

Lokální nastavení v C a v C++

1172 Čeho všeho se lokální nastavení týká



začátečník

Mnoho lidí – nejen programátorů – má představu, že lokální nastavení, to jsou „ty háčky a čárky“. Správné zobrazování písma národního jazyka je sice nezanedbatelnou, ale v podstatě už dávno vyřešenou součástí problému. Mezi další patří:

- *Abecední řazení.* Správné řazení hesel ve slovníku, jmen, hesel v rejstříku v knihách apod. se řídí národními standardy, které se pro různé jazyky liší. U nás je to [AŘ].
- *Zápis čísel.* Téměř celá kontinentální Evropa – snad s výjimkou Řeků a Švýcarů – používá desetinnou čárku, ale neshodne se v oddělovačích tisíců. U nás rozdělujeme číslce v delších číslech do skupin po třech mezerou, setkáme se ale i s tečkou, čárkou, apostrofem, a možná i dalšími znaky.
- *Zápis měny.* Zapisuje se znak měny před číslo nebo za něj? Odděluje se od částky mezerou?
- *Zápis data.* My jsme zvyklí na pořadí den – měsíc – rok, můžeme se ale setkat snad se všemi permutacemi těchto tří údajů. Liší se i oddělovače jednotlivých položek, jména měsíců atd.
- *Zápis času.* Je obvyklé počítat se čtyřiaadvacetihodinovým cyklem nebo s dvanáctihodinovým cyklem a říkat, že jsou čtyři hodiny odpoledne?
- *Počítání dnů v týdnu.* Je prvním dnem v týdnu neděle nebo pondělí?
- *Datum přechodu ke gregoriánskému kalendáři.* Při výpočtech týkajících se historických dat narazíme na problém, že totéž datum může v různých zemích – a někdy i v různých částech téže země – znamenat jiný den podle toho, kdy byla přijata gregoriánská reforma kalendáře. První země ho přijaly roku 1582, poslední v první polovině dvacátého století.

Bohužel, ne všechny tyto problémy lze řešit v současné verzi C++.

1173 Lokální nastavení v jazyce C



začátečník

Jazyk C umožňuje předepsat lokální nastavení *pro celý program* pomocí funkce `setlocale()` definované v hlavičkovém souboru `<locale.h>`. Tato funkce má dva parametry. Prvním je celé číslo (zpravidla vyjadřované pomocí maker definovaných `<locale.h>`), jež vyjadřují kategorii, již se má nastavení týkat, a druhým je znakový řetězec obsahující jméno lokálního nastavení.

Konstanty vyjadřující kategorie jsou `LC_ALL` (všechny kategorie), `LC_COLLATE` (abecední řazení), `LC_CTYPE` (práce se znaky), `LC_MONETARY` (formátování měny), `LC_NUMERIC` (formátování čísel) a `LC_TIME` (práce s časem).

Řetězci vyjadřujícímu jméno lokálního nastavení věnujeme samostatný tip.



Poznámka: Jazyk C umožňuje stanovit lokální nastavení pouze pro program jako celek.

1174 Lokální nastavení v C++



začátečník

Pro práci s lokálním nastavením v C++ slouží třída `locale` deklarovaná v hlavičkovém souboru `<locale>`. Má několik konstruktorů; z nich se zmíníme o dvou:

- Konstruktor bez parametrů vytvoří instanci kopírující aktuální lokální nastavení.
- Konstruktor s jedním parametrem typu `const char*` očekává řetězec určující požadované nastavení. Jako druhý parametr mu lze předat číslo vyjadřující kategorii. Předáme-li tomuto konstruktoru řetězec nepředstavující podporované lokální nastavení, vyvolá výjimku typu `runtime_error`.

Vytvoření instance třídy `locale` ovšem neznamená předpis lokálního nastavení pro celý program; k tomu použijeme statickou metodu `global()` této třídy:

```
locale loc("Czech_Czech Republic.1250");
locale::global(loc);
```



Poznámka: Třída `locale` představuje vlastně kontejner na *fazety*, instance třídy `facet`, které zapouzdřují jednotlivé kategorie lokálního nastavení.

1175 Jméno lokálního nastavení



začátečník

Všechny implementace musejí umět lokální nastavení "C", jež popisuje implicitní nastavení dané programátorskými zvyklostmi, a lokální nastavení "", které odpovídá implicitnímu lokálnímu nastavení danému implementací. Mnoho implementací také zná lokální nastavení "POSIX", jehož význam opět závisí na implementaci.

