SI 413: Programming Languages

Professor Keith Sullivan

This course is actually three courses!

Typically, SI 413 is split into three courses in a CS undergrad curriculum

- Functional programming
- Compilers
- Programming language design

Skills Outcome

Outcomes over the next few months:

- Learn a functional language
- Write an interpreter for a simple language
- Write a compiler for a virtual machine
- Learn a programming language on your own

Programming Languages

Why study programming languages?

- Understand obscure features
- Choose alternatives based on implementation knowledge
- Make good use of debuggers, linkers, and related tools
- Develop a vocabulary for describing programming languages

Phases of Programming

What does programming actually involve?

- Choose a language for the task
- Learn the language
- Write a program
- Compile the program
- Execute the program

Note: an interpreter essentially does compilation and execution simultaneously, on-the-fly.

Compiled vs. Interpreted

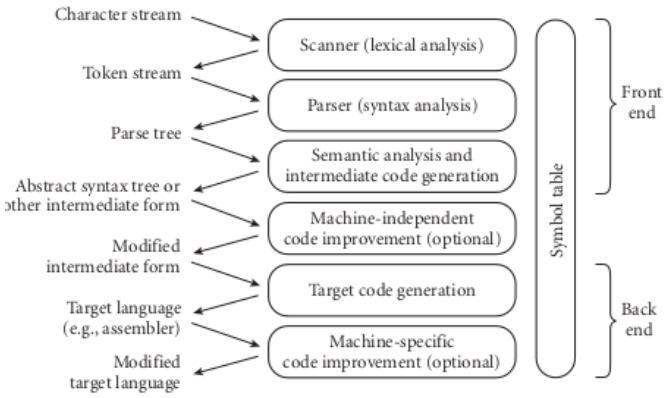
Common compiled languages:

Common interpreted languages:

In between options:

- Just in Time Compilation
- Bytecode Compilation

Compilation



Scanning

Read individual characters and group them into tokens

```
int main(int argc, char *argv)
{
  printf ("Hello World\n");
  return 1;
}
```

int	main	(	int	argc	,
char	*	argv	)	{	printf
(	"	Hello	World	\	n
"	)	;	return	1	; }

Parsing

- Organizes tokens from the scanner into a parse tree
- Parse tree shows how the tokens make a valid program
- Recursive rules called context free grammar

```
statements: statements statement | statement

statement:
| type ID LPAREN function_variables RPAREN tail
| CONST type ID ASSIGN expression SEMI
| CONTINUE SEMI
| RETURN SEMI
| BREAK SEMI

type: INT | CHAR | FLOAT | VOID | DOUBLE

tail: statement SEMI | LBRACE statements RBRACE
```

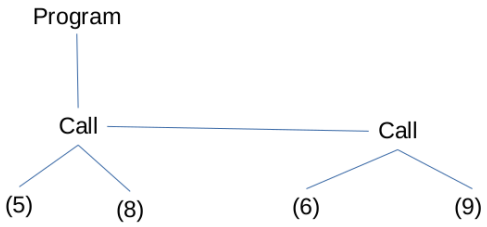
Semantic Analysis

- Discovery of the *meaning* of a program
- Symbol table: maps identifiers to information (type, scope, structure)
- Symbol table enforces *static semantic* rules of the language

Index	Symbol	Type	Value
1	int	type	
2	char*	type	
3	argc	(1)	
4	argv	(2)	
5	printf	func:(8) → (2)	
6	return	func:(9) → (7)	
7	void	type	
8	const_string	(2)	Hello world
9	const_int	(1)	1

Abstract Syntax Tree

Pares down parse tree to essentials and annotates with symbol table



Index	Symbol	Type	Value
1	int	type	
2	char*	type	
3	argc	(1)	
4	argv	(2)	
5	printf	func:(8) → (2)	
6	return	func:(9) → (7)	
7	void	type	
8	const_string	(2)	Hello world
9	const_int	(1)	1