Programski prevodioci

02 Osnovne faze

Fakultet tehničkih nauka, Novi Sad
22-23/Z
Dunja Vrbaški

Programski prevodilac **prevodi program** napisan u jednom jeziku na drugi jezik

Najčešće: viši programski jezik → asembler ili mašinski jezik
Odnosno: ono što programer razume → ono što računar razume

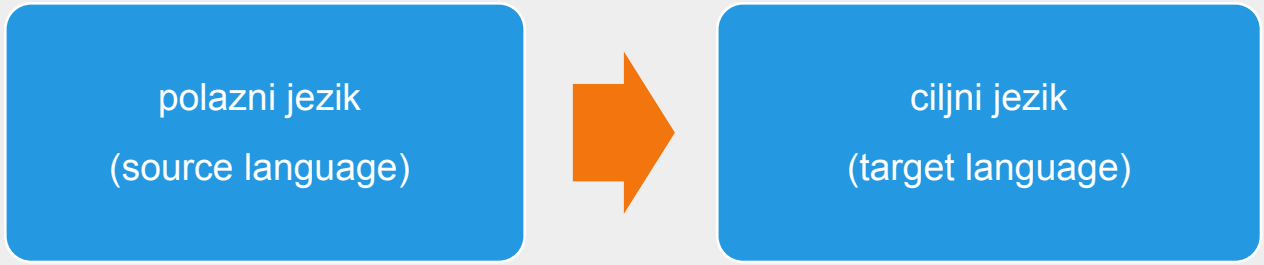
```cpp
int testFunction(int* input, int length) {
  int sum = 0;
  for (int i = 0; i < length; ++i) {
    sum += input[i];
  }
  return sum;
}
```

C++

```asm
testFunction(int*, int):
        test    esi, esi
        jle     .L4
        lea     eax, [rsi-1]
        lea     rdx, [rdi+4+rax*4]
        xor     eax, eax
.L3:
        add     eax, DWORD PTR [rdi]
        add     rdi, 4
        cmp     rdi, rdx
        jne     .L3
        ret
.L4:
        xor     eax, eax
        ret
```

x86-64

(gcc, -O2, c++17)



Najvažniji cilj: prevesti ispravno
Odnosno: očuvati značenje, dobiti očekivano ponašanje

Najvažniji cilj: prevesti ispravno

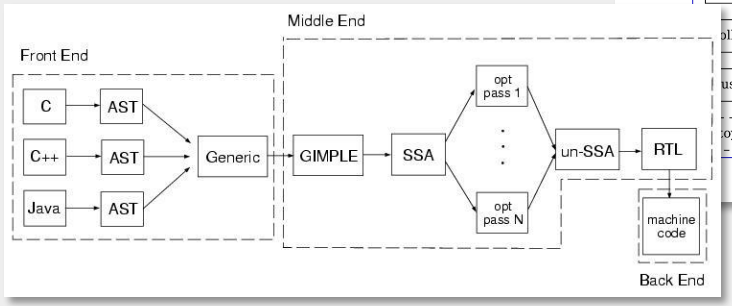
Ostali ciljevi (značajne karakteristike prevodilaca)

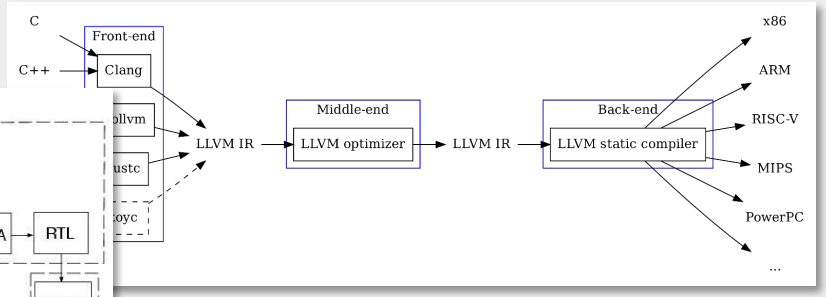
- brzo izvršavanje prevođenja
- brzo izvršavanje prevedenog programa
- optimalno korišćenje resursa
- informativne poruke o greškama, oporavak od grešaka
- mogućnost proširivanja i izmena

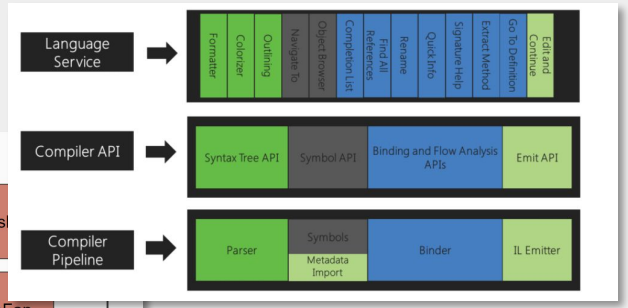
Kompleksan sistem koji se sastoji iz faza.

