# Skriptovací jazyk pro zpracování obrazu

FIT, VUT v brně

3. prosince 2007

## 1 Úvod

V tomto dokumentu bude podrobněji popsán jazyk a jeho překladač, navržený za účelem zpracování obrazu na osobním počítači.

## 2 Motivace k tvorbě jazyka

Jazyk byl navržen pro jednoduché zpracování obrazu na osobním počítači. Původní motivace vytvořit jazyk pro zpracování obrazu vznikla v rámci zadání diplomové práce. Problematika řešená v diplomové práci se týkala jen návrhu a překladu jazyka a už ne příliš samotným vykonáváním příkazů popsaných v jazyce (tj. interpretem jazyka).

V rámci projektu na ústavu počítačové grafiky a multimédií bylo na jazyku provedeno několik uprav a byl vytvořen základní interpret zdrojového kódu. Tento interpret umožňuje spouštět skripty zapsané v jazyce na osobním počítači a provádět demonstrativní úpravy obrazu. Interpret jazyka byl navržen způsobem, který umožňuje snadné přidávání vestavěných struktur, funkcí a metod, bez potřeby ovlivnění samotné syntaxe jazyka.

Z důvodu nevhodnosti navrženého jazyka pro využití na DSP procesorech, bude vyvinuta další verze, která by měla tento problém řešit už v rámci návrhu jazyka. Jedná se hlavně o návrh způsobu zpracování více vláken programu popsaného skriptem.

## 3 Použití programu (interpretu)

Program se spouští zadáním jména spustitelného souboru programu, kterému se jako první argument předá zdrojový soubor obsahující kód skriptu. Všechny ostatní argumenty předané programu realizujícího interpret budou předány skriptu jako argumenty programu. Příklad spouštění programu následuje.

```
./interpret process_img.src input.bmp output.bmp
```

Kde soubor `process_img.src` obsahuje program zapsaný ve skriptovacím jazyce, a názvy souborů `input.bmp` a `output.bmp` budou předány spouštěnému skriptu jako parametry.

# 4 Popis jazyka

Jazyk je netypovaný, strukturovaný a umožňuje definovat uživatelské funkce a struktury (bez dědičnosti). U každé struktury je dále možné definovat její proměnné (prvky struktury) a metody. Jazyk obsahuje několik vestavěných struktur, včetně jejich metod, pomocí kterých je možné s těmito strukturami manipulovat. Všechny datové typy jazyka (včetně vestavěných typů) jsou popsány výše zmíněnými strukturami.

Netypované proměnné v jazyce umožňují výběr prováděné operace na základě aktuálního typu proměnné. Nevýhodou tohoto přístupu je složitější režie interpretu při spouštění kódu.

Každá uživatelem definovaná funkce, struktura a blok příkazů má podobně jako v jazyce `C`, přiřazen svůj jmenný prostor proměnných. Každý identifikátor `str` v následujícím kódu popisuje jinou proměnnou.

```
fun (a,b,c) {
   str = a;
}

struct struct_s {
   a,b,str;
}

str = "retezec\n";
```

Na proměnné jednotlivých instancí struktur je možné se odkazovat stejně jako v `C` využitím tečkové notace, např: `str.x`. Kromě lokálních jmenných prostorů existuje i jeden globální, shrnující pod sebou všechny proměnné použité mimo jakoukoli funkci, nebo strukturu.

Mimo vestavěných struktur jsou v jazyce (interpretu) definovány některé vestavěné globální funkce, globální proměnné a konstanty.

Skript předaný interpretu se začíná vykonávat na první instrukci (výrazu) vložené v globálním prostoru.

# 5 Syntaxe jazyka

Základní syntaxe jazyka, do které spadají zápisy výrazu, volání funkcí a metod, zápis podmínek a cyklů je převzata z jazyků `C` a `C++`.

## 5.1 Výraz

Výraz se zapisuje podobně jako v jazyce `C`. Skládá se z posloupnosti proměnných a konstant, které se kombinují pomocí volání funkcí, metod a operátorů, které s nimi pracují. Po vyhodnocení jednoho výrazu získáme právě jednu hodnotu. Příklad jednoduchého výrazu následuje.

```
a = b * (c + d)
```

## 5.2 Podmíněné větvení

Obdobně jako v jazyce `C` je větvení programu na základě podmínky definováno klíčovým slovem `if` a samotná podmínka je zapsána v kulatých závorkách. Volitelně je možné definovat kód, který se provede pokud podmínka není splněná, tento se zapisuje za klíčové slovo `else`.

```
if (<expression>) <command>
else <command>
```

Obdobně jako v jazyce `C` je možné místo jednoho příkazu použít blok příkazu uzavřený ve složených závorkách.

