language design but also to understand the implementation issues that a good programmer has to understand.

Types of programming languages

- **Procedural** languages (e.g., C, Fortran, Pascal) are mainly concerned with what the computer should do (the procedural view).
- **Imperative** languages (e.g., Ada, Modula-2, Simula, Lisp) focus on the step-by-step execution of a program (the data flow view).
- **Functional** languages (e.g., Haskell, Scheme) emphasize the calculation aspects, where expressions are evaluated and values are computed.
- **Object-oriented** languages (e.g., Smalltalk, C++, Java, C#) are based on the concepts of objects, classes, and inheritance.
- **Scripting** languages (e.g., Python, Ruby, PHP, JavaScript) are often used for dynamic features such as HTML generation.
- **Database query languages** (e.g., SQL) are used for managing and querying databases.
- Some languages support multiple paradigms, such as Python, which can be used for both imperative and functional programming.

Converting source code to intermediate code

- **Lexical analysis** (ANR) breaks the source code into smaller units called tokens.
- **Syntactic analysis** (PARS) checks if the tokens form a valid program structure.
- **Semantic analysis** (SEM) checks the program for semantic errors.
- **Code generation** (G) produces the target code for the computer.

**Lex** and **Parser**

- **Lex** generates a lexical analyzer from a regular expression (regex) file.
- A scanner is a program that reads source code and converts it into tokens.

**Parser**

- A **recursive-descent** parser is one that recursively examines the input code and returns a parse tree if it is well-formed.
- A **top-down** parser starts with the entire program and proceeds to smaller units.
- A **generalized LR** parser is a type of LR parser that allows more complex grammars.
- A **LR(1)** parser is a powerful parser that can handle many common grammars.

**Grammar**

- A **grammar** is a set of rules that defines the syntax of a language.
- A **regular grammar** is one that generates regular languages.
- A **context-free grammar** is one that generates context-free languages.

**Compiler**

- A **compiler** is a program that translates source code into a target language.
- The **source program** is the program written in a high-level language.
- The **target program** is the program in a lower-level language.
- **Optimization** is the process of improving the target program's efficiency.

**Code examples**

```c
// A[j..len(A)-1] is sorted
imperative sort(A):
  i = 0
  while (i < len(A) - 1)
    if (A[j] > v) break;
    for (j = i; j < len(A) - 1; j++)
      if (A[j] > v) swap(A[j], A[j + 1])
    i = i + 1
  end
```

```csharp
// A[j..len(A)-1] is sorted
Haskell

```