Zpravidla však potřebujeme lokální nastavení pro určitý jazyk v určité zemi. Jména takovýchto lokálních nastavení závisí na implementaci, zpravidla se ale používá schéma

Jazyk_Země.KódováStránka

přičemž za jistých okolností lze kódovou stránku nebo zemi a kódovou stránku vynechat. To znamená, že např. jméno lokálního nastavení pro češtinu v Čechách s kódovou stránkou 852 může mít tvar "Czech_Czech Republic.852", podobné nastavení pro slovenštinu bude "Slovak_Slovakia.852", pro němčinu v Německu "German_germany.850", pro angličtinu v USA s kódovou stránkou pro Windows "English_United States.1252". Vedle toho se používají dvoupísmenné nebo třípísmenné zkratky, např. "cs_CZ", "cze", "sky" (slovenština) atd. Třípísmenné zkratky jsou určeny mezinárodním standardem ISO/IEC 3166.



Poznámka: Překladače pro Windows zpravidla podporují „dlouhé“ názvy, jako je "Czech_Czech Republic.852"; v překladačích pro unixové systémy se setkáme spíše se zkratkami, jako je „de_DE“ (němčina v Německu) atd.

1176 Kódové stránky



Kódová stránka určuje význam úzkých znaků s kódovým číslem vyšším než 127. Pro nás mohou být zajímavé tyto kódové stránky:

Kódová stránka	Význam
1250	Středoevropské jazyky pod Windows
852	Středoevropské jazyky pod DOS a v konzolovém okně Windows
28592	Středoevropské jazyky v kódování ISO 8859-2
10029	Čeština pro Macintosh
1252	Západoevropské jazyky pod Windows
850	Západoevropské jazyky pod DOS a v konzolovém okně Windows
437	Angličtina pod DOS a v konzolovém okně Windows
866	Jazyky používající cyrilici pod DOS a v konzolovém okně Windows
1251	Jazyky používající cyrilici pod Windows

1177 Výpis textu do souboru v daném kódování (jazyk C)



Do textového souboru `data.txt` chceme v kódové stránce 852 zapsat známou větu o žlutém koni a ďábelských ódách, jež obsahuje všechny znaky české abecedy s diakritickými znaménky. Nejprve tedy předepíšeme lokální nastavení:

```
setlocale(LC_ALL, "Czech_Czech Republic.852");
```

Dále vytvoříme proud, zapíšeme do něj řetězec a proud zase uzavřeme:

```
wchar_t * text = L"Žluťoučký kůň příšerně úpěl ďábelské ódy";
FILE *F = fopen("data.txt", "w");
fwprintf(F, text);
fclose(F);
```

To je vše.



Upozornění: Neznám rozumný důvod, proč by totéž nemělo fungovat i pro úzké znaky a funkci `fprintf()`. Nicméně překladač, ve kterém si tyto příklady připravuji (Visual C++ 2008), pro úzké znaky lokální nastavení ignoruje a použije kódovou stránku 1250. Podobně se chová i C++Builder 2009.

1178 Čtení textu ze souboru v daném kódování (jazyk C)



V souboru `data.txt` je znakový řetězec s větou o žlutém koni a my ho chceme přečíst a uložit v programu. Postup je jednoduchý: Předepíšeme lokální nastavení, otevřeme proud pro vstup, přečteme z něj řetězec širokých znaků a proud zase uzavřeme.

```
setlocale(LC_ALL, "Czech_Czech Republic.852");
```

```
wchar_t text[100];
FILE *F = fopen("data.dta", "r");
fwscanf(F, L"%30c", text);
fclose(F);
```



Upozornění: Stejně jako v případě zápisu ani zde nevidím jediný důvod, proč by podobná operace neměla fungovat i pro úzké znaky. Nicméně v mých překladačích nefunguje.

1179 Výpis textu do souboru v daném kódování pomocí objektových proudů (C++)



Při práci s objektovými datovými proudy jazyka C++ nemusíme předepisovat globální lokální nastavení; stačí vytvořit instanci a připojit ji k proudu pomocí metody `imbue()`. To znamená, že chceme-li vypsat větu o žlutém koni do souboru v kódování 8859-2, použijeme následující příkazy:

```
wchar_t * text1 = L"Žluťoučký kůň příšerně úpěl ábelské ódy";
wofstream F("data.dta");
F.imbue(locale("Czech_Czech Republic.28592"));
F << text1 << endl;
F.close();
```

Všimněte si, že kódování ISO 8859-2 se skrývá pod číslem 28592.



Upozornění: I tentokrát bylo třeba použít široké znaky, aby tento kód fungoval.

