VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PŘEKLADAČ NOVÉHO MODULÁRNÍHO JAZYKA

COMPILER OF NEW MODULAR LANGUAGE

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE **DANIEL ČEJCHAN**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZBYŇEK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citace

ČEJCHAN, Daniel. *Překladač nového modulárního jazyka*. Brno, 2017. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Křivka Zbyňek.

Překladač nového modulárního jazyka

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph. D. ... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Daniel Čejchan
14. listopadu 2016

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Obsah

Kapitola 1

Úvod

Tento dokument popisuje a zdůvodňuje část z mnoha rozhodnutí, která byla vykonána při procesu návrhu programovacího jazyka NATI a vzorového překladače pro něj. Rámcově prozkoumává syntaktické i sémantické prvky moderních programovacích jazyků a popisuje i nové koncepty a prvky.

Programovací jazyk se hodně inspiruje jazyky C++ a D[1]. Tyto jazyky budou používány pro srovnávání syntaxe a efektivity psaní kódu.

1.1 Motivace

Mou hlavní motivací je nespokojenost se stávajícími programovacími jazyky. Programování mám jako koníčka už od nějakých dvanácti let, a tak jsem ještě před nastoupením na FIT získal nějaké zkušenosti z psaní aplikací v Object Pascalu (Delphi), C++ (Qt, SDL + OpenGL) a PHP (webové aplikace). FIT pak můj repertoár (i když jen rámcově) rozšířil na valnou většinu dnes používaných jazyků. Bohužel jsem ale nenarazil na žádný, který by splňoval mé požadavky. Rozhodl jsem se tedy využít nutnost napsání bakalářské práce k uskutečnění mého dlouholetého snu.

Jazyk D Mému srdci nejbližší jazyk, na který jsem ve svém pátrání narazil, byl jazyk D. Jedná se o kompilovaný jazyk vycházející z C++ (binárky jsou do jisté míry kompatibilní) s velice podobnou syntaxí. Největší rozdíly jsou modulový systém[2] (zdrojový kód je rozdělen do modulů, které se vzájemně, i rekurzivně, importují; odpadá nutnost psát hlavičkové soubory), značně rozšířená funkčnost metaprogramování s šablonami a rozšířená schopnost vykonávat funkce za doby kompilace.

D mi byl velikou inspirací při navrhování mého jazyka. Bohužel i v D jsem narazil na strop možnosti (ačkoli byl značně výš než třeba v C++), kdy některé věci nešly napsat tak jednoduše, jak bych chtěl. Tento jazyk však dokazuje, že tato bariéra může být mnohem dál. Já ji chci ve svém jazyku ještě více posunout.

1.2 Cíle

[1] http://dlang.org/
[2] http://dlang.org/spec/module.html

Kapitola 2

Vlastnosti a syntaxe jazyka

Nejdříve musíme určit základní rysy jazyka, které potom budeme dále rozvíjet.

Použití jazyka Naším cílem je navrhnout tzv. *general purpose language*, tedy jazyk nezaměřený na konkrétní případy užití. Cílíme vytvořit „nástupce“ jazyka C++, který by se dal použít ve všech případech, kde se dá využít C++ – tedy i třeba na mikroprocesorových systémech. Složitější struktury se tím pádem budeme snažit řešit spíše vhodnou abstrakcí než zaváděním prvků, které kladou zvýšené nároky na výkon a paměť.

Kompilovaný vs. interpretovaný jazyk Ačkoli interpretované jazyky mají jisté výhody, platí se za ně pomalejším kódem a nutností zavádět interpret. V rámci této práce budeme navrhovat kompilovaný jazyk. Pro zjednodušení práce při psaní překladače nebudeme ale překládat přímo do strojového kódu, ale do jazyka C.

Syntaxe jazyka Abychom maximálně usnadnili přechod případných programátorů k našemu jazyku a zkrátili učební křivku, je vhodné se co nejvíce inspirovat již existujícími jazyky. Vzhledem k tomu, že se jedná o kompilovaný jazyk, je nejrozumnější vycházet se syntaxe rodiny jazyků C, které jsou hojně rozšířené a zažité.