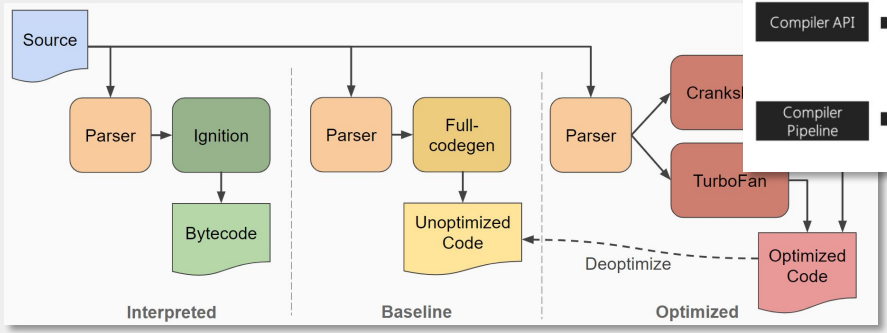
Svaka faza – razvijana, izučavana,
sa posebnim strukturama podataka i algoritmima.

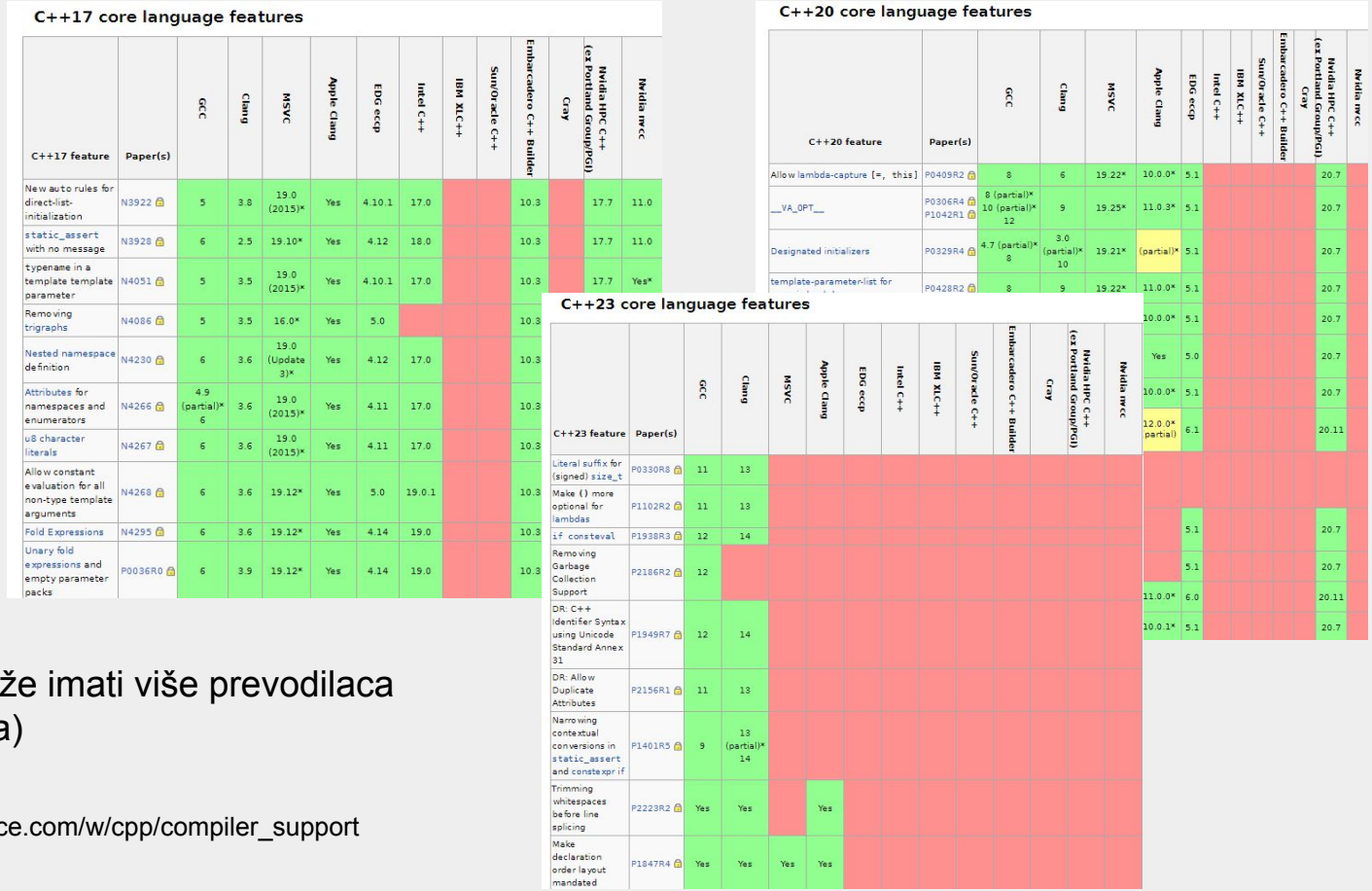
Značajan (i vidljiv) formalan uticaj teorije.