## 5.3 Cykly

Jazyk podporuje dva základní cykly, kterými jsou cyklus s podmínkou na začátku, a cyklus s podmínkou na konci. Stejně jako v jazyce `C` jsou tyto cykly definovány následujícím zápisem.

```
while (<expression>) <command>

do <command> while (<expression>);
```

## 5.4 Funkce

Pro definici funkce se používá klíčové slovo `fun`, za nímž se definuje jméno funkce, seznam jejích parametrů a mezi složenými závorkami samotný kód funkce. Parametry funkce před nimiž se nachází znak `&` se předávají odkazem místo kopie.

```
fun <name> (<parm>,<parm>,...)
{
   <command>
   ...
   <command>
}
```

### 5.5 Struktury

Struktury ve skriptu se definuji pomoci klíčového slova `struct` za nímž následuje jméno struktury a dále ve složených závorkách jsou definovány jednotlivé datové prvky struktury (jména jejích proměnných) a jednotlivé metody struktury, definované stejně jako globální funkce.

```
struct <name>
{
   <var>,<var>,...;

   fun <name>(<parm>,<parm>,...)
   {
      <command>
      ...
      <command>
   }
}
```

Ve vytvářené struktuře je možné jako metody definovat i jednotlivé operátory rozpoznávané jazykem, ovšem tato definice není tak jednoduchá jako v jazyce `C`, každý operátor je nutné definovat jeho názvem. Takže například operátor `+=` je nutné zapsat následujícím způsobem.

```
fun operator_binary_plus_equal(<parameter>)
{
   <command>
   ...
   <command>
}
```

Jména všech vestavěných funkcí je možné nalézt ve zdrojovém souboru `build_in.h`.

### 5.6 Parametry programu

K jednotlivým parametrů předaným programu se přistupuje stejně jako v jazyce `C` přes proměnné `argc` a `argv`. Kde `argc` udává počet parametrů předaných skriptu a proměnná `argv` obsahuje v poli objektů jednotlivé řetězce argumentů.

## 6 Vlákna

Nové vlákno programu se definuje klíčovým slovem `thread`. Za tímto klíčovým slovem se v kulatých závorkách specifikují parametry vlákna, které se mají

zkopírovat z proměnných mimo vlákno a poté se ve složených závorkách definuje vykonávané tělo vlákna. Tělo vlákna je obyčejný kód, takže je v něm možné volat funkce, vytvářet objekty apod. Samotná definice vlákna je výraz, který vrací při úspěšném vytvoření vlákna jeho identifikaci a v jiném případě identifikátor chyby. Protože je definice vlákna součásti výrazu v původním vlákně, je nutné za pravou závorku umístit středník, ukončující výraz. Následuje příklad vytvoření vlákna.

```
thread_id = thread()
{
   idx = 0;
   do {
      ...
   } while(++idx < 1000);
};
```

Vytvořené vlákna je nutné znovu spojit (`join`) s vláknem, které je vytvořilo, nebo úplně uvolnit (`detach`) z kontroly hlavního vlákna.

# 7 Struktury a vestavěné struktury

Struktura v jazyce popisuje seznam proměnných (blíže nespecifikovaného datového typu) a seznam operací definovaných pro práci s touto strukturou. Proměnné se nerozlišují na veřejné a neveřejné, všechny proměnné struktury jsou veřejné.

Vestavěné struktury neobsahují žádné datové proměnné, a tvoří základní datové elementy jazyka. Také všechny metody a operátory nad objekty vestavěných struktur jsou definovány interpretem a tj. nejsou definovány jako kód v jazyce.

Instance jednotlivých struktur se vytvářejí pomocí klíčového slova `new` a to následujícím zápisem.

```
str = new string_s
```

Nejsou definovány konstruktory a destruktory, výše uvedený zápis vytvoří pouze nový objekt, jehož všechny proměnné jsou nastaveny jako objekty struktury `blank_s` (viz. seznam vestavěných struktur).

Následuje seznam vestavěných struktur s jejich krátkým popisem.