Programovací paradigmata Jelikož vycházíme z jazyků C++ a D, přejímáme i jejich paradigmata a základní koncepty. Náš jazyk bude tedy umožňovat strukturované, funkcionální i objektově orientované programování.

2.1 Běžné prvky programovacích jazyků

2.1.1 Systém dedičnosti tříd

Jazyky C++ a D mají různě řešené třídní systémy. Zatímco v C++ existuje jen jeden typ objektu, který pracuje s dědičností (*class* a *struct* jsou z pohledu dědění identické), a to s dědičností vícenásobnou a případně i virtuální, D má systém spíše podobný Javě – třídy (*class*) mohou mít maximálně jednoho rodiče, navíc ale existují rozhraní (*interface*), které však nemohou obsahovat proměnné.

Ačkoli vícenásobné dědění není třeba často, občas potřeba je a jen těžko se nahrazuje. Rozhraní nemohou obsahovat proměnné, což omezuje jejich možnosti. D nabízí ještě jedno

řešení – tzv. *template mixins*[1] – které funguje velice podobně jako kopírování bloků kódu přímo do těla třídy. Toto řešení rozbíjí model dědičnosti (na *mixiny* se nelze odkazovat, nefungují jako rozhraní); navíc, protože jsou vložené funkce prakticky součástí třídy, nefunguje v určitých případech kontrola přepisování (*overridingu*).

Ačkoli je implementace systému vícenásobné dědičnosti tak, jak je například v C++, složitější, její implementace je uskutečnitelná, kód nezpomaluje a fakticky rozšiřuje možnosti jazyka. C++ model dědičnosti umí vše, co umí Javovský model, a ještě víc.

Jazyk NATI tedy bude umožňovat třídní dědičnost, a to způsobem podobným C++. Nicméně implementace systému třídní dědičnosti není primárním cílem projektu, a tak je možné, že v rámci bakalářské práce nebude plně implementován.

Dá se přemýšlet i o zavedení různých direktiv, které umožňují dále nastavovat tento systém – tedy například přidat možnost nastavení zamezení generování tabulky virtuálních metod, atp.

2.1.2 Automatická správa paměti – garbage collector

Spousta moderních jazyků (mezi nimi i D) obsahuje garbage collector (dále GC) v základu. Jeho zavedení nenabízí pouze výhody – programy mohou být pomalejší, GC zvyšuje nároky na CPU a paměť (těžko se zavádí v mikroprocesorech); navíc efektivní implementace GC je velice složitá.

Rozumným kompromisem se jeví být volitelné používání automatické správy paměti. NATI by měl umožňovat efektivní napsání GC jakožto knihovny; GC by tedy mohla být jedna ze základních knihoven jazyka. Tvorba této knihovny ale není předmětem tohoto projektu.

2.1.3 Implicitní konstantnost proměnných a propagace konstantnosti

Koncept konstantnosti proměnných byl zaveden jednak jako prvek kontroly při psaní kódu, jednak zvětšil potenciál kompilátorů při optimalizování kódu. Označení proměnné za konstantní ale u jazyku C++ (i D, Java, ...) vyžaduje ale napsání dalšího slova (modifikátoru *const* u C, C++ a D, *final* u Javy), a tak tuto praktiku (označovat všechno, co se dá, jako konstantní) spousta programátorů nepraktikuje, zčásti kvůli lenosti, zčásti kvůli zapomnětlivosti.

Některé jazyky (například Rust[2]) přišly s opačným přístupem – všechny proměnné jsou implicitně konstantní a programátor musí použít nějakou syntaktickou konstrukci k tomu, aby to změnil. NATI tento přístup také zavede. V rámci koherence syntaxe jazyka (která je rozvedena v dalších kapitolách tohoto textu), připadají v úvahu dvě možnosti:

1. Vytvoření dekorátoru v kontextu `typeWrapper`[3], nejlogičtěji `@mutable` nebo `@mut`
2. Vyčlenění operátoru; nejlepším kandidátem je suffixový operátor `Type!`, protože nemá žádnou standardní sémantiku, je nepoužitý a znak vykřičníku je intuitivně asociován s výstrahou, což je zase asociovatelné s mutabilitou.

