

Tým xhricma00 varianta vv-BVS

Marek Hric
xhricma00

Mikuláš Lešiga
xlesigm00

Roman Andraščík
xandrar00

Adam Veselý
xvesela00

December 5, 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 bez výrazov
- Semantická analýza bez výrazov
- LL Gramatika
- LL tabuľka, precedenčná tabuľka

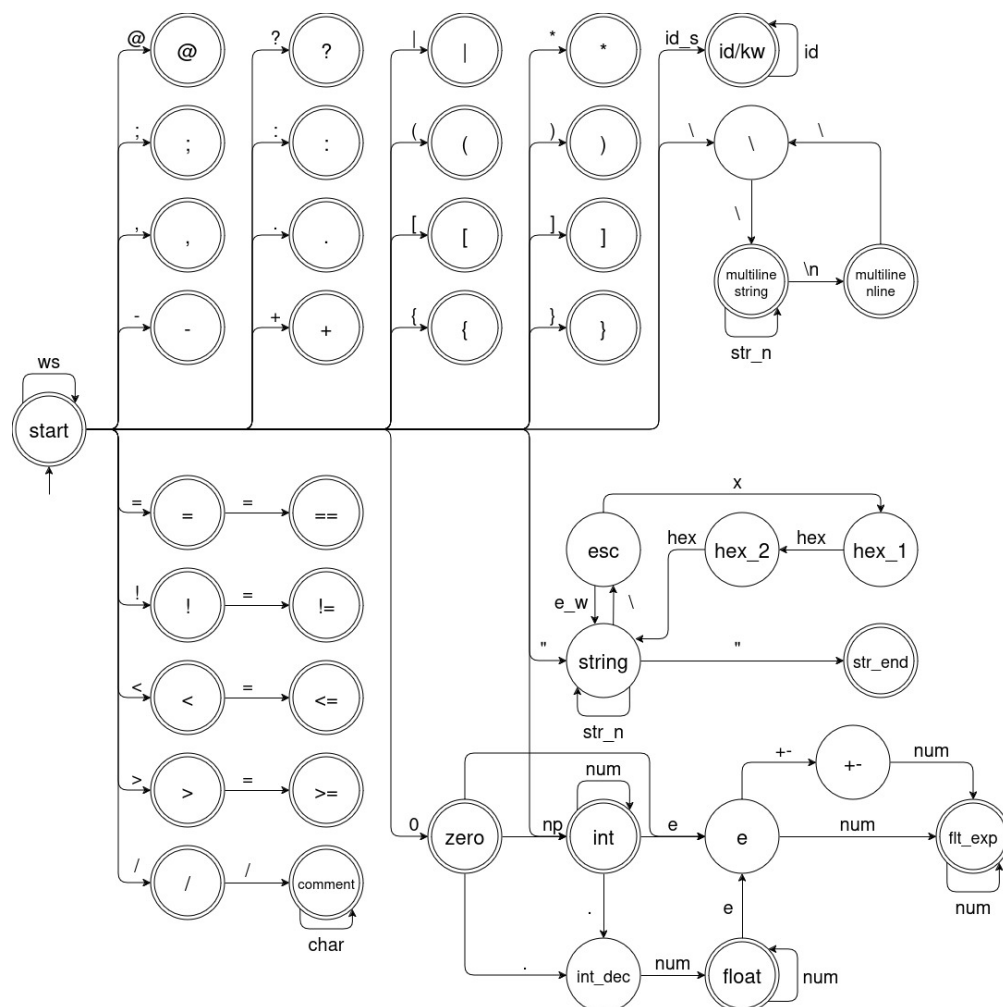
Roman Andraščík :

- Testy
- Pomocné funkcie
- Symtable
- Dokumentácia

Adam Veselý :

- Syntaktická analýza - výrazy
- Semantická analýza - výrazy
- Dátové štruktúry

Diagram konečného automatu :



Legenda:

