ZPRO 10. cvičení

Paměťové třídy proměnných

- Automatické
  - Klíčové slovo `auto`
- Statické
  - Životnost po celou dobu programu
  - Klíčové slovo `static`
- Externí
  - Globální proměnná v rámci více souborů
  - Klíčové slovo `extern`
- Registrované
  - Doporučení překladači uložit proměnnou do registru
  - Klíčové slovo `registr`

Využití automatických proměnných

- Vrácení více proměnných

- C++ 11: přes N-tice – tuple
- `#include <tuple>`
- `Funkce: tie()`

- C++ 17: přes [] (umí nahradit tie)

Příklad: Základní tvar zlomku

STL kontejnery

- Ukládají posloupnosti dat
- Rozlišujeme 4 druhy:
  - Sequence containers
    - provide access to half-open sequences of elements
  - Associative containers
    - provide associative lookup based on a key
  - Container adaptors
    - provide specialized access to underlying containers
  - Almost containers
    - are sequences of elements that provide most, but not all, of the facilities of a container

Sequence Containers

Sequence Containers	
vector<T,A>	A contiguously allocated sequence of Ts; the default choice of container
list<T,A>	A doubly-linked list of T; use when you need to insert and delete elements without moving existing elements
forward_list<T,A>	A singly-linked list of T; ideal for empty and very short sequences
deque<T,A>	A double-ended queue of T; a cross between a vector and a list; slower than one or the other for most uses

Associative Containers - Ordered

Ordered Associative Containers (§iso.23.4.2) **C** is the type of the comparison; **A** is the allocator type	
map<K,V,C,A>	An ordered map from **K** to **V**; a sequence of (**K**,**V**) pairs
multimap<K,V,C,A>	An ordered map from **K** to **V**; duplicate keys allowed
set<K,C,A>	An ordered set of **K**
multiset<K,C,A>	An ordered set of **K**; duplicate keys allowed

- Tyto kontejnery jsou implementovány jako vyvážené stromy – konkrétně red-black trees

Associative Containers - Unordered

Unordered Associative Containers (§iso.23.5.2) **H** is the hash function type; **E** is the equality test; **A** is the allocator type	
unordered_map<K,V,H,E,A>	An unordered map from **K** to **V**
unordered_multimap<K,V,H,E,A>	An unordered map from **K** to **V**; duplicate keys allowed
unordered_set<K,H,E,A>	An unordered set of **K**
unordered_multiset<K,H,E,A>	An unordered set of **K**; duplicate keys allowed

- Tyto kontejnery jsou implementovány jako hashové tabulky

Container Adaptors

Container Adaptors C is the container type	
priority_queue<T,C,Cmp>	Priority queue of Ts; Cmp is the priority function type
queue<T,C>	Queue of Ts with push() and pop()
stack<T,C>	Stack of Ts with push() and pop()

- Defaultně reprezentuje stack a priority_queue pomocí vectorem a queue pomocí deque

"Almost Containers"	
T[N]	A fixed-size built-in array: N contiguous elements of type T; no size() or other member functions
array<T,N>	A fixed-size array of N contiguous elements of type T; like the built-in array, but with most problems solved
basic_string<C,Tr,A>	A contiguously allocated sequence of characters of type C with text manipulation operations, e.g., concatenation (+ and +=); basic_string is typically optimized not to require free store for short strings (§19.3.3)
string	basic_string<char>
u16string	basic_string<char16_t>
u32string	basic_string<char32_t>
wstring	basic_string<wchar_t>
valarray<T>	A numerical vector with vector operations, but with restrictions to encourage high-performance implementations; use only if you do a lot of vector arithmetic
bitset<N>	A set of N bits with set operations, such as & and |
vector<bool>	A specialization of vector<T> with compactly stored bits