Při designu NATI byla zvolena druhá možnost, především kvůli „upovídanosti“ kódu a ještě kvůli jednomu důvodu, který je popsán níže.

[1] https://dlang.org/spec/template-mixin.html
[2] https://doc.rust-lang.org/nightly/book/mutability.html
[3] Viz specifikace jazyka NATI, oddíl *Decoration contexts*

Propagace konstantnosti Jazyk D je navržen tak, že je-li ukazatel konstantní (nelze měnit adresu, na kterou ukazuje), je přístup k paměti, na kterou ukazuje, také konstantní. Není tedy možné mít konstantní ukazatel na nekonstantní paměť. Toto je zbytečné limitování; tranzitivní konstantnost se může hodit, ale měla by být volitelná (v NATI má toto opět potenciál pro řešení v dekorátorech).

V NATI tedy nebude vynucená propagace konstantnosti; můžeme mít konstantní ukazatel na nekonstantní data. Chceme-li mít nekonstantní ukazatel na nekonstantní data, musíme tedy specifikovat mutabilitu dvakrát. Ve dříve navržených dvou případech by to tedy vypadalo takto (syntaxe pro referenci je `Typ?`, viz 2.1.4):

1. `@mut ( @mut Typ )?`
2. `Typ!?!`

Alternativou by bylo třeba ještě zcela využít gramatiku C++, nicméně ta je všeobecně považována za velice matoucí. Ačkoli seskupení operátorů `?!?` (mutabilní reference na mutabilní hodnotu) může bezesporu v některých programátorech zpočátku vyvolat zděšení, první varianta je na tom ještě hůř. Po pochopení syntaktických pravidel, která jsou průhledná a jasná, se tato konstrukce ukáže být veskrze jednoduchá. Nicméně je třeba přiznat, že tento případ může pravděpodobně být nejhůře přijímaným prvek v jazyku.

Srovnání syntaxe C++, D a NATI

```
// C++
int a, b; // Mutable integer
const int c, d; // Const integer
int *e, *f; // Mutable pointer to mutable integer
const int *g, *h; // Mutable pointer to const integer
int * const i, * const j; // Const pointer to mutable integer
const int * const k, * const l; // Const pointer to const integer
```

```
// D
int a, b; // Mutable integer
const int c, d; // Const integer
int* e, f; // Mutable pointer to mutable integer
const( int )* g, h; // Mutable pointer to const integer
// const pointer to mutable integer not possible
const int* k, l; // Const pointer to const integer
```

```
// NATI
Int32! a, b; // Mutable integer
Int32 c, d; // Const integer
Int32!?! e, f; // Mutable pointer to mutable integer
Int32?! g, h; // Mutable pointer to const integer
Int32!? i, j; // Const pointer to mutable integer
Int32? k, l; // Const pointer to const integer
```

2.1.4 Reference, ukazatelé a jejich syntaxe

Způsob deklarace ukazatelů je v C++ i D problematický z hlediska parsování. Výraz `a * b` může totiž znamenat buď výraz násobení `a` krát `b`, stejně tak ale může znamenat deklaraci proměnné `b` typu ukazatel na `a` (obdobně i u reference). Očividným řešením je použití jiného znaku pro označování ukazatelů.

V NATI je tímto znakem otazník (`?`). V C++ a D je používán pouze v ternárním operátoru (`cond ? expr1 : expr2`), kteréhož funkčnost NATI obstarává jiným způsobem (viz 2.2.1); tím pádem je znak otazníku „postradatelný“.

Ukazatel vs. reference Typ ukazatel umožňující ukazatelovou aritmetiku je v dnešní době považován za potenciálně nebezpečný prvek, který by se měl užívat jen v nutných případech. Toto se řeší zavedením referencí, které v různých jazycích fungují mírně rozdílně, všeobecně se ale dá říci, že se jedná o ukazatele, které nepodporují ukazatelovou aritmetiku.