1180 Čtení textu ze souboru v daném kódování pomocí objektových datových proudů (C++)



Máme soubor v kódové stránce 852, chceme z něj přečíst jeden řádek textu a uložit ho v instanci `lajna` typu `wstring`. K tomu použijeme následující příkazy:

```
wstring lajna;
wifstream F("data.dta");
F.imbue(locale("Czech_Czech Republic.852"));
getline(F, lajna);
F.close();
```

Zde jsme využili funkci `getline<>()` deklarovanou v hlavičkové souboru `<string>`, jež umí načíst celý řádek včetně mezer.

1181 Výpis českého textu na konzolu a čtení z ní



Chceme-li vypisovat na konzolu český text, musíme nastavit u proudů směřujících na konzolu odpovídající lokální nastavení. To znamená, že pod Windows by mělo stačit např.

```
locale loc("Czech_Czech Republic.852");
cout.imbue(loc);
```

a následující příkaz

```
cout << " Žluťoučký kůň příšerně úpěl ďábelské ódy" << endl;
```

by měl správně vypsat uvedenou větu. Nicméně překladače, které mám k dispozici (C++Builder 2009 a Visual C++ 2008) to dokáží pouze v případě, že použijeme široké znaky a odpovídající proudy, tedy že napíšeme

```
wcout.imbue(loc);
wcout << L" Žluťoučký kůň příšerně úpěl ďábelské ódy" << endl;
```

Podobně při čtení českého textu musíme použít proud `wcin` a číst široké znaky.

1182 Abecední řazení: obecné problémy



Pravidla abecedního řazení jsou v řadě zemí upravena národními normami; u nás je to [AŘ].

Pro abecední řazení nelze přímo využít porovnávání kódových čísel znaků, protože abecední znaky netvoří v žádné kódové stránce – ani v Unicode – souvislou řadu. Tu tvoří pouze odděleně malá a velká písmena anglické abecedy, a to se nehodí ani pro angličtinu.

Vedle toho ovšem můžeme narazit na další problémy:

- Některé znaky se nerozlišují. Například převážná většina jazyků – včetně češtiny – při řazení nerozlišuje malá a velká písmena, tzn. *a* a *A* se pokládají za týž znak. Vedle toho např. finština nerozlišuje při řazení *V* a *W*.
- Jeden znak se při řazení považuje za skupinu znaků. Např. německé přehlásky *ä*, *ö*, *ü* se řadí jako *ae*, *oe*, *ue*. (V češtině se s ničím podobným nesetkáme.)
- Skupina znaků se řadí jako jeden znak. V češtině se takto chová *ch*, ve španělštině *ch* a *ll*, v chorvatštině *dž* aj.
- Některé znaky s akcenty mají tzv. sekundární řadící platnost. To znamená, že při porovnávání dvou hesel se tyto znaky nejprve berou bez diakritického znaménka, a teprve když je takto nelze rozlišit, vezmou se v úvahu diakritická znaménka. S takovými pravidly se vedle češtiny setkáme např. v některých románských jazycích.

1183 Abecední řazení v češtině



Protože jsou pravidla českého abecedního řazení pro mnoho programátorů pravidelným zdrojem překvapení, dovolím si zde ve stručnosti uvést to nejpodstatnější:

1. České abecední řazení je dvouřádkové.
2. Znaky *a*, *b*, *c*, *č*, *d*, *e*, *f*, *g*, *h*, *ch*, *i*, *j*, *k*, *l*, *m*, *n*, *o*, *p*, *q*, *r*, *ř*, *s*, *š*, *t*, *u*, *v*, *w*, *x*, *y*, *z*, *ž* mají primární řadící platnost a řadí se v pořadí zde uvedeném.

3. Znaký *á, ď, é, ě, í, ň, ó, ě, ú, ů, ý* mají sekundární řadící platnost, tj. při prvním průchodu se berou jako odpovídající znaky bez diakritických znamének.
4. Nelze-li určit pořadí hesel v prvním průchodu, použije se druhý průchod. Při něm se znaky se sekundární řadící platností řadí za odpovídající znaky s primární řadící platností (tedy *á* za *a*).
5. Je-li třeba porovnávat znaky se sekundární řadící platností mezi sebou, řadí se v pořadí, v jakém jsou uvedeny v bodě 3.

Další pravidla určují zacházení s diakritickými znaménky a znaky z jiných abeced atd.; ty si zde dovoluji pominout.