Front End
Middle End
Back End
C
C++
Java
AST
AST
AST
Generic
GIMPLE
SSA
opt pass 1
opt pass N
un-SSA
RTL
machine code


C
C++
Front-end
Clang
LLVM IR
Middle-end
LLVM optimizer
LLVM IR
Back-end
LLVM static compiler
x86
ARM
RISC-V
MIPS
PowerPC
...


Language Service
Formatter
Colorizer
Outlining
Navigate To
Object Browser
Completion List
Find All References
Rename
QuickInfo
Signature Help
Extract Method
Go To Definition
Edit and Continue
Compiler API
Syntax Tree API
Symbol API
Binding and Flow Analysis APIs
Emit API
Compiler Pipeline
Parser
Symbols
Metadata Import
Binder
IL Emitter


Source
Parser
Ignition
Bytecode
Parser
Full-codegen
Unoptimized Code
Parser
Cranks
TurboFan
Optimized Code
Deoptimize
Interpreted
Baseline
Optimized



C++17 core language features

C++17 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc
New auto rules for direct-list-initialization	N3922	5	3.8	19.0 (2015)*	Yes	4.10.1	17.0			10.3		17.7	11.0
static_assert with no message	N3928	6	2.5	19.10*	Yes	4.12	18.0			10.3		17.7	11.0
typename in a template template parameter	N4051	5	3.5	19.0 (2015)*	Yes	4.10.1	17.0			10.3		17.7	Yes*
Removing trigraphs	N4086	5	3.5	16.0*	Yes	5.0				10.3			
Nested namespace definition	N4230	6	3.6	19.0 (Update 3)*	Yes	4.12	17.0			10.3			
Attributes for namespaces and enumerators	N4266	4.9 (partial)* 6	3.6	19.0 (2015)*	Yes	4.11	17.0			10.3			
u8 character literals	N4267	6	3.6	19.0 (2015)*	Yes	4.11	17.0			10.3			
Allow constant evaluation for all non-type template arguments	N4268	6	3.6	19.12*	Yes	5.0	19.0.1			10.3			
Fold Expressions	N4295	6	3.6	19.12*	Yes	4.14	19.0			10.3			
Unary fold expressions and empty parameter packs	P0036R0	6	3.9	19.12*	Yes	4.14	19.0			10.3			

C++20 core language features

C++20 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc
Allow lambda-capture [=, this]	P0409R2	8	6	19.22*	10.0.0*	5.1						20.7	
__VA_OPT__	P0306R4 P1042R1	8 (partial)* 10 (partial)* 12	9	19.25*	11.0.3*	5.1						20.7	
Designated initializers	P0329R4	4.7 (partial)* 8	3.0 (partial)* 10	19.21*	(partial)*	5.1						20.7	
template-parameter-list for	P0428R2	8	9	19.22*	11.0.0*	5.1						20.7	
					10.0.0*	5.1						20.7	
					Yes	5.0						20.7	
					10.0.0*	5.1						20.7	
					12.0.0* partial)	6.1						20.11	
						5.1						20.7	
						5.1						20.7	
					11.0.0*	6.0						20.11	
					10.0.1*	5.1						20.7	

C++23 core language features

C++23 feature	Paper(s)	GCC	Clang	MSVC	Apple Clang	EDG eccp	Intel C++	IBM XLC++	Sun/Oracle C++	Embarcadero C++ Builder	Cray	Nvidia HPC C++ (ex Portland Group/PGI)	Nvidia nvcc
Literal suffix for (signed) size_t	P0330R8	11	13										
Make () more optional for lambdas	P1102R2	11	13										
if consteval	P1938R3	12	14										
Removing Garbage Collection Support	P2186R2	12											
DR: C++ Identifier Syntax using Unicode Standard Annex 31	P1949R7	12	14										
DR: Allow Duplicate Attributes	P2156R1	11	13										
Narrowing contextual conversions in static_assert and constexpr if	P1401R5	9	13 (partial)* 14										
Trimming whitespaces before line splicing	P2223R2	Yes	Yes		Yes								
Make declaration order layout mandated	P1847R4	Yes	Yes	Yes	Yes								

Jedan jezik može imati više prevodilaca (implementacija)

https://en.cppreference.com/w/cpp/compiler_support

Dizajn programskog jezika

Osmisliti + definisati + implementirati

→ Softverski projekat

[osmisliti] Šta sve treba odlučiti?

Ideja

Opšti ciljevi – edukacija, istraživanje ili profesionalno; DSL ili opšte namene; izvršavanje programa ili međujezik...

Specifični tehnički ciljevi - brzina kompajliranja, brzina izvršavanja, sigurnost, pristup i upravljanje memorijom,...

Razvoj - koristimo alate ili sve radimo samostalno; svaki korak zasebno ili neke spajamo; koji jezik koristimo za implementaciju;…

sve odluke povezane; često postoji kompromis

[definisati] Šta sve treba precizirati?

