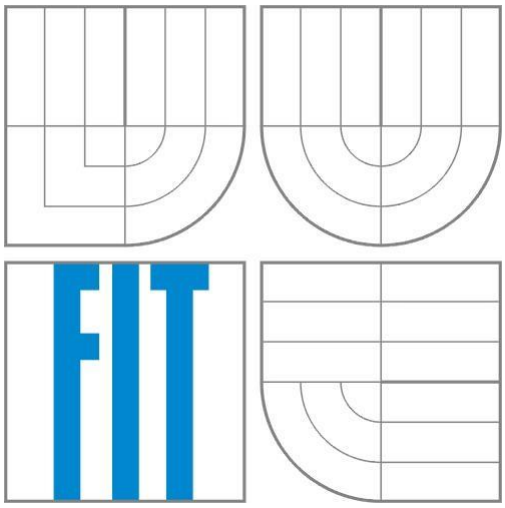


POKYNY PRO OBHAJOBU PROJEKTU

INTERPRET IMPERATIVNÍHO JAZYKA IFJ13

Tým 022, varianta a/3/II

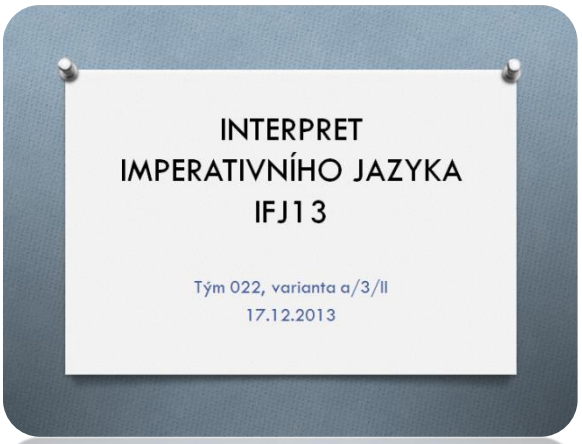
17. 12. 2013

Vedoucí týmu:
Pavel Beran (xberan33): 20%

Členové týmu:
Tomáš Vojtěch (xvojte02): 20%
Martin Fajčík (xfajci00): 20%
Martin Kalábek (xkalab06): 20%
Ondřej Soudek (xsoude01): 20%

Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1



Dobré dopoledne, vážená komise. Mé jméno je *Jméno Příjmení* a rád bych Vás v následujících minutách seznámil s naší implementací projektu do předmětů IFJ a IAL.

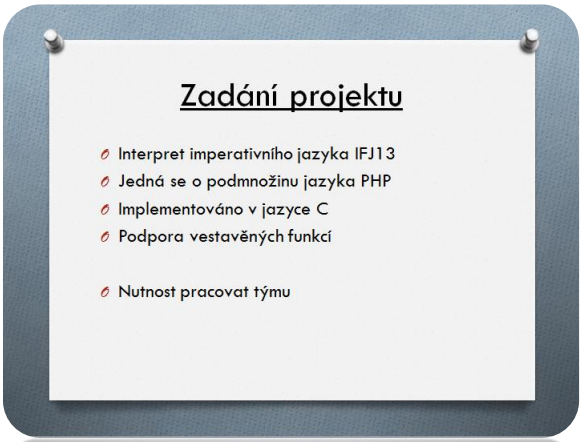
2

Nejprve bych Vám rád představil jednotlivé členy našeho řešitelského týmu.

Jimi jsou: - Pavel Beran, který vývoj interpretu vedl
- Tomáš Vojtěch,
- Martin Fajčík,
- Martin Kalábek,
a Ondřej Soudek

Prezentující zde představí pouze zbylé členy týmu, neboť sebe již představil dříve.

3

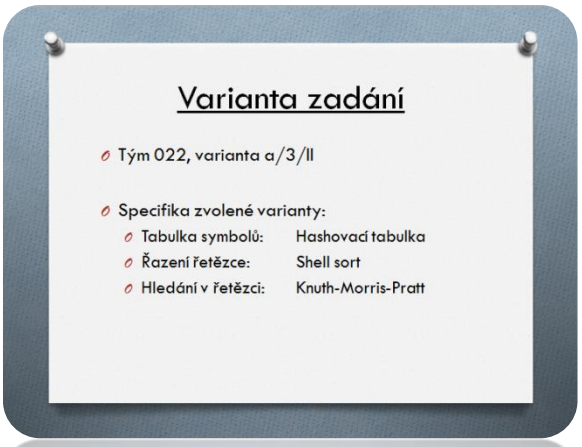


Podstatou zadaného projektu byla implementace interpretu imperativního programovacího jazyka.