Z těchto pravidel plyne, že např. platí "ťuhýk" < "tygr" (znak < znamená „je v abecedě před“). Nejdříve se totiž porovnají hesla obsahující pouze znaky s primární řadící platností, tedy "ťuhk" a "tygr", a protože zde je pořadí již jasné, skončíme. Na druhé straně při určování pořadí "padla" < "padlá" < "pádla" znaky s primární řadící platností o pořadí nerozhodnou, a proto se pořadí určí až při druhém průchodu na základě skutečnosti, že *a* je před *á*.

1184 Porovnání dvou řetězců podle pravidel abecedního řazení v C



Chceme-li určit pořadí dvou hesel – řetězců – podle pravidel abecedního řazení, použijeme funkci `collate()` definovanou v hlavičkovém souboru `<string.h>`; pracujeme-li se širokými znaky, použijeme funkci `wscoll()` definovanou buď ve `<string.h>`, nebo v `<wchar.h>`.

Tyto funkce vracejí zápornou hodnotu, jsou-li předané řetězce ve správném pořadí (první parametr je v abecedě před druhým, tedy „je menší“ než druhý), nulu, jsou-li si rovný, a kladnou hodnotu, jsou-li v nesprávném pořadí.

Máme např. znakové pole

```
char *pole[] = {"hor", "chór", "ťuhýk", "tygr", "Anna"};
```

a chceme-li porovnat třetí a čtvrtý prvek, napíšeme

```
setlocale(LC_ALL, "czech");
int i = strcoll(pole[2], pole[4]);
```

V tomto případě vrátí funkce `strcoll()` kladnou hodnotu, neboť *ťuhýk* je v abecedě před *tygrem*. (Viz též tip 344.)

1185 Abecední řazení pole řetězců v jazyce C



Máme opět pole řetězců

```
char *pole[] = {"hor", "chór", "nula", "ňouma", "Anna"};
```

a chceme ho seřadit podle pravidel českého abecedního řazení. K tomu použijeme funkci `qsort()` deklarovanou v `<stdlib.h>` a jako porovnávací funkci použijeme vhodně zabalenou funkci `strcoll()`:

```
int porovnej(const void* prvek1, const void* prvek2)
{
    return strcoll(*(char**)prvek1, *(char**)prvek2);
}
```

Pak už zbývá jen určit lokální nastavení a zavolat `qsort()`. (Viz též tip 1132.)

```
setlocale(LC_ALL, "czech"); // Stačí jazyk, není třeba celé jméno
qsort(pole, 5, sizeof(const char*), porovnej);
```

Výsledkem bude pole obsahující uvedené řetězce v pořadí "Anna", "hor", "chór", "ňouma" a "nula".

1186 Abecední porovnání dvou instancí typu string v C++



Máme dvě instance typu `string` a chceme zjistit jejich vzájemné pořadí při abecedním řazení. K tomu můžeme využít instance třídy `locale`, jejíž přetížený operátor volání funkce vrací `true`, je-li první parametr v abecedě před druhým, a `false` v opačném případě (tedy i v případě rovnosti). To znamená, že máme-li pole

```
string pole[] = {"Zdena", "Hór", "chór", "nula", "ňouma"};
```

můžeme napsat

```
locale loc("czech");
bool je = loc(pole[1], pole[2]);
```

a protože řetězce "Hór" a "chór" jsou ve správném pořadí, bude proměnná `je` obsahovat `true`.

1187 Abecední řazení pole řetězců v C++



Při řazení pole řetězců typu `string` nebo `wstring` použijeme funkci `sort<>()`, které jako třetí parametr – komparátor – předáme instanci třídy `locale`. To znamená, že chceme-li seřadit pole řetězců

```
wstring pole[] = {L"Zdena", L"Hór", L"chór", L"nula", L"ňouma"};
```

použijeme příkaz

```
locale loc("Czech_Czech Republic.852");
sort(pole, pole+5, loc);
```

1188 Formátování čísel v jazyce C



Při výstupu reálných čísel pomocí funkcí z rodiny `fprintf()` se použije desetinná čárka nebo tečka v závislosti na aktuálním lokálním nastavení. Také při vstupu reálných čísel pomocí funkcí z rodiny `fscanf()` bude program očekávat desetinnou čárku nebo tečku v závislosti na aktuálním lokálním nastavení.

Oddělovače tisíců – alespoň v současné verzi – nejsou podporovány ani při vstupu, ani při výstupu, i když jsou součástí dat o lokálním nastavení.

1189 Údaje o nastavených hodnotách pro formátování čísel



V jazyce C je k dispozici datový typ `struct lconv`, jehož složky obsahují údaje o nastavených hodnotách pro formátování čísel. Například složka `decimal_point` typu `char*` obsahuje řetězec pro vyjádření desetinné tečky nebo čárky, složka `thousands_sep` (opět typu `char*`) obsahuje řetězec vyjadřující oddělovač tisíců, složka `currency_symbol` obsahuje symbol měny atd.

