mezera
- načtení celého řádku **getline(kam,maxkolik,** delim**)** – čte řádek po \n nebo delim (zahodí), **get(kam, maxkolik,** delim**)** – čte řádek (delim nenačte)
- načtení znaku **get** – čte všechny znaky, zastaví se na bílém znaku
- **put** – uložení znaku (Pouze jeden znak bez vlivu formátování)
- vrácení znaku – **putback**
- ”vyčtení” znaku tak aby zůstal v zařízení – **peek**

```
int i,j;
 cout << ” zadejte dvě celá čísla \n”;
 cin>> i>> j;
 cout<<´\n´<<i<<”/”<<j<<”=”<<double(i)/j<<endl;
```

formátování základních typů

- je možné pomocí modifikátorů, manipulátorů nebo nastavením formátovacích bitů
- ovlivnění tvaru, přesnosti a formátu výstupu
- může být v ”**iomanip**”
- přetížení operátorů << a >> pro parametr typu manip, nebo pomocí ukazatelů na funkce
- přesnost výsledku má přednost před nastavením
- *manipulátory* - funkce pracující s typem stream – mají stream & jako parametr i jako návratovou hodnotu, mění parametry streamu
- slouží buď pro nastavení nové, nebo zjištění stávající hodnoty
- některé působí na jeden (následující) výstup, jiné trvale
- bity umístěny ve třídě ios (staré streamy), nově v **ios_base** – kde jsou společné vlastnosti pro input i output, které nezávisí na templatové interpretaci (ios)

- nastavení bitů – pomocí **setf** s jedním parametrem (vrátí současné)
- **setf** se dvěma parametry – nastavení bitu + nulování ostatních bitů ve skupině (označení bitu, označení společné skupiny bitů)
- nulování bitů – **unsetf**
- někdy (dříve) setioflags, resetioflags, flags
- pro uchování nastavení bitů je předdefinován typ **fmtflags**
- **i = os.width(j)** – šířka výpisu, pro jeden znak, default 0
- **os << setw(j) << i;**

- **i = os.fill(j)** – výplňový znak, pro jeden výstup, default mezera
- **os<<setfill(j) << i;**
- **ios_base::left, ios_base::right, left, right -** zarovnání vlevo vpravo – **fmtlags orig = os.setf(ios_base::left, ios_base::adjustfield)** – manipulátory bity nastaví i nulují
- **ios_base::internal**, internal – znaménko zarovnáno vlevo, číslo vpravo
- bity **left, right, internal** patří do skupiny **ios_base::adjustfield**
- **ios_base::showpos**, manipulátor **showpos, noshowpos** – zobrazí vždy znaménko (+, **-**)
- **ios_base::uppercase, uppercase, nouppercase** – zobrazení velkých či malých písmen v hexa a u exponentu

- **ios_base:: dec,ios_base::hex, ios_base::oct, dec, oct, hex** – přepínání formátů tisku (bity patří do skupiny – **ios_base::basefield**)
- **setbase** – nastavení soustavy
- **ios_base::showbase, showbase, noshowbase** – tisk 0x u hexa
- **ios_base::boolalpha, boolalpha, noboolalpha** – tisk ”true”, ”false”
- **os.precision(j)** – nastavení přesnosti, významné číslice, default 6
- **os<<setprecision(j) << i**
- **ios_base::showpoint, showpoint, noshowpoint** – nastavení tisku desetinné tečky
- **ios_base::fixed, fixed** – desetinná tečka bez exponentu
- **ios_base::scientific, scientific** – exponenciální tvar
- bity **fixed, scientific** patří do **ios_base::floatfield**
- **eatwhite** – přeskočení mezer, **writes** – tisk řetězce …

práce se soubory

- podobné mechanizmy jako vstup a výstup pro konzolu
- přetížení operátorů >> a <<
- fstream, ofstream, ifstream, iofstream
- objekty – vytváří se konstruktorem, zanikají destruktorem
- první parametr – název otevíraného souboru
- lze otevřít i metodou open, zavřít metodou close
- metoda is_open pro kontrolu otevření (u MS se vztahuje na vytvoření bufferu a pro test otevření se doporučuje metoda fail())

```
ofstream os(”nazev souboru”);
os << ”vystup”;
os.close( );
os.open(”jiny soubor.txt”);
if (!os.is_open()) …
```

- druhý parametr udává typ otevření, je definován jako enum v ios_base
- **ios_base::in** pro čtení
- **ios_base::out** pro zápis
- **ios_base:: ate** po otevření nastaví na konec souboru
- **ios_base::app** pro otevření (automaticky out) a zápis (vždy) za konec souboru
- **ios_base::binary** práce v binárním tvaru
- **ios_base::trunc** vymaže existující soubor

```
ofstream os(”soub.dat”,
ios_base::out|ios_base::ate|ios_base::binary);
istream is(”soub.txt”,ios_base::in);
fstream iostr(”soub.txt”,
                ios_base::in | ios_base::out);
```

