

# Tým xhricma00 varianta vv-BVS

**Marek Hric**  
xhricma00

Mikuláš Lešiga  
xlesigm00

Roman Andraščík  
xandrar00

Adam Veselý  
xvesela00

December 4, 2024

## Rozdelenie bodov

xhricma00: 25%

xlesigm00: 25%

xandrar00: 25%

xvesela00: 25%

## Rozšírenia

ORELSE

UNREACHABLE

BOOLTHEN

FOR

WHILE

FUNEXP

## Rozdelenie prace :

---

Marek Hric :

- Lexikálna analýza
- Generovanie kodu

Mikuláš Lešiga :

- Syntaktická analýza
- Semantická analýza
- LL Gramatika
- Tabulky

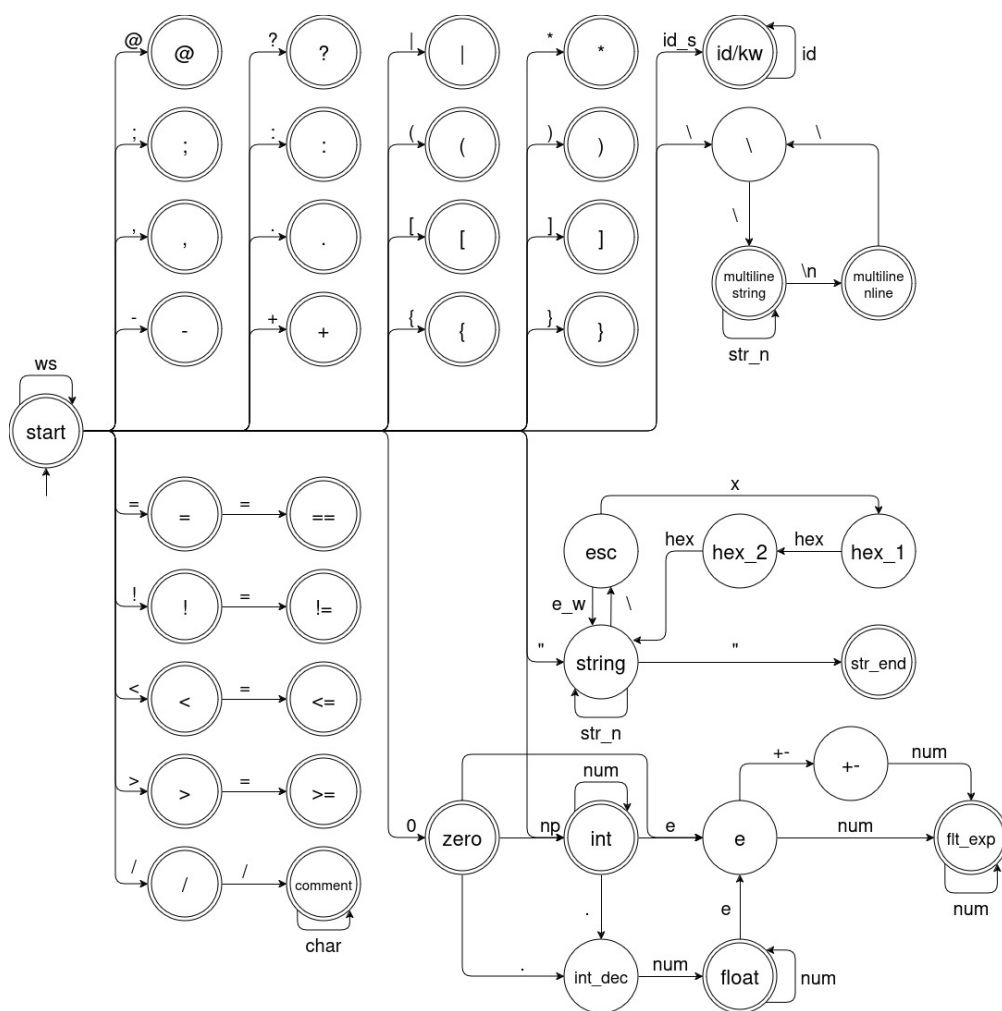
Roman Andraščík :

- Testy
- Pomocné funkcie
- Symtable
- Dokumentácia

Adam Veselý :