Ukazatel na strukturu `lconv` s aktuálním nastavením získáme voláním funkce `localeconv()` bez parametrů, jejíž prototyp najdeme v hlavičkovém souboru `<locale.h>`. Tento ukazatel však nesmíme využít ke změnám nastavení.

1190 Formátování čísel při výstupu v C++



Chceme-li formátovat vystupující čísla podle lokálního nastavení, použijeme v C++ opět metodu `imbue()` datových proudů. Příkazy

```
double n = 12345.765432;
cout.imbue(locale("Czech_Czech republic.852"));
cout << fixed << setprecision(3) << n << endl;
```

způsobí výstup

```
12 345,765
```

Použijeme-li místo českého nastavení `"German_Germany.850"` a poručíme-li si 5 desetinných míst, dostaneme výstup

```
12.345,76543
```

a při nastavení `"French_Switzerland.850"` dostaneme

```
12'345.76543
```



Upozornění: V některých starších překladačích (např. Visual C++ 2002) fungoval výstup čísel formátovaných podle lokálního nastavení pouze pro proudy orientované na široké znaky, v jiných se podle lokálního nastavení řídilo pouze použití desetinné tečky nebo čárky nebo ani to ne.

1191 Čtení formátovaných čísel v C++



Počítač by měl dokázat po sobě přecházet, co napsal – tedy měl by umět číst čísla formátovaná podle národních zvyklostí. V C++ to většinou funguje (až na jisté výjimky, jak uvidíme dále).

Základem je opět „napustit“ lokální nastavení do vstupního proudu pomocí metody `imbue()`

```
cin.imbue(locale("Czech_Czech Republic.852"));
```

Je-li `x` proměnná typu `double` a napíšeme-li

```
cout << "Zadej číslo";
cin >> n;
```

můžeme v konzolovém okně napsat

```
123456,7
```

a zadaná hodnota se správně načte. Napíšeme-li ovšem

```
12 3456,7
```

načte se pouze hodnota 12, neboť mezera působí jako oddělovač vstupních polí. Na stejný problém narazíme i ve francouzském nastavení ("`French_France.850`"). Na druhé straně, nastavíme-li německé prostředí ("`german_germany.850`") a zadáme-li

```
12.3456,7
```

přečte se zadaná hodnota bez problémů. Na problémy nenarazíme v žádném prostředí, které používá jako oddělovač jiný znak než mezeru.

1192 Převody mezi úzkými a širokými znaky



Převodem mezi širokými a úzkými znaky jsme se zabývali už v tipech 302, 303, 361, 362, 396 a 397. Další nástroje pro převod mezi úzkými a širokými znaky v závislosti na lokálním nastavení nabízejí datové proudy.

Metodě `narrow()` předáme znak ve znakové sadě proudu a ona nám vrátí odpovídající hodnotu typu `char`, pokud existuje. Jako druhý parametr můžeme předat znak, který má tato metoda vrátit, jestliže v úzké znakové sadě odpovídající znak neexistuje; implicitně je to znak s hodnotou 0.

Metodě `widen()` předáme úzký znak (typu `char`) a ona nám vrátí znak ve znakové sadě daného proudu.

1193 Fazety lokálního nastavení



Třída `locale` se v podstatě chová jako kontejner (nikoli kontejner ve smyslu STL) obsahující fazety, což jsou instance šablonových tříd odvozených od třídy `facet`. Jsou rozděleny do několika kategorií. Parametrem šablony je znakový typ.

V kategorii `numeric` jsou fazety `num_get<>`, `num_put<>` a `num_punct<>`, v kategorii `time` najdeme fazety `time_get<>` a `time_put<>`, v kategorii `monetary` leží fazety `money_get<>`, `money_put<>` a `money_punct<>`, v kategorii `ctype` je to `ctype<>` a `codecvt<>`, v kategorii `collate` najdeme fazetu `collate<>` a v kategorii `messages` je jediná fazeta `messages<>`.

Fazety, jejichž jména končí `_get`, se týkají vstupních operací, fazety se jménem končícím `_put` se týkají výstupních operací. Fazety končící `_punct` definují formátovací znaky, jako je desetinná tečka nebo čárka apod.

Fazeta `ctype<>` je věnována klasifikaci znaků, fazeta `codecvt<>` má na starosti převody mezi kódováními.