Jednalo se o jazyk IFJ13, jenž je podmnožinou PHP a byl vytvořen právě pro účely tohoto projektu.

Projekt bylo nutné implementovat v jazyce C a to v 4-5ti členném týmu. Výsledný interpret měl podporovat také několik vestavěných funkcí.

4

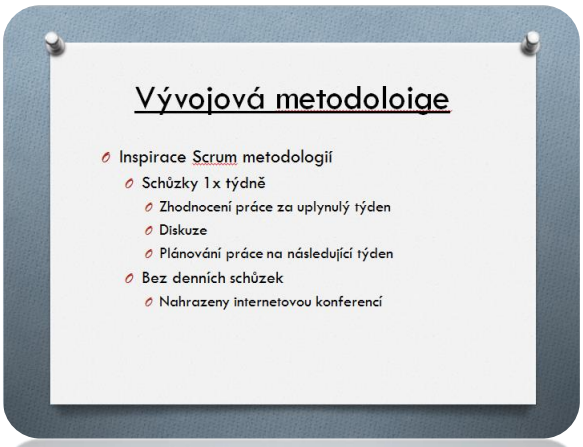


Zadání mělo mnoho variant a každý tým si mohl zaregistrovat dle svého výběru jednu variantu.

Zvolili jsme po společné dohodě variantu, která nám specifikovala implementaci:

- tabulky symbolů pomocí hashovací tabulky
- řazení řetězců pomocí Shell sort algoritmu
- a vyhledávání podřetězců v řetězcích pomocí Knuth-Morris-Pratt algoritmu

5

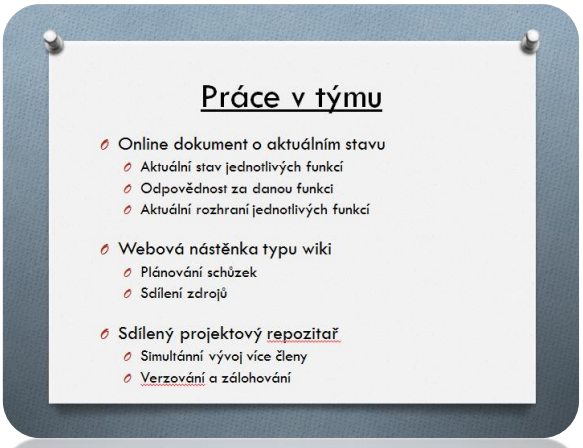


Jelikož se jednalo o poměrně rozsáhlý týmový projekt, bylo nezbytné zvolit metodologii vývoje a zavést určitá pravidla, aby se předešlo komunikačním problémům.

Naše metodologie se do jisté míry inspirovala osvědčenou Scrum metodologií s tím, že jsme ji modifikovali pro naše potřeby:

- Naše vývojové cykly byly kratší a týmové schůzky se tak konaly 1x týdně. Na schůzce se vždy probrala práce provedená za uplynulý týden, diskutovalo se vývoji a aktuálně implementované problematice. Následně se naplánovala práce na následující týden.
- Denní schůzky jsme zcela vypustili, neboť by spotřebovávali příliš mnoho času. Nahradili jsme je online konferencí.

6



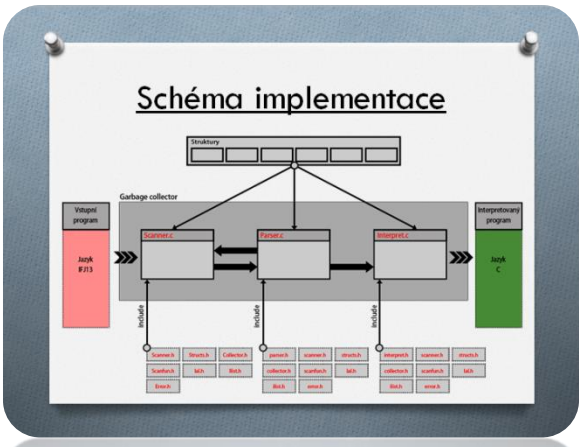
Týmový charakter projektu s sebou nesl několik komplikací, se kterými se bylo třeba vypořádat.

Jednou z takových komplikací bylo rozdělení práce. To jsme řešili tvorbou online dokumentu obsahujícího aktuální stav jednotlivých funkcí, odpovědnost za danou funkci a aktuální rozhraní. V pozdější fázi vývoje se tým rozdělil na dvě vývojové skupiny, aby byl vývoj urychlen.

Dále bylo třeba zajistit předávání informací mezi členy týmu, čehož jsme dosáhli pomocí webové nástěnky.