- `blank_s` - popisuje prázdnou proměnnou, na tuto hodnotu je nastaven typ každé proměnné, do které nebyla zatím přiřazena žádná jiná hodnota.
- `char_s` - struktura, jejíž objekt popisuje jeden znak, stejným způsobem jako datový typ `char` v jazyce `C`.

- `short_s` - struktura popisující celé číslo na dvou bajtech, stejně jako datový typ `short` v jazyce `C`.
- `int_s` - struktura popisující celé číslo na čtyřech bajtech, stejně jako datový typ `int` v jazyce `C`.
- `double_s` - struktura popisující číslo s plovoucí řádovou čárkou na osmi bajtech, stejně jako typ `double` v jazyce `C`.
- `color_s` - struktura definující datový typ určený k popisu barvy pomocí tří barevných složek `red`, `green` a `blue`.
- `thread_s` - struktura definující objekty popisující jednotlivá vlákna programu.
- `string_s` - struktura popisující řetězec znaků, včetně základních operací definovaných nad tímto řetězcem.
- `local_img_s` - struktura určená k popisu obrazu složeného ze tří základních složek `red`, `green`, `blue` a definující základní operace nad tímto obrazem.
- `char_array_s` - struktura definující a popisující pole znaků, včetně základních operací nad tímto polem.
- `int_array_s` - struktura popisující pole celých čísel (`int`) a základní operace aplikovatelné na tuto strukturu.
- `double_array_s` - obdobně jako u znaku a celých čísel definuje tato struktura pole čísel s plovoucí řádovou čárkou (`double`).
- `array_s` - struktura definující pole obecných objektu, do tohoto pole je možné vložit objekt jakékoli struktury, včetně struktury definované uživatelem.
- `mutex_s` - struktura objektů popisujících mutexy používané k synchronizaci jednotlivých vláken programu.
- `condition_s` - obdobně jako u mutexů tato struktura popisuje objekty používané k synchronizaci vláken programu.

Podrobnější informace o jednotlivých vestavěných strukturách a jejich metodách se nacházejí v dokumentaci jazyka.

# 8 Jednotlivé části zdrojového textu

Zdrojový text je rozdělen na několik součástí, určených jejich funkcí v rámci překladače, nebo interpretu.

## 8.1 Soubory `main.h` a `main.cc`

Tyto soubory obsahují hlavní funkci programu, ve které začíná vykonávání programu. Tato funkce se postará o načtení zdrojového textového souboru, vytvoří objekt překladače a interpretu, přeloží zdrojový kód pomocí překladače a spustí hlavní vlákno skriptu na první globální instrukci v programu. Po ukončení vykonávání skriptu odstraní jeho globální proměnné a konstanty z paměti, a poté odstraní samotný interpret. Soubory `main.*` dále obsahují dvě pomocné funkce, kde první z nich zapisuje na výstup seznam symbolů interpretu po překladu skriptu a druhá zapisuje na výstup vygenerovaný kód, který je následně spouštěn interpretem.

## 8.2 Soubory `basic.h` a `basic.cc`

V těchto souborech jsou obsaženy základní konstanty, používané ve všech dalších částech programu. Dále se v nich nachází kód určený pro kontrolu přidělování a uvolňování paměti, který je možné při překladu vynechat zakomentováním definice `MEM_CHECK`. Poslední důležitou části popsanou v těchto souborech jsou makra a funkce pro test a výpis chyb detekovaných při překladu a vykonávání programu. V souboru `basic.h` je seznam konstant identifikující jednotlivé chybové zprávy.

## 8.3 Soubory `struct.h` a `struct.cc`

Soubory `struct.*` obsahují definice a funkce jednodušších struktur používaných v rámci celého programu. Mezi tyto struktury patří například `string_s` popisující znakový řetězec a `mutex_s` zobecňující mutex sloužící k synchronizaci vláken. Dále jsou v těchto souborech definovány generované struktury také dále používané na více místech ve zdrojích programu. Do této skupiny patří například dynamické pole většiny základních typů, atd.

## 8.4 Soubory `image.h` a `image.cc`

V těchto souborech jsou definovány dvě struktury, určené pro práci s obrazem popsaným třemi složkami RGB na obdélníkovém výřezu o nějaké velikosti. První z těchto struktur `img_3ub_s` popisuje samotný obraz, tj. jeho velikost, paměť obsahující hodnoty jednotlivých pixelů, a operace, které je možné na tento obraz aplikovat. Druhá z těchto struktur slouží jako obal k první. Pro každou funkci první struktury definuje obalovací funkci, která spravuje mutexy jednotlivých obrazů, aby nebylo možné ve vícevláknovém programu operovat dvěma vlákny nad jedním obrazem současně.