K některým z uvedených fazet existují ještě fazety `XXX_byname<>`. Fazeta s tímto jménem poskytuje stejné služby jako fazeta `XXX`, pouze má navíc konstruktor s parametrem, kterým je řetězec se jménem lokálního nastavení.

Programátor může definovat vlastní fazety a přidat si je do lokálního nastavení.

1194 Jak použít fazetu lokálního nastavení



Chceme-li pracovat s fazetou lokálního nastavení, použijeme šablonovou funkci `use_facet<>()`, které jako parametr šablony zadáme jméno požadované fazety a jako parametr funkce pak instanci třídy `locale`; tato funkce vrátí odkaz na požadovanou fazetu. Pozor, parametr šablony se v signatuře této funkce objevuje pouze jako typ vrácené hodnoty, takže ho musíme uvádět.

Představte si, že máme proměnnou

```
wchar_t c = L'Ř', d;
```

a chceme použít služby fazety `ctype` českého lokálního nastavení k převodu velkého písmene na malé. K tomu nám poslouží příkaz

```
c = use_facet<std::ctype<wchar_t>>(<loc>).tolower(c);
```



Poznámka: Funkce `tolower<>()`, definovaná v hlavičkovém souboru `<locale>`, s níž jsme se setkali v tipu 315, je jen „uživatelsky příjemná“ zkratka pro uvedené volání.

1195 Zjištění systémového času a data



Chceme-li zjistit aktuální čas, použijeme funkci `time()`, jež vrací hodnotu typu `time_t` (jde o počet sekund od půlnoci 1.1.1970). Této funkci předáme jako parametr ukazatel

na proměnnou, do níž chceme výsledek uložit; předáme-li hodnotu 0, funkce výsledek neuloží, pouze vrátí.

Pomocí funkce `gmtime()` můžeme převést hodnotu typu `time_t` na hodnotu typu `struct tm`, jež obsahuje složky představující den, měsíc, rok, hodinu, minutu atd. Hodnotu typu `struct tm` lze potom formátovat v závislosti na lokálním nastavení pomocí funkce `strftime()`. Podívejme se na příklad:

```
time_t cas;  
cas = time(0);  
struct tm *tajm = gmtime(&cas);
```

Všechny funkce a datové typy uvedené v tomto tipu jsou definovány v hlavičkovém souboru `<time.h>`.

1196 Formátování data a času: funkce `strftime` nebo `wcsftime`()



začátečník

Chceme-li formátovat datum a čas podle aktuálního lokálního nastavení, použijeme funkci `strftime()` definovanou v hlavičkovém souboru `<time.h>`. Tato funkce očekává čtyři parametry.

1. Pole typu `char` dostatečně velké, aby mohlo pojmout řetězec, který bude výsledkem.
2. Celé číslo udávající velikost tohoto pole (maximální možnou velikost výsledku).
3. Formátovací řetězec vytvářený podobně jako formátovací řetězec funkcí `fprintf()` (srovnej tip 1024 a následující).
4. Ukazatel na proměnnou typu `struct tm`, jež obsahuje čas, který chceme formátovat.

Výsledek bude uložen v prvním parametru.

Pracujeme-li se širokými znaky, použijeme funkci `wcsftime()` deklarovanou v hlavičkovém souboru `<time.h>` nebo `<wchar.h>`, která se od `strftime()` liší jen tím, že první a třetí parametr jsou řetězce širokých znaků.

1197 Výpis aktuálního data a času



začátečník

Proměnná `tajm` obsahuje ukazatel na strukturu `tm`, jež obsahuje aktuální datum a čas, který chceme převést na znakový řetězec. K tomu použijeme příkazy

```
setlocale(LC_ALL, "Czech_Czech Republic.852");  
char text[100]; // Zde bude výsledek  
strftime(text, 100, "%c", tajm);
```

Po provedení těchto příkazů bude pole `text` obsahovat řetězec

```
8.1.2010 15:09:23
```

Kdybychom použili implicitní lokální nastavení "C", dostali bychom výsledek

01/08/10 15:09:23

Při nastavení "English_United_States.437" dostaneme např.

1/8/2010 3:09:23 PM

1198 Formátovací řetězec pro strftime() [37-26]



Formátovací řetězec používaný funkcí `strftime()` podléhá podobným pravidlům jako řetězec používaný funkcemi z rodiny `printf()`. Je to libovolný znakový řetězec, který může obsahovat několik specifikací konverzí. Ty začínají znakem `%`, za nímž následuje znak určující konverzi. Mezi znakem `%` a specifikací konverze může být ještě modifikátor `E` nebo `O`.