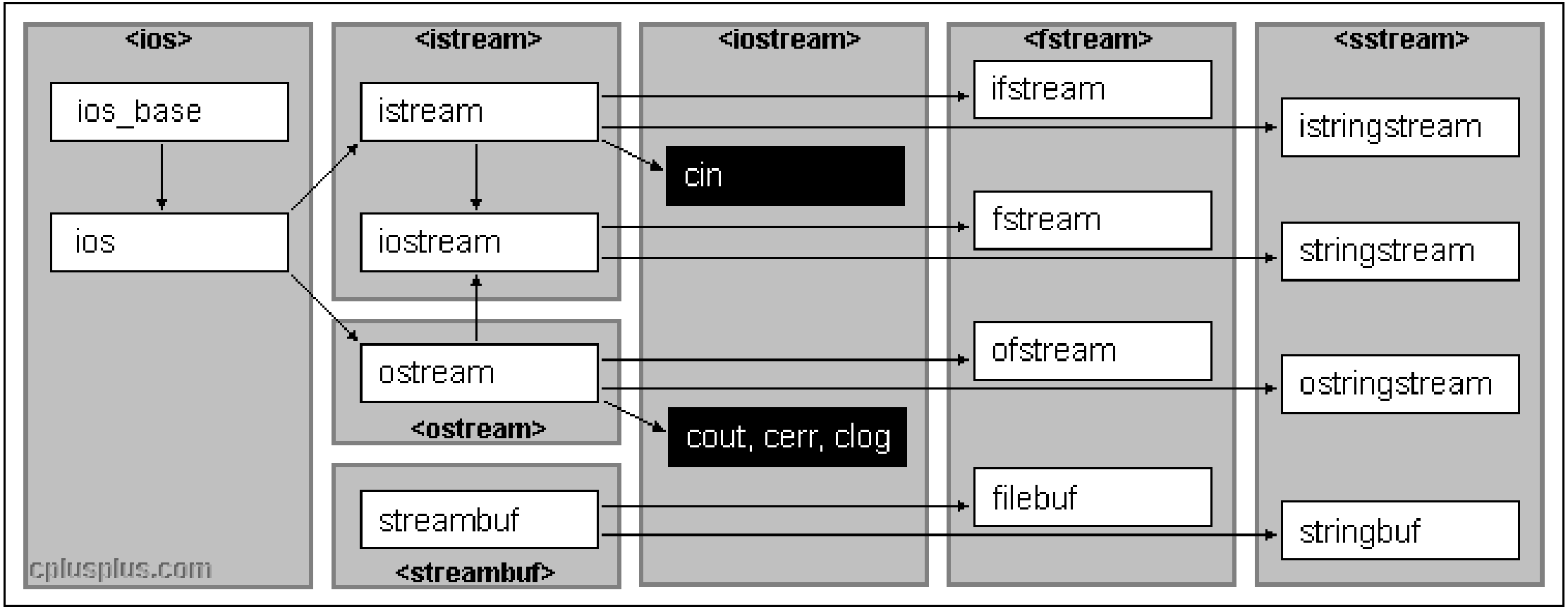
- Kopírování

Poznámka k polím

Prefer a container, such as **vector**, **string**, or **array**, over an array when you have a choice. The implicit array-to-pointer conversion and the need to remember the size for a built-in array are major sources of errors (e.g., see §27.2.1).

Prefer the standard strings to other strings and to C-style strings. The pointer semantics of C-style strings imply an awkward notation and extra work for the programmer, and they are a major source of errors (such as memory leaks) (§36.3.1).

Příklad: Ukázka vector, list, map, array


<ios>
ios_base
ios
<istream>
istream
iostream
<ostream>
ostream
<streambuf>
streambuf
<iostream>
cin
cout, cerr, clog
<fstream>
ifstream
fstream
ofstream
filebuf
<sstream>
istringstream
stringstream
ostringstream
stringbuf
cplusplus.com

Práce s datovými proudy

- Datový proud
  - Nástroj pro přenos dat mezi zdrojem a spotřebičem (např. souborem a programem)
- Standardní datové proudy v C++
  - cin – standardní vstup (obvykle klávesnice), instance třídy („proměnná“) **istream**
  - cout – standardní výstup (obvykle klávesnice), instance třídy **ostream**
  - cerr – standardní chybový výstup instance třídy **ostream**

Práce se soubory

- Se soubory pracujeme podobně jako s datovými proudy **cin** a **cout**
- Datové proudy pro soubory jsou deklarované v hlavičkovém souboru **fstream** (ASCII kódování)
  - **ifstream -** proud pouze pro vstup
  - **ofstream –** proud pouze pro výstup
  - **fstream –** pokud chceme střídat vstup a výstup
- Datové proudy pro soubory s širokými znaky (Unicode) deklarované v souboru **wfstream** (wifstream, wofsteam, wfstream)
- Rozlišujeme textové a binární soubory

- Používané metody
  - open() – otevře datový proud (soubor)
  - is_open() – vrátí bool, jestli je proud (soubor) otevřený
  - close() – uzavře datový proud (provede potřebné operace, uloží, …)
  - operátory <<, >> – fungují jako jsme viděli u cin a cout

  (jsou zděděný od společného předka ios, istream, ostream)
  - getline() – načte řádek/kus textu ze vstupního datového proudu
  - ignore() – načte a zahodí kus textu ze vstupního datového proudu
  - clear() – změní/zruší chybový stav datového proudu

Otevření souboru

```
ofstream f1, f2, f3;

f1.open("text.txt");              // relativní cesta, aktuální adresář
f2.open(".\\slozka\\text.txt"); //relativní cesta, jiný adresář (windows)
f3.open("C:\\tmp\\text.txt");   //absolutní cesta (windows)
```

Zápis do souboru

```
ofstream f1;

f1.open("text.txt");                // relativní cesta, aktuální adresář
if (f1.is_open())
{
    f1 << "Čísla od 1 do 10" << endl;
    for (int i = 1; i < 11; i++)
        f1 << i << endl;
    f1.close();
}
else
{
    // CHYBA
}
```

Příklad vypsání obsahu souboru na konzoli

```
string s;

soubor >> s;                  // jedno slovo

getline(soubor, s);           // cely radek

getline(soubor, s, '.'); // text az po znak . (tecka)
```

Metoda open

- Režimy otevření:
  - ios::app – zápis, nepřepíše obsah, začíná na konci
  - ios::binary – otevře binárně (výchozí je textově)
  - ios::in – otevřít pro vstup
  - ios::out – otevřít pro výstup
  - os::trunc – zápis, smaže obsah na začátku, přepíše (je výchozí)

Příklad datový proud jako parametr funkce

Manipulátory

- Knihovna iomanip
- Bez parametrů
  - **endl, left, right**
- S parametry
  - **setw(), setprecision(), setfill()**
- Příklad implementace vlastního manipulátoru „čára“

```
ofstream soubor;
soubor.open("pi.txt");
if (soubor.is_open())
{
    double pi = 3.14159265;
    soubor << pi << endl;
    soubor << endl << right << setw(10) << setfill('_')
        << setprecision(3) << pi << endl;
    soubor << endl << left << setw(10) << setfill('_')
        << setprecision(3) << pi << endl;
    soubor.close();
}
```