V C++ reference neumožňují měnit adresu odkazované paměti. V praxi ale programátor poměrně často potřebuje měnit hodnotu odkazu a musí se proto uchylovat k ukazatelům, které podporují ukazatelovou aritmetiku a pro přístup k odkazované hodnotě je třeba buď použít dereferenci (která je implementována prefixovým operátorem hvězdička a má tendenci znepřehledňovat kód) nebo speciální syntaktickou konstrukci (místo `x.y` `x->y`) pro přístup k prvkům odkazované hodnoty.

D k problému přistupuje takto:

1. Třídy jsou vždy předávané odkazem; adresa odkazované paměti se dá měnit, ukazatelová aritmetika není podporována (jako v Javě).

2. Existuje typ ukazatel, který pracuje s ukazatelovou aritmetikou. Pro přístup k odkazovanému prvku je třeba použít dereferenci (`*x`); k prvkům odkazované hodnoty lze použít klasické `x.y` (u C++ je třeba `x->y`). U dvojitého ukazatele je již potřeba použít dereferenci.

3. Existuje i reference podobná té v C++, nicméně ta se dá použít pouze v několika málo případech (například v parametrech a návratových typech funkcí)

2.2 Další změny oproti C++/D

2.2.1 Ternární operátor

Přílohy

Příloha A

Jak pracovat s touto šablonou

V této kapitole je uveden popis jednotlivých částí šablony, po kterém následuje stručný návod, jak s touto šablonou pracovat.

Jedná se o přechodnou verzi šablony. Nová verze bude zveřejněna do konce roku 2016 a bude navíc obsahovat nové pokyny ke správnému využití šablony, závazné pokyny k vypracování bakalářských a diplomových prací (rekapitulace pokynů, které jsou dostupné na webu) a nezávazná doporučení od vybraných vedoucích. Jediné soubory, které se v nové verzi změní, budou projekt-01-kapitoly-chapters.tex a projekt-30-prilohy-appendices.tex, jejichž obsah každý student vymaže a nahradí vlastním. Šablonu lze tedy bez problémů využít i v současné verzi.

Popis částí šablony

Po rozbalení šablony naleznete následující soubory a adresáře:

bib-styles Styly literatury (viz níže).

obrazky-figures Adresář pro Vaše obrázky. Nyní obsahuje placeholder.pdf (tzv. TODO obrázek, který lze použít jako pomůcku při tvorbě technické zprávy), který se s prací neodevzdává. Název adresáře je vhodné zkrátit, aby byl jen ve zvoleném jazyce.

template-fig Obrázky šablony (znak VUT).

fitthesis.cls Šablona (definice vzhledu).

Makefile Makefile pro překlad, počítání normostran, sbalení apod. (viz níže).

projekt-01-kapitoly-chapters.tex Soubor pro Váš text (obsah nahraďte).

projekt-20-literatura-bibliography.bib Seznam literatury (viz níže).

projekt-30-prilohy-appendices.tex Soubor pro přílohy (obsah nahraďte).

projekt.tex Hlavní soubor práce – definice formálních částí.

Výchozí styl literatury (czechiso) je od Ing. Martínka, přičemž anglická verze (englishiso) je jeho překladem s drobnými modifikacemi. Oproti normě jsou v něm určité odlišnosti, ale na FIT je dlouhodobě akceptován. Alternativně můžete využít styl od Ing. Radima Loskota

nebo od Ing. Radka Pyšného[1]. Alternativní styly obsahují určitá vylepšení, ale zatím nebyly řádně otestovány větším množstvím uživatelů. Lze je považovat za beta verze pro zájemce, kteří svoji práci chtějí mít dokonalou do detailů a neváhají si nastudovat detaily správného formátování citací, aby si mohli ověřit, že je vysázený výsledek v pořádku.