Všechny specifikace konverzí budou z formátovacího řetězce odstraněny a nahrazeny znaky vyjadřujícími odpovídající údaj, a to podle lokálního nastavení (kategorie `LC_TIME`).

1199 Význam specifikací konverzí pro funkci strftime()



Význam specifikací konverzí ukazuje následující tabulka. V hranatých závorkách je uvedena odpovídající složka struktury `tm`. Všechny reprezentace (jména i zkratky) odpovídají platnému lokálnímu nastavení. Konverze jsou uvedeny podle standardu [ISOC].

Konverze	Význam (čím bude nahrazena)
<code>%a</code>	zkratka dne v týdnu [<code>tm_wday</code>]
<code>%A</code>	plné jméno dne v týdnu [<code>tm_wday</code>]
<code>%b</code>	zkratka jména měsíce (nebo římská číslice) [<code>tm_mon</code>]
<code>%B</code>	plné jméno měsíce [<code>tm_mon</code>]
<code>%c</code>	datum a čas
<code>%C</code>	rok vyjádřený jako dvojice číslic (tedy číslem v rozmezí 00–99) [<code>tm_year</code>] (!)
<code>%d</code>	dvouciferné pořadové číslo dne v měsíci (01–31). [<code>tm_mday</code>]
<code>%D</code>	je ekvivalentní řetězci " <code>%m/%d/%y</code> ". [<code>tm_mon</code> , <code>tm_mday</code> , <code>tm_year</code>] (!)
<code>%e</code>	pořadové číslo v měsíci (1–31); jednociferné číslo bude doplněno mezerou. [<code>tm_mday</code>] (!)
<code>%F</code>	je ekvivalentní " <code>%Y-%m-%d</code> " (formát data podle ISO 8601). [<code>tm_year</code> , <code>tm_mon</code> , <code>tm_mday</code>] (!)
<code>%g</code>	poslední dvě číslice roku podle týdnů (viz dále) (00–99). [<code>tm_year</code> , <code>tm_wday</code> , <code>tm_yday</code>] (!)
<code>%G</code>	rok podle týdnů (viz dále) (např. 1997) [<code>tm_year</code> , <code>tm_wday</code> , <code>tm_yday</code>] (!)
<code>%h</code>	je ekvivalentní " <code>%b</code> ". [<code>tm_mon</code>] (!)
<code>%H</code>	hodina při 24hodinovém cyklu (00–23). [<code>tm_hour</code>]
<code>%I</code>	hodina při 12hodinovém cyklu (01–12). [<code>tm_hour</code>]

Konverze	Význam (čím bude nahrazena)
%j	pořadové číslo dne v roce (001–366). [tm_yday]
%m	pořadové číslo měsíce v roce (01–12). [tm_mon]
%M	minuta (00–59). [tm_min]
%n	přechod na nový řádek (!)
%p	označení denní doby při 12hodinovém cyklu (dop., odp., AM, PM apod.) [tm_hour]
%r	čas při 12hodinovém cyklu. [tm_hour, tm_min, tm_sec] (!)
%R	je ekvivalentní "%H:%M" [tm_hour, tm_min] (!)
%S	vteřina (sekunda) jako číslo v rozmezí 00–60 [tm_sec] (!)
%t	tabulátor (!)
%T	je ekvivalentní "%H:%M:%S" (formát času podle ISO 8601) [tm_hour, tm_min, tm_sec] (!)
%u	pořadové číslo dne v týdnu podle ISO 8601 (1–7, pondělí má číslo 1 [tm_wday] (!)
%U	pořadové číslo týdne v roce (první neděle je první den prvního týdne) jako číslo v rozmezí 00–53. [tm_year, tm_wday, tm_yday]
%V	pořadové číslo týdne v roce podle ISO 8601 (viz dále) jako číslo v rozmezí 01–53. [tm_year, tm_wday, tm_yday] (!)
%w	pořadové číslo dne jako číslo v rozmezí 0–6, neděle má číslo 0. [tm_wday]
%W	pořadové číslo týdne v roce (první poddělí jako první den prvního týdne) jako číslo v rozmezí 00–53 [tm_year, tm_wday, tm_yday]
%x	reprezentace data podle lokálního nastavení
%X	reprezentace času podle lokálního nastavení
%y	poslední 2 číslice roku (00–99) [tm_year]
%Y	rok (např. 1997) [tm_year]
%z	posunutí oproti univerzálnímu času (UTC, Greenwichský čas) ve formátu podle ISO 8601 např. 0430 znamená 4 hodiny a 30 minut za UTC, tedy na západ od Greenwiche), nebo nic, není-li možno určit časové pásmo [tm_isdst]
%Z	jméno nebo zkratka časového pásma, nebo nic, nelze-li časové pásmo určit [tm_isdst]
%%	vypíše znak %.