- **ws** - white space
- **id_s** - znaky začiatku identifikátora (`_a-zA-Z`)
- **id** - znaky identifikátora (`_a-zA-Z0-9`)
- **num** - čísla (0-9)
- **np** - čísla bez nuly (1-9)
- **hex** - hexa čísla (0-9a-fA-F)
- **e** - exponent (eE)
- **char** - ľubovoľný znak
- **e_w** - znaky escape sekvencii bez x (n,t,r)
- **str_n** - ľubovoľný znak bez `\n`

LL-gramatika :

1. $\langle prog \rangle \rightarrow \langle prolog \rangle \langle function_def \rangle \langle end \rangle$
2. $\langle prolog \rangle \rightarrow \text{const ID} = @import(\langle expression \rangle);$
3. $\langle end \rangle \rightarrow EOF$
4. $\langle function_def \rangle \rightarrow \text{pub fn ID}(\langle param_list \rangle) \langle function_type \rangle \langle block \rangle \langle function_def \rangle$
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. $\langle statement \rangle \rightarrow \langle var_declaration \rangle \langle statement \rangle$
12. $\langle statement \rangle \rightarrow ID \langle ID_found \rangle \langle statement \rangle$
13. $\langle statement \rangle \rightarrow \langle if_statement \rangle \langle statement \rangle$
14. $\langle statement \rangle \rightarrow \langle for_loop \rangle \langle statement \rangle$
15. $\langle statement \rangle \rightarrow \langle while_loop \rangle \langle statement \rangle$
16. $\langle statement \rangle \rightarrow \langle return_statement \rangle \langle statement \rangle$
17. $\langle statement \rangle \rightarrow \langle break \rangle \langle statement \rangle$
18. $\langle statement \rangle \rightarrow \langle continue \rangle \langle statement \rangle$
19. $\langle statement \rangle \rightarrow \varepsilon$
20. $\langle ID_found \rangle \rightarrow = \langle asgn_found \rangle;$
21. $\langle ID_found \rangle \rightarrow (\langle expression_list \rangle);$
22. $\langle ID_found \rangle \rightarrow : \langle while_loop \rangle$
23. $\langle var_declaration \rangle \rightarrow \text{const ID} : \langle type_complete \rangle = \langle asgn_found \rangle;$
24. $\langle var_declaration \rangle \rightarrow \text{var ID} : \langle type_complete \rangle = \langle asgn_found \rangle;$
25. $\langle if_statement \rangle \rightarrow \text{if}(\langle expression \rangle) \langle if_found \rangle$

26. $\langle \text{if_found} \rangle \rightarrow \langle \text{optional_value} \rangle \langle \text{block} \rangle \langle \text{else_statement} \rangle$
27. $\langle \text{else_statement} \rangle \rightarrow \text{else} \langle \text{block} \rangle$
28. $\langle \text{else_statement} \rangle \rightarrow \varepsilon$
29. $\langle \text{optional_value} \rangle \rightarrow |ID|$
30. $\langle \text{optional_value} \rangle \rightarrow \varepsilon$
31. $\langle \text{while_loop} \rangle \rightarrow \text{while}(\langle \text{expression} \rangle) \langle \text{optional_value} \rangle \langle \text{optional_statements} \rangle \langle \text{block} \rangle \langle \text{else_statement} \rangle$
32. $\langle \text{return_statement} \rangle \rightarrow \text{return} \langle \text{expression} \rangle;$
33. $\langle \text{expression_list} \rangle \rightarrow \langle \text{expression} \rangle \langle \text{comma_expr_found} \rangle$
34. $\langle \text{expression_list} \rangle \rightarrow \varepsilon$
35. $\langle \text{comma_expr_found} \rangle \rightarrow , \langle \text{expression_list} \rangle$
36. $\langle \text{comma_expr_found} \rangle \rightarrow \varepsilon$
37. $\langle \text{type} \rangle \rightarrow i32$
38. $\langle \text{type} \rangle \rightarrow f64$
39. $\langle \text{type} \rangle \rightarrow []u8$
40. $\langle \text{type} \rangle \rightarrow \text{bool}$
41. $\langle \text{for_loop} \rangle \rightarrow \text{for}(\langle \text{expression} \rangle) \langle \text{optional_value} \rangle \langle \text{block} \rangle$
42. $\langle \text{optional_statements} \rangle \rightarrow \text{:} \langle \text{block} \rangle$
43. $\langle \text{optional_statements} \rangle \rightarrow \varepsilon$
44. $\langle \text{type_complete} \rangle \rightarrow \langle \text{question_mark} \rangle \langle \text{type} \rangle$
45. $\langle \text{question_mark} \rangle \rightarrow ?$
46. $\langle \text{question_mark} \rangle \rightarrow \varepsilon$
47. $\langle \text{single_statement} \rangle \rightarrow \langle \text{var_declaration} \rangle$
48. $\langle \text{single_statement} \rangle \rightarrow ID \langle ID_found \rangle$
49. $\langle \text{single_statement} \rangle \rightarrow \langle \text{if_statement} \rangle$
50. $\langle \text{single_statement} \rangle \rightarrow \langle \text{for_loop} \rangle$
51. $\langle \text{single_statement} \rangle \rightarrow \langle \text{while_loop} \rangle$