- Syntaktická analýza
- Semantická analýza
- Dátové štruktúry

## Diagram konečného automatu :



## LL-gramatika :

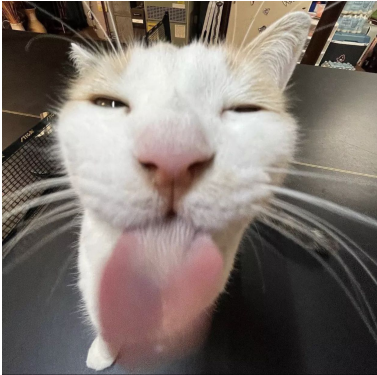
---

1.  $\langle prog \rangle \rightarrow \langle prolog \rangle \langle function\_def \rangle \langle end \rangle$
2.  $\langle prolog \rangle \rightarrow constID = @import(\langle expression \rangle);$
3.  $\langle end \rangle \rightarrow EOF$
4.  $\langle function\_def \rangle \rightarrow pub\ fn\ ID(\langle param\_list \rangle) \langle function\_type \rangle \langle block \rangle \langle function\_def \rangle$
5.  $\langle function\_def \rangle \rightarrow \varepsilon$
6.  $\langle param\_list \rangle \rightarrow ID : \langle type\_complete \rangle \langle comma\_par\_found \rangle$
7.  $\langle param\_list \rangle \rightarrow \varepsilon$
8.  $\langle comma\_par\_found \rangle \rightarrow , \langle param\_list \rangle$
9.  $\langle comma\_par\_found \rangle \rightarrow \varepsilon$
10.  $\langle block \rangle \rightarrow \{ \langle statement \rangle \}$
11.  $\langle block \rangle \rightarrow \langle single\_statement \rangle$
12.  $\langle statement \rangle \rightarrow \langle var\_declaration \rangle \langle statement \rangle$
13.  $\langle statement \rangle \rightarrow ID \langle ID\_found \rangle \langle statement \rangle$
14.  $\langle statement \rangle \rightarrow \langle if\_statement \rangle \langle statement \rangle$
15.  $\langle statement \rangle \rightarrow \langle for\_loop \rangle \langle statement \rangle$
16.  $\langle statement \rangle \rightarrow \langle while\_loop \rangle \langle statement \rangle$
17.  $\langle statement \rangle \rightarrow \langle return\_statement \rangle \langle statement \rangle$
18.  $\langle statement \rangle \rightarrow \langle break \rangle \langle statemet \rangle$
19.  $\langle statement \rangle \rightarrow \langle continue \rangle \langle statemet \rangle$
20.  $\langle statement \rangle \rightarrow \varepsilon$
21.  $\langle ID\_found \rangle \rightarrow = \langle asgn\_found \rangle;$
22.  $\langle ID\_found \rangle \rightarrow (\langle expression\_list \rangle);$
23.  $\langle ID\_found \rangle \rightarrow : \langle while\_loop \rangle$
24.  $\langle var\_declaration \rangle \rightarrow const\ ID : \langle type\_complete \rangle = \langle asgn\_found \rangle;$
25.  $\langle var\_declaration \rangle \rightarrow var\ ID : \langle type\_complete \rangle = \langle asgn\_found \rangle;$