Makefile kromě překladu do PDF nabízí i další funkce:

- přejmenování souborů (viz níže),
- počítání normostran,
- spuštění vlny pro doplnění nezlomitelných mezer,
- sbalení výsledku pro odeslání vedoucímu ke kontrole (zkontrolujte, zda sbalí všechny Vámi přidané soubory, a případně doplňte).

Nezapomeňte, že vlna neřeší všechny nezlomitelné mezery. Vždy je třeba manuální kontrola, zda na konci řádku nezůstalo něco nevhodného – viz Internetová jazyková příručka[2].

Pozor na číslování stránek! Pokud má obsah 2 strany a na 2. jsou jen „Přílohy“ a „Seznam příloh“ (ale žádná příloha tam není), z nějakého důvodu se posune číslování stránek o 1 (obsah „nesedí“). Stejný efekt má, když je na 2. či 3. stránce obsahu jen „Literatura“ a je možné, že tohoto problému lze dosáhnout i jinak. Řešení je několik (od úpravy obsahu, přes nastavení počítadla až po sofistikovanější metody). **Před odevzdáním proto vždy překontrolujte číslování stran!**

Doporučený postup práce se šablonou

1. **Zkontrolujte, zda máte aktuální verzi šablony.** Máte-li šablonu z předchozího roku, na stránkách fakulty již může být novější verze šablony s aktualizovanými informacemi, opravenými chybami apod.

2. **Zvolte si jazyk**, ve kterém budete psát svoji technickou zprávu (česky, slovensky nebo anglicky) a svoji volbu konzultujte s vedoucím práce (nebyla-li dohodnuta předem). Pokud Vámi zvoleným jazykem technické zprávy není čeština, nastavte příslušný parametr šablony v souboru projekt.tex (např.: `documentclass[english]{fitthesis}` a přeložte prohlášení a poděkování do angličtiny či slovenštiny.

3. **Přejmenujte soubory.** Po rozbalení je v šabloně soubor projekt.tex. Pokud jej přeložíte, vznikne PDF s technickou zprávou pojmenované projekt.pdf. Když vedoucímu více studentů pošle projekt.pdf ke kontrole, musí je pracně přejmenovávat. Proto je vždy vhodné tento soubor přejmenovat tak, aby obsahoval Váš login a (případně zkrácené) téma práce. Vyhněte se však použití mezer, diakritiky a speciálních znaků. Vhodný název tedy může být např.: „xlogin00-Cisteni-a-extrakce-textu.tex“. K přejmenování můžete využít i přiložený Makefile:

   ```
   make rename NAME=xlogin00-Cisteni-a-extrakce-textu
   ```

[1] BP Ing. Radka Pyšného http://www.fit.vutbr.cz/study/DP/BP.php?id=7848
[2] Internetová jazyková příručka http://prirucka.ujc.cas.cz/?id=880

4. Vyplňte požadované položky v souboru, který byl původně pojmenován projekt.tex, tedy typ, rok (odevzdání), název práce, svoje jméno, ústav (dle zadání), tituly a jméno vedoucího, abstrakt, klíčová slova a další formální náležitosti.

5. Nahraďte obsah souborů s kapitolami práce, literaturou a přílohami obsahem svojí technické zprávy. Jednotlivé přílohy či kapitoly práce může být výhodné uložit do samostatných souborů – rozhodnete-li se pro toto řešení, je doporučeno zachovat konvenci pro názvy souborů, přičemž za číslem bude následovat název kapitoly.

6. Nepotřebujete-li přílohy, zakomentujte příslušnou část v projekt.tex a příslušný soubor vyprázdněte či smažte. Nesnažte se prosím vymyslet nějakou neúčelnou přílohu jen proto, aby daný soubor bylo čím naplnit. Vhodnou přílohou může být obsah přiloženého paměťového média.

7. Nascanované zadání uložte do souboru zadani.pdf a povolte jeho vložení do práce parametrem šablony v projekt.tex (`documentclass[zadani]{fitthesis}`).

8. Nechcete-li odkazy tisknout barevně (tedy červený obsah – bez konzultace s vedoucím nedoporučuji), budete pro tisk vytvářet druhé PDF s tím, že nastavíte parametr šablony pro tisk: (`documentclass[zadani,print]{fitthesis}`). Barevné logo se nesmí tisknout černobíle!