52. $\langle single_statement \rangle \rightarrow \langle return_statement \rangle$
53. $\langle single_statement \rangle \rightarrow continue;$
54. $\langle single_statement \rangle \rightarrow break;$
55. $\langle function_type \rangle \rightarrow \langle type \rangle$
56. $\langle function_type \rangle \rightarrow void$
57. $\langle asgn_found \rangle \rightarrow \langle expression \rangle$
58. $\langle continue \rangle \rightarrow continue;$
59. $\langle break \rangle \rightarrow break \langle cycle_current_control \rangle$
60. $\langle break \rangle \rightarrow continue \langle cycle_current_control \rangle$
61. $\langle cycle_current_control \rangle \rightarrow ID$
62. $\langle cycle_current_control \rangle \rightarrow \varepsilon$

Precendečná-tabuľka :

	.?	!	*/	+-	orelse	r	and	or	()	i	\$
.?	<								<	>	<	>
!		<	>	>	>	>	>	>	<	>	>	>
*/		<	>	>	>	>	>	>	<	>	<	>
+-		<	<	>	>	>	>	>	<	>	<	>
orelse		<	<	<	>	>	>	>	<	>	<	>
r		<	<	<	<		>	>	<	>	<	>
and		<	<	<	<	<	>	>	<	>	<	>
or		<	<	<	<	<	<	>	<	>	<	>
(<	<	<	<	<	<	<	<	<	=	<	
)	>	>	>	>	>	>	>	>		>		>
i	>		>	>	>	>	>	>		>		>
\$	<	<	<	<	<	<	<	<	<		<	:)

7

[illegible]

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 extenzívne používané v hlavnej funkcii. Funkcia *get_token* berie postupne znaky zo štandardného vstupu a vytvára token. Tokenu je priradení jeho typ a hodnota, ktorá mu odpovedá. Funkcia začína určovaním jednoznakových 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 nejedna 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 precedenčnej 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()*. Táto 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 precedenčnú 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 prevzatí 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

sématicku chybu.

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

Generovanie kódu

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 abstraktného 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

Character Buffer

Implementované v súboroch *c_buff.c* *c_buff.h*.

Implementácia Character Buffer je využitá hlavne v časti Scanner, kde slúži na bezproblémové získavanie dát a ich následnú validáciu. Na prácu so scannerom ho neskôr využívajú aj časti Parser a Expression Parser. Štruktúra obsahuje klasické funkcie *c_buff_init*, *c_buff_free*, *c_buff_enqueue*, *c_buff_dequeue*, *c_buff_is_empty*.

Dynamic String

Implementované 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

Implementované 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.

Poznámky

BOOLTHEN

Nepodporuje :

- Viac relačných operátorov za sebou