V neposlední řadě bylo třeba zajistit možnost současné práce více členů. Za tímto účelem jsme vytvořili sdílený projektový repozitář, který zároveň sloužil jako verzovací a zálohovací systém.

7



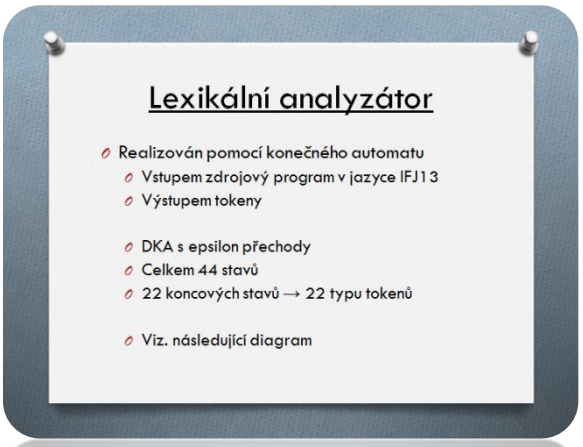
Pro ilustraci naší implementace jsme si připravili názorné schéma. Program lze rozčlenit na tři podprogramy. Lexikální analyzátor (na obrázku znázorněn jako scanner), Syntaktický analyzátor (znázorněn jako parser) a interpret trojadresného kódu.

Ze schématu plyne, že komunikace mezi parserem a scannerem probíhá v obou směrech, zatím co mezi parserem a interpretem již pouze v jednom směru.

Jak schéma naznačuje, jednotlivé části používají společné implementace struktur.

Dále je vidět, že celý interpret je zastřešen vlastní implementací správy paměti zabraňující únikům paměti

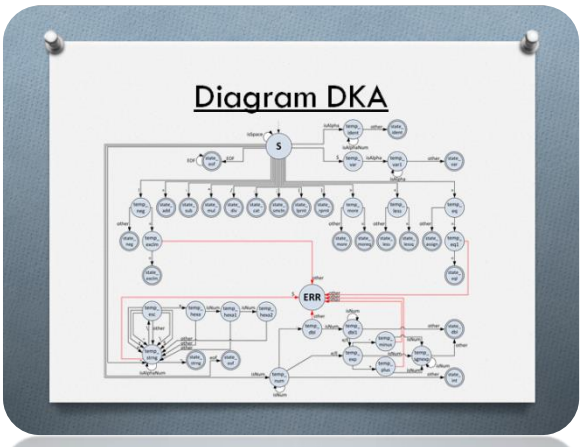
8



První částí je tedy lexikální analyzátor. Je realizován pomocí deterministického konečného automatu, jehož diagram uvedu v zápětí.

Vstupem lexikálního analyzátoru je zdrojový program v jazyce IFJ13 a výstupem jsou jednotlivé tokeny.

9



Nyní je již vidět slíbený diagram konečného automatu. Lze z něj vyčíst, že automat disponuje celkem 44 stavy, z nichž 22 je koncových.

Jinými slovy automat umí rozpoznat 22 typů tokenů.

10

Syntaktický analyzátor

- Kombinace dvou metod
  - Rekurzivní sestup
    - Pro obecnou analýzu
    - Pomocí LL gramatiky
  - Precedenční analýza
    - Pro zpracování výrazů
    - Pomocí precedenční tabulky
- Realizuje část sémantické analýzy

Druhou částí implementace je syntaktický analyzátor, který využívá dvou metod a sice:

- Metody shora dolu neboli rekurzivního sestupu pro obecnou analýzu, za využití LL gramatiky.

a

- Metody zdola nahoru neboli Precedenční analýzy, s využitím precedenční tabulky pro zpracování výrazů.

Na tomto místě se také realizuje část sémantické analýzy.

11

Interpret vnitřního kódu

- Realizován rekurzivně
  - Vhodné pro parametry a návratové hodnoty
  - Každá instance má svou pomocnou TS
- Provádí 3AC
  - 28 typů instrukcí
- Vstupem je seznam instrukcí
- Výstupem interpretace programu v jazyce C
- Realizuje většinu sémantické analýzy

Poslední implementovanou částí je interpret tříadresného kódu, který jsme implementovali rekurzivně. Rozhodli jsme se tak z důvodu snazšího předávání parametrů a návratových hodnot. Pro každou instanci interpretu se vytváří nová pomocná tabulka symbolů, pomocí které se předávají parametry.

Důležité bylo navrhnout vhodnou instrukční sadu. Ve výsledku se naše sada skládá z 28 typů instrukcí.

Na tomto místě se realizuje zbývající část sémantické analýzy, která nemohla být vzhledem k dynamickému typování jazyka IFJ13 provedena dříve.

12

Děkujeme Vám za pozornost.