Jezike, alate, arhitekture – koje tehnologije koristiti

Leksiku jezika – skup pravila koja definišu šta sve čini jezik (azbuka i dozvoljene reči)

Sintaksu jezika – skup pravila koja definišu ispravno formirane programske strukture (gramatika)

Semantiku jezika – skup pravila koja određuju i upravljaju značenjem jezika

Strukture podataka i alg. – koje će se kreirati u različitim fazama prevođenja (tabele, stabla…)

Obradu grešaka – gde i na koji način se vrši obrada grešaka

Međujezik – poznavati postojeći ili formirati novi

Optimizacije – obavezne i dodatne

Testove – za svaku fazu

Arhitekturu rešenja – procesi, integracije, automatizacija, build alati, testiranje

→ funkcionalan sistem

[implementirati] Šta sve treba poznavati?

Strukture podataka – heš tabele, stabla, grafovi,...

Algoritmi – obilasci stabla/grafova, rekurzivni spust, bojenje grafova,…

Teorija formalnih jezika – regularni jezici, kontekstno osetljive/slobodne gramatike,...

Teorija automata – konačni automati (deterministički, nedeterministički)…

Programski jezici – jezik za koji pravimo prevodilac; jezik u kom implementiramo prevodilac; ciljni jezik

Arhitektura računara, operativni sistemi – zbog ciljnog jezika

Metode testiranja

Build alati

Razvojni alati

Terminologija i teorija

Jednostavno ili ne?

Primitivni prevodilac, univerzitet, hobby jezik – nije jako teško

Prevodilac za industriju, DSL – znatno teže

Prevodilac opšte namene - prilično ozbiljno (standardizacije)

Teorija – matematika, formalizmi → vidljivost značajna i na osnovnim nivoima

Puno različitih povezanih koraka/delova → lako može postati nepregledno/neuredno

Proširivost

Korisnici

Korisnici su programeri!

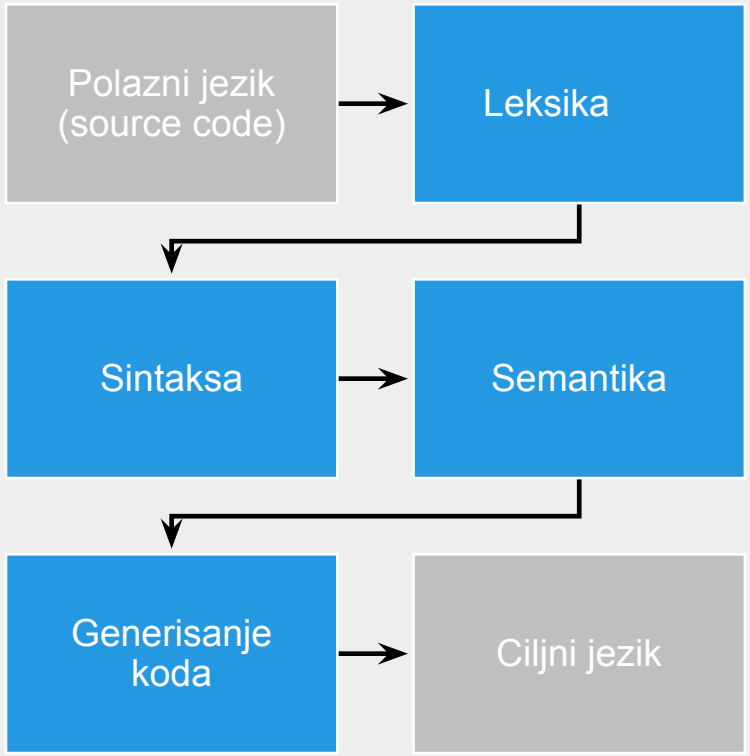
Bagovi i problemi u kompajleru – veliki problem