## 8.5 Soubory `parse_actions.h` a `parse_actions.cc`

Tyto dva soubory jsou klíčové pro překlad zdrojového souboru obsahujícího kód skriptu do posloupnosti instrukcí mezikódu spouštěného v interpretu.

Jsou v nich obsaženy funkce, které se volají na základě redukcí pravidel v překladači, který provádí analýzu zdola nahoru. Tyto funkce na základě pořadí svého spouštění generují cílový kód, a různé pomocné proměnné, jako například seznam symbolů, funkcí, jmenné prostory proměnných atd. Pro pochopení významu jednotlivých funkcí je nutné znát pořadí jejich volání v závislosti na tvaru rozkládaného výrazu.

## 8.6 Soubory `run_actions.h` a `run_actions.cc`

Tyto soubory obsahují seznam funkcí, které se volají při interpretaci překladačem vygenerovaného kódu. Každé instrukci odpovídá právě jedna funkce. Interpretace se takto redukuje na volání těchto funkcí na základě indexu určeného instrukcí. Smysluplnost pořadí instrukcí má na starosti překladač a jednotlivé instrukční funkce spolupracují pomocí zásobníku, na jehož vrcholu se nacházejí indexy aktuálně zpracovávaných hodnot.

## 8.7 Soubory `build_in.h` a `build_in.cc`

Jednotlivé vestavěné struktury, funkce a operátory definované v interpretu se nacházejí právě v těchto dvou souborech. Pokud vznikne potřeba přidat další vestavěnou metodu například ke struktuře `string_s` stačí přidat název této funkce do seznamu jmen, přidat její konstantní identifikátor do seznamu funkcí této struktury a zapsat kód nové funkce. V souboru `build_in.cc` se nacházejí jednotlivé vestavěné funkce spouštěné při samotné interpretaci kódu.

## 8.8 Soubory `script_parser.h`, `value_location.cc` a `script_parser.cc`

První soubor z této trojice je hlavičkový soubor společný oběma zbývajícím souborům, a obsahuje definici konstant a funkcí používaných pro překlad skriptu a ukládání hodnot běžícího skriptu. Je v něm definováno konstantní pole popisující rozkladovou tabulku používanou k realizaci analýzy zdola nahoru, konstanty identifikující typ proměnné, modifikátory spravovaných hodnot, identifikátory konstant interpretu, identifikátory instrukcí generovaného kódu a další.

Soubor `value_location.cc` obsahuje metody struktury spravující všechny lokální, dočasné, globální a konstantní hodnoty a proměnné.

V posledním souboru `script_parser.cc` se nalézají základní funkce pro rozklad a interpretaci skriptu. Funkce realizující lexikální analýzu se jmenuje `recognize_terminal` a je realizována konečným automatem. Funkce realizující samotný syntaktický rozklad se jmenuje `parse_script` a provádí rozklad zdola nahoru definovaný rozkladovou tabulkou. Při každé redukci podle nějakého pravidla, kromě takových redukcí, ke kterým je přiřazeno nulové pravidlo je volána odpovídající funkce ze souboru `parse_actions.cc`. Před

samotným překladem je nutné zavolat funkci `initialize_parser` která připraví překladač a nastaví jeho dynamické proměnné na počáteční hodnoty.

Ve funkcích `run_thread` a `run_main_thread` se provádí samotná interpretace kódu. Tyto funkce na základě kódu v jednotlivých blocích volají funkce ze souboru `run_actions.cc`, a tím vykonávají skript.

# 9 Lexikální a syntaktický analyzátor

Základní části překladače, které tvoří lexikální a syntaktický analyzátor se generují automaticky, na základě popisu jazyka. Soubor popisující diskutovaný skriptovací jazyk se nachází v adresáři `script_syntax` a jmenuje se `script_rules.txt`.

V tomto souboru je obsažen popis tvaru jednotlivých terminálních symbolů, dále seznam neterminálních symbolů a popis jednotlivých rozkladových pravidel, tvořících základ syntaktické analýzy.

# 10 Závěr

Seznam souborů v rámci autorizovaného softwaru se nachází v souboru `content.txt`.