Ne všechny komerční překladače však implementují všechny konverze. Např. v této tabulce jsem označil vykřičníkem v závorkách konvence, které neimplementuje Visual C++ 2008. Navíc konverze %z a %Z se v tomto překladači chovají stejně, vypíší např. Střední Evropa (běžný čas). (Píši tuto kapitolu v zimě.)

Překladač C++Builder zná konverzi ještě méně.

1200 Modifikátory E a O



Podle standardu lze mezi znak % a znak vyjadřující požadovanou konverzi vložit modifikátory E nebo O.

- Modifikátor E lze použít v konverzích %Ec, %EC, %Ex, %EX, %Ey a %EY a znamená „alternativní reprezentaci zobrazovaných údajů“; standard ovšem nespecifikuje, co to znamená.
- Modifikátor O (písmeno „O“) lze použít v konverzích %Od, %Oe, %OH, %OI, %Om, %OM, %OS, %Ou, %OU, %OV, %Ow, %OW, %Oy a znamená použití alternativních symbolů pro číslice.

Komerční překladače, které mám k dispozici, však tyto modifikátory neimplementují.

1201 Nestandardní modifikátor # (Visual C++)



Překladač Visual C++ 2008 nabízí místo modifikátorů E a O modifikátor # s následujícím významem:

Konverze %c znamená „dlouhý formát data a času“. Při českém nastavení například dostaneme

```
8. leden 2010 19:46:30
```

a při americkém nastavení

```
Friday, January 08, 2010 7:47:50 PM
```

Konverze %x znamená „dlouhý formát data“, např.

```
8. leden 2010
```

V americkém nastavení je zde opět navíc jméno dne v týdnu.

Konverze %d, %H, %I, %j, %m, %M, %S, %U, %w, %W, %y, %Y mají stejný účinek jako odpovídající konverze bez znaku #, pouze se v nich nahradí úvodní nuly mezerami, pokud tam ovšem nějaké úvodní nuly jsou. To znamená, že např. formátovací řetězec "%d. %m." vytvoří řetězec "08. 01.", zatímco "%d. %m." vytvoří " 8. 1.".

1202 Počítání týdnů v roce podle ISO 8601



Podívejme se na způsob určení prvního týdne v roce předepsaný standardem ISO 8601 a používaný i u nás. Podle tohoto standardu týden začíná pondělím (na rozdíl od Ameriky, kde začíná nedělí). Za první týden v roce se počítá týden, který obsahuje 4. ledna. Je to první týden v roce, který obsahuje čtvrtek, a také první týden v roce, který má alespoň 4 dny.

Jestliže týden, který obsahuje 1. leden, má jen 3 dny z nového roku nebo méně, počítá se jako 53. týden z roku minulého. Pak hovoříme o „roku podle týdnů“ (week based year).

1203 Datum ve formátu ISO 8601



znalec

Datum ve formátu ISO 8601 se skládá ze čtyřciferného čísla roku, pomlčky, dvouciferného čísla měsíce (v případě potřeby zleva doplněného nulou), další pomlčky a dvouciferného čísla dne (opět v případě potřeby doplněného zleva nulou). Například 8.1.2010 zapíšeme ve formátu ISO jako 2010-01-08.

Výhodou tohoto formátu je, že takto zapsaná data lze snadno porovnávat, a to lexikograficky, tedy např. pomocí funkce `strcmp()` – viz tip 343.

Chceme-li získat pomocí funkce `strftime()` datum ve formátu ISO, použijeme konverzi `"%F"`, pokud je v dané implementaci k dispozici, nebo `"%Y %m %d"`.

1204 Identifikátory v programech



pokročilý

Standard jazyků C a C++ umožňuje používat v identifikátorech i znaky národních abeced a překladače to konečně také začínají dovolovat.

Jsem si vědom, že pro mnohé české programátory je představa identifikátoru obsahujícího znaky národních abeced něčím svatokrádežným – jednou mi na toto téma kdosi napsal, že „to opravdový profesionál nedělá!!!“, a to s ještě podstatně větším počtem vykřičníků.

Nicméně skuteční profesionálové, s nimiž jsem se setkal, to dělají, jestliže jim to pomůže při psaní programu; dobře zvolené jméno je jedním z důležitých předpokladů pro srozumitelnost programu a použití vlastního jazyka je nejpohodlnější cestou, jak se k tomu dostat.