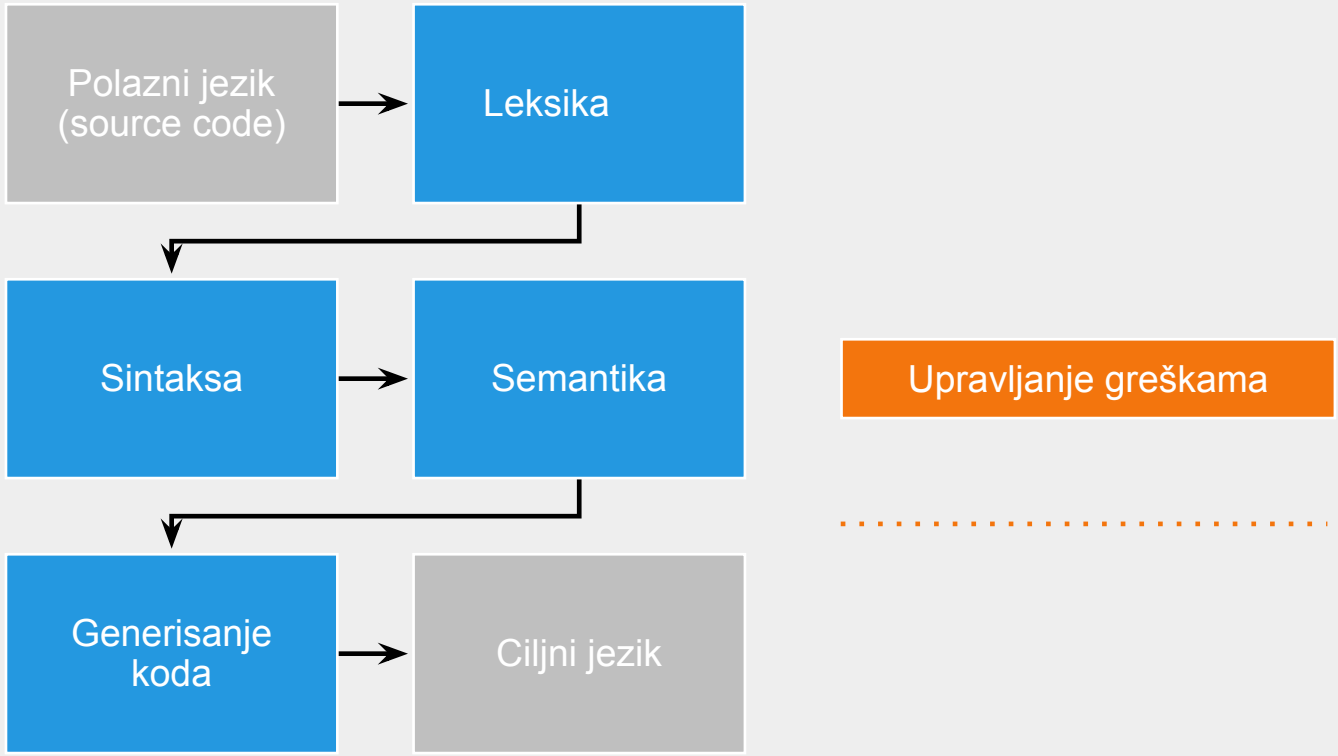
Programer očekuje: brzo + pouzdano + jednostavno za korišćenje

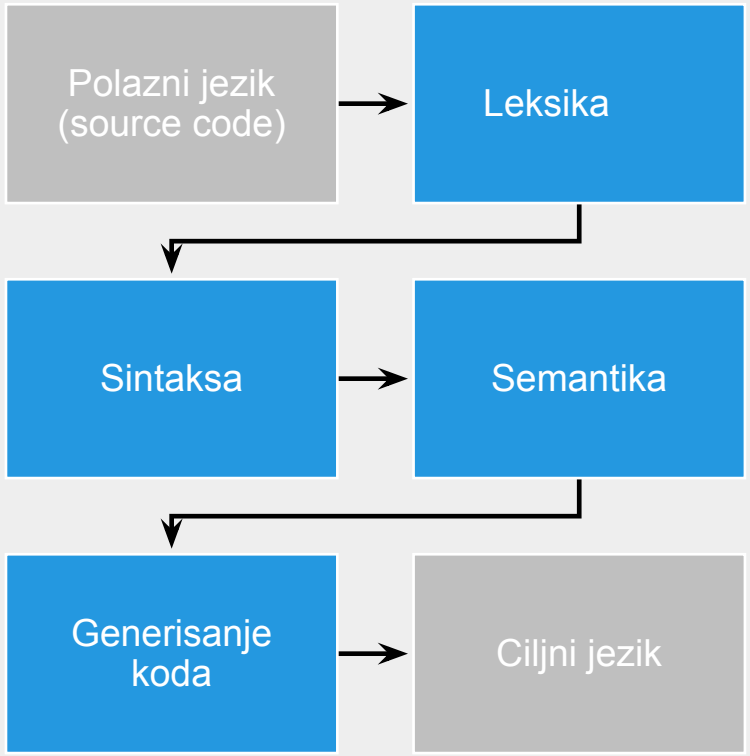
U opštem slučaju - ne obraća pažnju na kompajler – rešava SVOJ problem

Da bi se (šire) prihvatio prevodilac – korisnici su zahtevniji nego korisnici ostalih softverskih rešenja.