9. Vzor desek, do kterých bude práce vyvázána, si vygenerujte v informačním systému fakulty u zadání. Pro disertační práci lze zapnout parametrem v šabloně (více naleznete v souboru fitthesis.cls).

10. Nezapomeňte, že zdrojové soubory i (obě verze) PDF musíte odevzdat na CD či jiném médiu přiloženém k technické zprávě.

Pokyny pro oboustranný tisk

- Zapíná se parametrem šablony: `\documentclass[twoside]{fitthesis}`
- Po vytištění oboustranného listu zkontrolujte, zda je při prosvícení sazební obrazec na obou stranách na stejné pozici. Méně kvalitní tiskárny s duplexní jednotkou mají často posun o 1–3 mm. Toto může být u některých tiskáren řešitelné tak, že vytisknete nejprve liché stránky, pak je dáte do stejného zásobníku a vytisknete sudé.
- Za titulním listem, obsahem, literaturou, úvodním listem příloh, seznamem příloh a případnými dalšími seznamy je třeba nechat volnou stránku, aby následující část začínala na liché stránce (\cleardoublepage).
- Konečný výsledek je nutné pečlivě překontrolovat.

Užitečné nástroje

Následující seznam není výčtem všech využitelných nástrojů. Máte-li vyzkoušený osvědčený nástroj, neváhejte jej využít. Pokud však nevíte, který nástroj si zvolit, můžete zvážit některý z následujících:

MikTeX LaTeX pro Windows – distribuce s jednoduchou instalací a vynikající automatizací stahování balíčků.

TeXstudio Přenositelné opensource GUI pro LaTeX. Ctrl+klik umožňuje přepínat mezi zdrojovým textem a PDF. Má integrovanou kontrolu pravopisu, zvýraznění syntaxe apod. Pro jeho využití je nejprve potřeba nainstalovat MikTeX.

JabRef Pěkný a jednoduchý program v Javě pro správu souborů s bibliografií (literaturou). Není potřeba se nic učit – poskytuje jednoduché okno a formulář pro editaci položek.

InkScape Přenositelný opensource editor vektorové grafiky (SVG i PDF). Vynikající nástroj pro tvorbu obrázků do odborného textu. Jeho ovládnutí je obtížnější, ale výsledky stojí za to.

GIT Vynikající pro týmovou spolupráci na projektech, ale může výrazně pomoci i jednomu autorovi. Umožňuje jednoduché verzování, zálohování a přenášení mezi více počítači.

Overleaf Online nástroj pro LaTeX. Přímo zobrazuje náhled a umožňuje jednoduchou spolupráci (vedoucí může průběžně sledovat psaní práce), vyhledávání ve zdrojovém textu kliknutím do PDF, kontrolu pravopisu apod. Zdarma jej však lze využít pouze s určitými omezeními (někomu stačí na disertaci, jiný na ně může narazit i při psaní bakalářské práce) a pro dlouhé texty je pomalejší.

Užitečné balíčky pro LaTeX

Studenti při sazbě textu často řeší stejné problémy. Některé z nich lze vyřešit následujícími balíčky pro LaTeX:

- `amsmath` – rozšířené možnosti sazby rovnic,
- `float, afterpage, placeins` – úprava umístění obrázků,
- `fancyvrb, alltt` – úpravy vlastností prostředí Verbatim,
- `makecell` – rozšíření možností tabulek,
- `pdflscape, rotating` – natočení stránky o 90 stupňů (pro obrázek či tabulku),
- `hyphenat` – úpravy dělení slov,
- `picture, epic, eepic` – přímé kreslení obrázků.

Některé balíčky jsou využity přímo v šabloně (v dolní části souboru fitthesis.cls). Nahlédnutí do jejich dokumentace může být rovněž užitečné.

Sloupec tabulky zarovnaný vlevo s pevnou šířkou je v šabloně definovaný „L“ (používá se jako „p“).