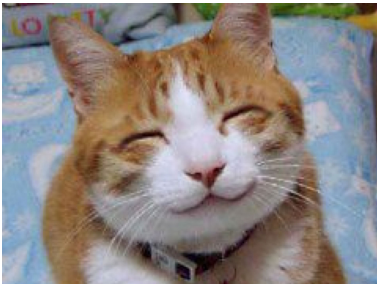
26.  $\langle \text{if\_statement} \rangle \rightarrow \text{if}(\langle \text{expression} \rangle) \langle \text{if\_found} \rangle$
27.  $\langle \text{if\_found} \rangle \rightarrow \langle \text{optional\_value} \rangle \langle \text{block} \rangle \langle \text{else\_statement} \rangle$
28.  $\langle \text{if\_found} \rangle \rightarrow \langle \text{single\_statement} \rangle \text{else} \langle \text{single\_statement} \rangle$
29.  $\langle \text{else\_statement} \rangle \rightarrow \text{else} \langle \text{block} \rangle$
30.  $\langle \text{else\_statement} \rangle \rightarrow \varepsilon$
31.  $\langle \text{optional\_value} \rangle \rightarrow |ID|$
32.  $\langle \text{optional\_value} \rangle \rightarrow \varepsilon$
33.  $\langle \text{while\_loop} \rangle \rightarrow \text{while}(\langle \text{expression} \rangle) \langle \text{optional\_value} \rangle \langle \text{optional\_statements} \rangle \langle \text{block} \rangle \langle \text{else\_statement} \rangle$
34.  $\langle \text{return\_statement} \rangle \rightarrow \text{return} \langle \text{expression} \rangle;$
35.  $\langle \text{expression\_list} \rangle \rightarrow \langle \text{expression} \rangle \langle \text{comma\_expr\_found} \rangle$
36.  $\langle \text{expression\_list} \rangle \rightarrow \varepsilon$
37.  $\langle \text{comma\_expr\_found} \rangle \rightarrow , \langle \text{expression\_list} \rangle$
38.  $\langle \text{comma\_expr\_found} \rangle \rightarrow \varepsilon$
39.  $\langle \text{type} \rangle \rightarrow i32$
40.  $\langle \text{type} \rangle \rightarrow f64$
41.  $\langle \text{type} \rangle \rightarrow []u8$
42.  $\langle \text{type} \rangle \rightarrow \text{bool}$
43.  $\langle \text{for\_loop} \rangle \rightarrow \text{for}(\langle \text{expression} \rangle) \langle \text{optional\_value} \rangle \langle \text{block} \rangle$
44.  $\langle \text{optional\_statements} \rangle \rightarrow \text{:} \langle \text{block} \rangle$
45.  $\langle \text{optional\_statements} \rangle \rightarrow \varepsilon$
46.  $\langle \text{type\_complete} \rangle \rightarrow \langle \text{question\_mark} \rangle \langle \text{type} \rangle$
47.  $\langle \text{question\_mark} \rangle \rightarrow ?$
48.  $\langle \text{question\_mark} \rangle \rightarrow \varepsilon$
49.  $\langle \text{single\_statement} \rangle \rightarrow \langle \text{var\_declaration} \rangle$
50.  $\langle \text{single\_statement} \rangle \rightarrow ID \langle ID\_found \rangle$
51.  $\langle \text{single\_statement} \rangle \rightarrow \langle \text{if\_statement} \rangle$

- 52.  $\langle \text{single\_statement} \rangle \rightarrow \langle \text{for\_loop} \rangle$
- 53.  $\langle \text{single\_statement} \rangle \rightarrow \langle \text{while\_loop} \rangle$
- 54.  $\langle \text{single\_statement} \rangle \rightarrow \langle \text{return\_statement} \rangle$
- 55.  $\langle \text{function\_type} \rangle \rightarrow \langle \text{type} \rangle$
- 56.  $\langle \text{function\_type} \rangle \rightarrow \text{void}$
- 57.  $\langle \text{asgn\_found} \rangle \rightarrow \langle \text{expression} \rangle$
- 58.  $\langle \text{asgn\_found} \rangle \rightarrow \text{if}(\langle \text{expression} \rangle) \langle \text{expression} \rangle \text{ else } \langle \text{expression} \rangle$
- 59.  $\langle \text{continue} \rangle \rightarrow \text{continue};$
- 60.  $\langle \text{break} \rangle \rightarrow \text{break};$

## LL-tabulka :



## Precendecna-tabulka :



## Lexikálna analýza

---

Riešenie lexikálnej analýzy sme začali vytvorením diagramu deterministického konečného automatu. Následne sme na jeho základe začali vypracovávať implementáciu. Implementácia sa nachádza v súbore *scanner.c*, ktorý pracuje s tokenmi deklarovanými v súbore *token.h*. Hlavnou funkciou *scanner.c* je funkcia *get\_token*. Pre uľahčenie práce a prehľadnosti kódu sme si deklarovali niekoľko makier, ktoré sú extensívne používané v hlavnej funkcii. Funkcia *get\_token* berie postupne znaky zo štandardného vstupu a vytvára token. Tokenu je priradený jeho typ a hodnota, ktorá mu odpovedá. Funkcia začína určovaním jednonakových tokenov, ktoré vie určiť hneď na začiatku. Pokračuje identifikáciou komentárov, ktoré následne ignoruje. Po identifikácii komentárov zisťuje či sa jedna o ID alebo Keyword, pri kľúčových slovách sa následne určuje aj ich typ. Ak sa nejedná ani o jedno pokračuje kontrolou dátových typov, pri ktorých ukladá aj ich hodnoty.