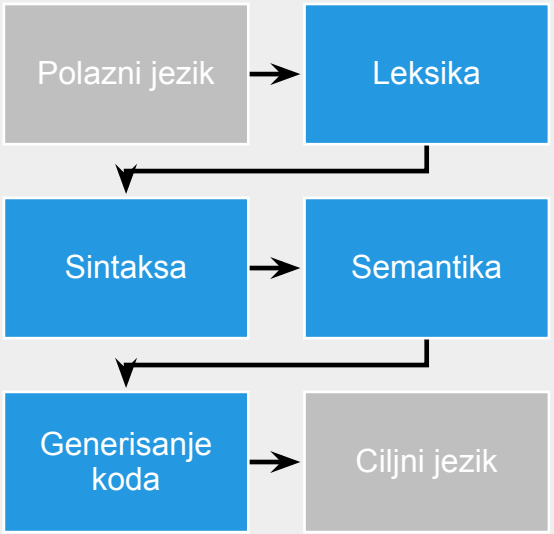
Obično očekuju i neku biblioteku (core, standard) i alate

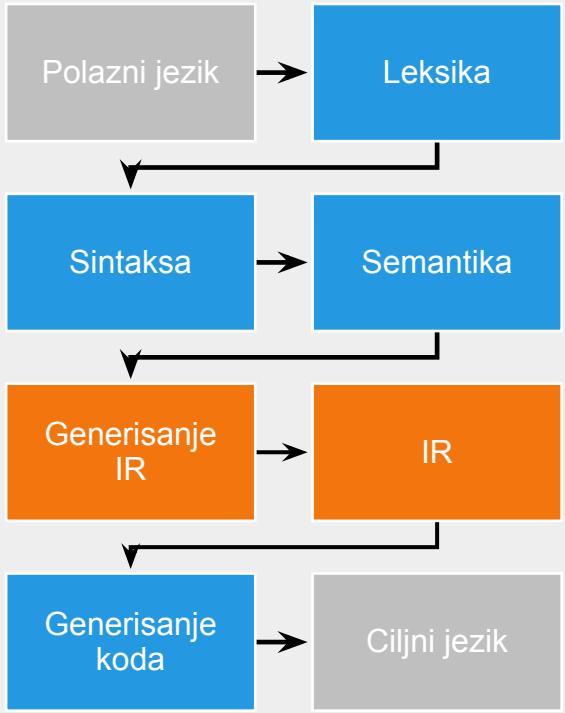

Polazni jezik
(source code)
Leksika
Sintaksa
Semantika
Generisanje
koda
Ciljni jezik


Polazni jezik
(source code)
Leksika
Sintaksa
Semantika
Generisanje
koda
Ciljni jezik
Upravljanje greškama

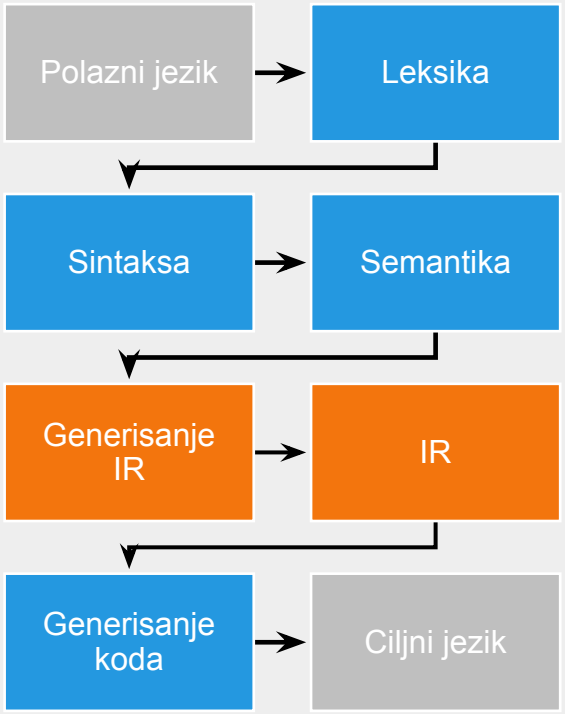


- Upravljanje greškama
- Transofrmacije koda
- Analiza koda
- Optimizacije
- Upravljanje resursima
- ...





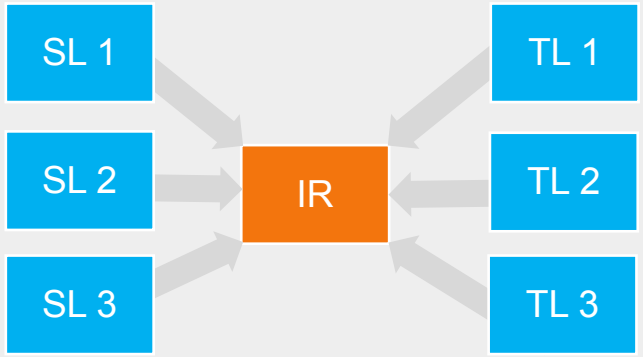
IR – Intermediate representation (međureprezentacija)