- ios::nocreate - nově nepodporováno - otevře pouze existující soubor (nevytvoří)
- ios::noreplace - nově nepodporováno - otevře pouze když vytváří (neotevře existující)

zdroj: www.devx.com
záměna nocreate

```
fstream fs(fname, ios_base::in);
// attempt open for read
if (!fs)
{
 // file doesn't exist; don't create a new one
}
else //ok,file exists. close and reopen in write mode
{
 fs.close();
 fs.open(fname,ios_base::out); //reopen for write
}
```

záměna noreplace

```
fstream fs(fname, ios_base::in);
// attempt open for read
if (!fs)
{
 // file doesn't exist; create a new one
 fs.open(fname, ios_base::out);
}
else //ok, file exists; ??close and reopen in write mode??
{
 fs.close()
 fs.open(fname, ios_base::out); //???
// reopen for write (???)
}
```

- zjištění konce souboru – metoda **eof** – ohlásí až po načtení prvního za koncem souboru
- zjišťování stavu – bity stavu – ios_base::io_state
- **goodbit** – v pořádku
- **badbit** – vážná chyba (např. chyba zařízení, ztráta dat linky, přeplněný buffer …) – problém s bufferem (HW)
- **failbit -** méně závažná chyba, načten špatný znak (např. znak písmene místo číslice, neotevřen soubor) – problém s formátem (daty)
- **eofbit** – dosažení konce souboru
- zjištění pomocí metod – **good( ), bad( ),fail( )**(fail+badbit)**, eof( )**
- zjištění stavu -

```
if(is.rdstate()&
        (ios_base::badbid|ios_base::failbit)) …
```

- smazání nastaveného bitu (po chybě, i po dosažení konce souboru) pomocí clear(bit)

- při chybě se generují výjimky **ios_base::failure**. Výjimku je možné i nastavit pro **clear** pomocí **exceptions (iostate ist)**
- práce s binárním souborem **write(bufer, kolik), read(bufer,kolik)**
- pohyb v souboru – **seekp** (pro výstup) a **seekg** (pro vstup), parametrem je počet znaků a odkud **(ios_base::cur, ios_base::end, ios_base::beg**)
- zjištění polohy v souboru **tellp** (pro výstup) a **tellg** (pro vstup)
- **ignore** – pro přesun o daný počet znaků, druhým parametrem může být znak, na jehož výskytu se má přesun zastavit. na konci souboru se končí automaticky

implementace ve třídách

- třída je nový typ – aby se chovala standardně – přetížení << a >> pro streamy

```
istream& operator >> (istream &s, komplex &a ) {
char c = 0;
s >> c; // levá závorka
s >>a.re>>c;//reálná složka a oddělovací čárka
s>>im>>c;//imaginární složka a konečná závorka
 return s;
}

ostream &operator << (ostream &s, komplex &a ) {
s << ‘ ( ’ << a.real << ‘,’ << a.imag << ‘) ‘;
return s;
}

template<class charT, class Traits>
basic_ostream<charT, Traits> &
operator <<(basic_ostream <charT, Traits>& os,
const Komplex & dat)
```

obecná realizace

- streamy jsou realizovány hierarchií tříd, postupně přibírajících vlastnosti
- zvlášť vstupní a výstupní verze
- ios*_base – obecné definice, většina enum konstant (dříve ios), bázová třída nezávislá na typu – ios*base
- **streambuf** – třída pro práci s bufery – buďto standardní, nebo v konstruktoru dodat vlastní (pro file dědí filebuf, pro paměť strstreambuf, pro konzolu conbuf …)(streamy se starají o formátování, bufery o transport dat), pro nastavení (zjištění) buferu rdbuf
- istream, ostream – ještě bez buferu, už mají operace pro vstup a výstup (přetížené << a >>) – iostream
- iostream = istream + ostream (obousměrný)
- ifstream, ofstream – pro práci s diskovými soubory, automaticky buffer, fstream.h
- istrstream, ostrstream, strstream – pro práci s řetězci, paměť pro práci může být parametrem konstruktoru – strstream.h
- třídy xxx_withassign – rozšíření (istream, ostream) , přidává schopnost přesměrování (např. do souboru,) – cin, cout
- nedoporučuje se dělat kopie, nebo přiřazovat streamy
- stav streamu je možné kontrolovat i pomocí **if (!sout)**, kdy se používá přetížení operátoru !, které je ekvivalentní **sout.fail(),** nebo lze použít **if (sout),** které používá přetížení **operátoru ()** typu **void *operator ()**, a který vrací **!sout.fail().** (tedy nevrací good).