## Syntaktická analýza

---

Riešenie syntaktickej analýzy sme započali vytvorením LL gramatiky, LL tabuľky a precedencnej tabuľky. Následne na ich základe sme vypracovali súbor *parser.c* a *exp\_parser.c*. Tieto súbory pracujú s uzlami deklarovanými v súbore *ast.h*. Spustenie syntaktickej analýzy započne zavolaním funkcie *Parse()*. Tato funkcia postupne prechádza cez tokeny a priradzuje ich do uzlov, pomocou ktorých postupne tvorí abstraktný syntaktický strom na základe LL gramatiky. Súbor *parser.c* ďalej riadi aj precedencnu analýzu volaním funkcii zo súboru *exp\_parser.c*. Tento súbor vytvorí strom výrazov, ktorý je následne pripojený do syntaktického stromu.

## Semantická analýza

---

Sémantická analýza je implementovaná v súboroch *sem\_anal.c*, *symtable.c*, *sem\_anal.h* a *symtable.h*. Spustenie sémantickej analýzy započne zavolaním funkcie *analyse()*. Sémantická analýza je založená na rekurzívnom prechode AST stromu, ktorý je prevzatý od funkcie *parse()*, ktorá ho vygenerovala. Funkcia ďalej využíva globálne deklarovaný AST strom, v ktorom sa nachádzajú built in funkcie. Pri generácii nového AST stromu sa do neho vkladajú built in funkcie práve z tohto globálneho stromu. Funkcia *analyse()* po spustení hľadá *main* a následne postupne rekurzívne prechádza cez AST strom, kde kontroluje validitu dátových typov uložených v ASTNode štruktúrach. Po tejto kontrole započne aj kontrola navratových hodnôt. Po úspešnej validácii dát predáva nový AST strom funkcii *codegen()*, ktorá začína generáciu kódu. Ak validácia neprebehne

úspešne, vyhlási sematicku chybu.

Symtable, ktorý tato funkcia využíva je implementovaný ako AVL strom.

## Generovanie kodu

---

Generátor je implementovaný v súboroch *codegen\_priv.h*, *codegen.h* a *codegen.c*. Spustenie generácie kódu započne zavolaním funkcie *codegen()*. Kód je generovaný na základe rekurzívneho prechádzania abstraktívneho syntaktického stromu podľa dátového typu uloženého v štruktúre *ASTNode*. Kód ďalej využíva aj pomocný Linked List na ukladanie deklarovaných premenných v danej funkcii.



# Dátové štruktúry

---

## Circular Buffer

Implementovane v súboroch *circ\_buff.c* *circ\_buff.h*.

Implementácia Circular Buffer je využitá hlavne v časti Scanner, kde slúži na bezpreblemové získavanie dát a ich následnú validáciu. Na prácu so scannerom ho neskôr využívajú aj časti Parser a Expression Parser. Štruktúra obsahuje klasické funkcie *circ\_buff\_init*, *circ\_buff\_free*, *circ\_buff\_enqueue*, *circ\_buff\_dequeue*, *circ\_buff\_is\_empty*.

## Dynamic String

Implementovane v súboroch *dyn\_str.c*, *dyn\_str.h*.

Implementácia dynamického reťazca je využitá hlavne v Scanner časti programu, kde sprostredkúva validáciu a uschovávanie dát, neskôr je použitá aj v časti Codegen, kde slúži na uľahčenie validácie dát. Štruktúra dynamického reťazca obsahuje klasické funkcie *dyn\_str\_init*, *dyn\_str\_grow*, *dyn\_str\_append*, *dyn\_str\_append\_str* a *dyn\_str\_free*.

## Stack

Implementovane v súboroch *stack.c*, *stack.h*.

Implementáciu nášho zásobníku využívame v Expression Parser časti programu. Štruktúra zásobníku je implementovaná s klasickými funkciami *stackInit*, *stackPush*, *stackPop*, *stackIsEmpty*, *stackClear* a *stackGetTop*. Zásobník sme zvolili pre jeho optimálny prístup k dátam a zachovanie jednoduchosti kódu.