Front end

Middle end

Back end



Leksička analiza - Skener

```
if (x == 5)
{
   y = 3; //komentar
}
```

if	(	x	==	5	)	{	y	=	3	;	}

lekseme

```
<KEYW, IF, 1> <LPAREN,, 1> <IDENT, x, 1>
<ROP, EQ, 1> <NUM, 5, 1> <RPAREN,, 1>
<LBRAC,, 2> <IDENT, y, 3> <ASSIGN,, 3>
<SC,, 3> <RBRAC,, 4>
```

tokeni

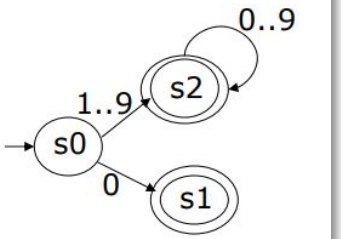
Regularni izrazi
Flex
C
build
test primeri

automati

```
identifier      ->  letter ( letter | digit )*
ar_operator     ->  "+" | "-"
rel_operator    ->  "==" | "<"
```

```
SRC = scanner
.PHONY: clean

$(SRC): lex.yy.c
    gcc -o $@ $+

lex.yy.c: $(SRC).l
    flex $<

clean:
    rm -f lex.yy.c
    rm -f $(SRC).tab.c
    rm -f $(SRC).tab.h
    rm -f $(SRC)
```

```
%option noyywrap yylineno
%{
  union {
    int i;
    char *s;
  } yylval;

  enum tokens { _TYPE = 1, _IF, _ELSE, _RETURN,
                _LPAREN, _RPAREN, _LBRACKET, _RBRAC
                _SEMICOLON, _ASSIGN, _AROP, _RELOP,
                _INT_NUMBER, _UINT_NUMBER };
  char *token_strings[] = { "NONE", "_TYPE", "_IF",
```




```
int abs(int i) {
  int res;
  if(i > 0)
    res = 0 - i;
  else
    res = i;
  return res;
}

int main() {
  return abs(-5);
}
```

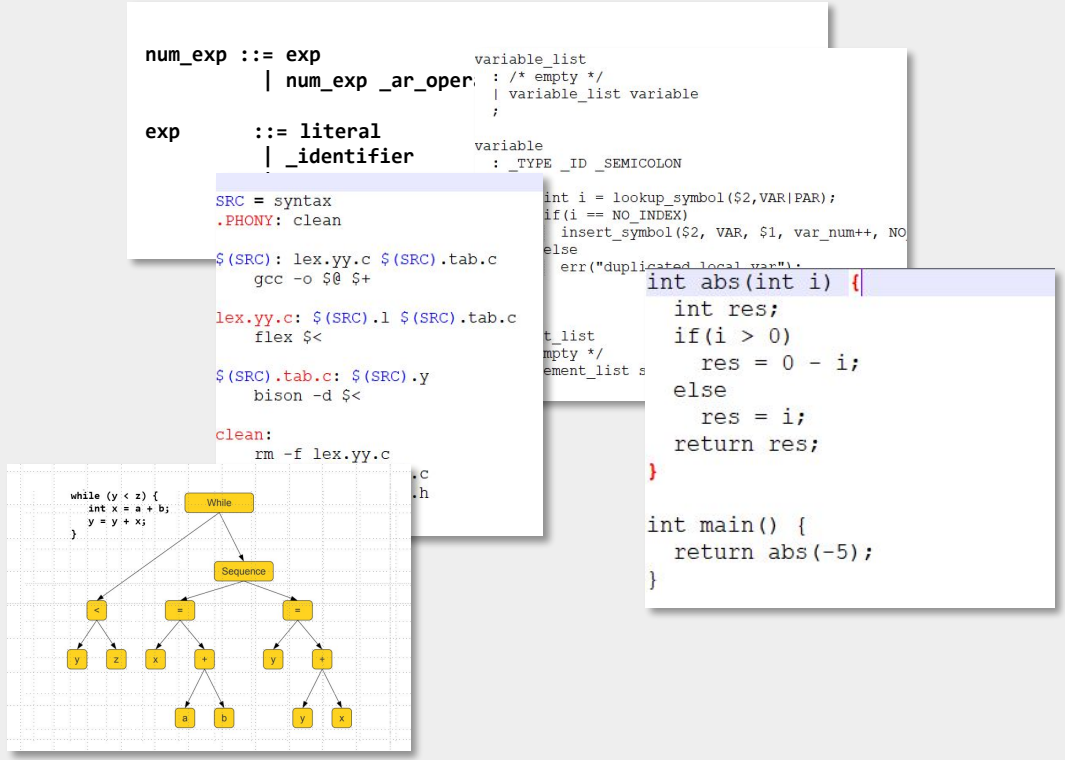
Sintaksna analiza - Parser

```
x = 5;
y 3;
```

<IDENT, x> <ASSIGN, >, <NUM, 5> - OK
<IDENT, y> , <NUM, 3> - Greška

(na primer)

Gramatike (formalno)
Flex
Bison
C
build
test primeri



Semantička analiza

```
int x = 5;
string str = “abc”;
x = str;
```

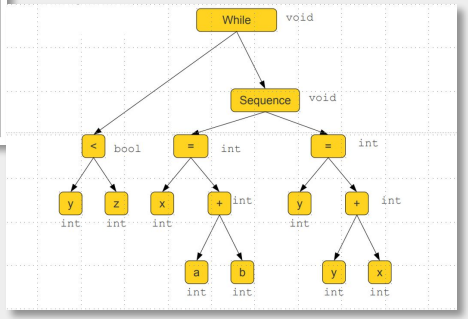
Pravila semantike
Flex
Bison
C
build
test primeri

tabela simbola

- svi globalni identifikatori moraju biti međusobno različiti
- lokalni identifikatori raznih funkcija mogu biti identični

```
if(get_type(fun_idx) == VOID)
   err("Function cannot return value");
else if(get_type(fun_idx) != get_type($2))
   err("incompatible types in return");
return_count++;
```

STRING SIMBOLA	VRSTA SIMBOLA			
main	FUN	p		
p	PAR	parametra	parametra	-
x	VAR	tip promenljive	redni broj promenljive	-
100	LIT	tip literala	-	-
%0	REG	tip sadrzaja registra	-	-



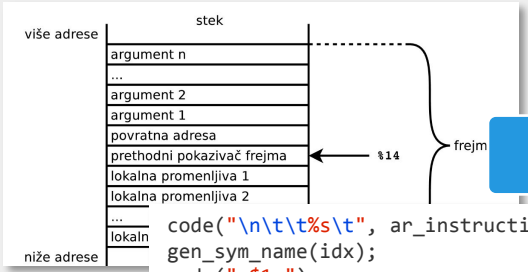
Generisanje koda


```
int f(int p) {
  int a;
  return p + a;
}
```

```
f:
    PUSH %14
    MOV %15,%14
    SUBS %15,$4,%15
@f_body:
    ADDS 8(%14),-4(%14),%0
   ...
```

Asembler (hipotetski)
Flex
Bison
C
build
test primeri

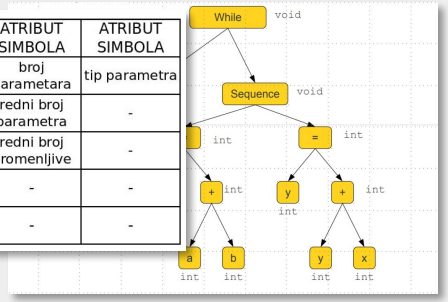
emulator



source → hip asm

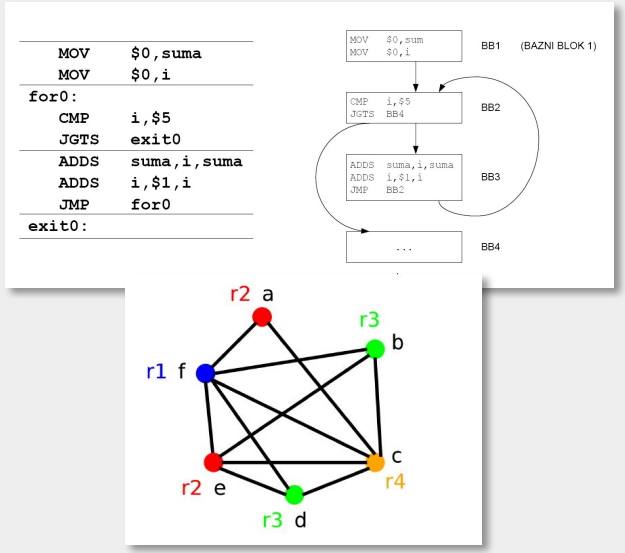
```
code("\n\t\t%s\t", ar_instructions[ADD + (get_type(idx) - 1) * AROP_NUMBER]);
gen_sym_name(idx);
code(",$1,");
gen_sym_name(idx);
```

STRING SIMBOLA	VRSTA SIMBOLA	TIP SIMBOLA	ATRIBUT SIMBOLA	ATRIBUT SIMBOLA
main	FUN	povratni tip funkcije	broj parametara	tip parametra
p	PAR	tip parametra	redni broj parametra	-
x	VAR	tip promenljive	redni broj promenljive	-
100	LIT	tip literala	-	-
%0	REG	tip sadrzaja registra	-	-



hip asm → ? → target

- runtime
- upravljanje memorijom
- optimizacije
- kompajliranje, interpretiranje, JIT
- virtualne mašine, međukod
- tipovi
- …



Source code	SC			Programski jezici (3-4)
Leksika	SC + Gramatika L → Tokeni	Tabela simbola, AST kreiranje		Formalni jezici
Sintaksa	Gramatika S → AST			Konačni automati
Semantika	AST → AST’	Tabela simbola, AST korišćenje		Heš tabele, stabla, grafovi – SPA
Generisanje IR	AST’ → IR			Optimizacije (koda/međukoda, lokalne/globalne)
Optimizacije	IR → IR’			Razvojni alati
Generisanje koda	IR’ → TC TC → TC’			Pomoćni alati
Optimizacije				Dizajn softvera i razvojni procesi
Target code	TC			Teorija – Oblast: PL