PARADIGMS AND COMPUTER PROGRAMMING FUNDAMENTALS (PCPF)

ITC305

2022-23

Subject In-charge

Ms. Aaysha Shaikh

Professor Dept. of Information Technology

SFIT Room No. 322

email: aayshashaikh@sfit.ac.in

Module 0

Lecture 0

Objectives, Outcomes, Syllabus and Marking

Scheme 8/3/2022 Lecture 0: Introduction to PCPF

Tests and Marking Scheme

- Course Code: ITC305
- Course Name: Paradigms and Computer Programming Fundamentals
- No. of Lectures: 3 per week
- No. of Tests:
  - Quiz / Poll / Presentations
  - Assignments – 2
  - IATs -2 (20 marks)
  - Final Exam (80 marks)

Objectives of PCPF Course

Sr. No.	Objectives
1	To introduce various programming paradigms and the basic constructs that underline any programming language.
2	To understand data abstraction and object orientation.
3	To introduce the basic concepts of declarative programming paradigms through functional and logic programming.
4	To design solutions using declarative programming paradigms through functional and logic programming.
5	To introduce the concepts of concurrent program execution.
6	To understand use of scripting language for different problem domains.

Ms. Aasha Shaikh

Outcomes of PCPF Course

Sr. No.	Course Outcomes
CO1	To **Comprehend** and **Compare** different programming paradigms.
CO2	To **Comprehend** the Object Oriented Constructs and **Use** them in program design.
CO3	To **Comprehend** and **Use** the concepts of declarative programming paradigms through functional and logic programming.
CO4	To **Design** and **Develop** programs based on declarative programming paradigm using functional and/or logic programming.
CO5	To **Comprehend** role of concurrency in parallel and distributed programming.
CO6	To **Comprehend** different application domains for use of scripting languages.

Ms. Aasha Shaikh

Module-0

- **Prerequisite (2 Hrs)**
  - Compilation and interpretation Focus on overview of compilation steps.

Ms. Aasha Shaikh

Module-1

- **Introduction to Programming Paradigms and Core Language Design Issues (10 Hrs)**

- Introduction to different programming paradigms.
- Introduction Names, Scopes, and Bindings, Scope Rules, Storage Management.
- Type Systems, Type Checking, Equality Testing and Assignment.
- Subroutine and Control Abstraction: Stack Layout, Calling sequence, parameter passing
- Generic subroutines and modules. Exception handling, Co routines and Events.

- **Self-learning Topics:**
  - Implementation of basic concepts using any programming language.

Module-2

- **Imperative Paradigm: Data Abstraction in Object Orientation (5 Hrs)**

– Grouping of data and Operations- Encapsulation, Overloading, Polymorphism, Inheritance, Initialization and Finalization, Dynamic Binding.

- **Self-learning Topics:**

– Implementation of OOP concepts using OOP language. 8/3/2022

Ms. Aasha Shaikh

Module-3

- **Declarative Programming Paradigm: Functional Programming (7 Hrs)**

– Introduction to Lambda Calculus, Functional Programming Concepts, Evaluation order, Higher order functions, I/O- Streams and Monads.

- **Self-learning Topics:**

– Implementation of I/O using any programming language.

- **Declarative Programming Paradigm: Logic Programming (6 Hrs)**
  - Logic Programming with PROLOG - Resolution and Unification, Lists, Arithmetic execution order, imperative control flow, database manipulation, PROLOG facilities and deficiencies.
- **Self-learning Topics:**
  - Implementation of basic operation and control flow using PROLOG in healthcare.

Module-5

- **Alternative Paradigms: Concurrency (4 Hrs)** – Concurrent Programming Fundamentals, Implementing synchronization, Message Passing - Background and Motivation, Multi threaded programs, Communication and Synchronization, Language and Libraries, Thread creation Syntax

- **Self-learning Topics:**

– Implementation of module IV concepts for real time application.

Ms. Aasha Shaikh

Module-6

- **Alternative Paradigms: Scripting Languages (5 Hrs)** – Common characteristics, Different Problem domains for using scripting, Use of scripting in Web development–server and clients side scripting, Innovative features of scripting languages - Names and Scopes, string and pattern manipulation ,data types ,object orientation.
- **Self-learning Topics:**
  – Implement a simple website for client-server.

IMPERATIVE PROGRAMMING :

The imperative paradigm is the oldest and the most popular programming paradigm.

PROCEDURAL PROGRAMMING:

Procedural programming is a refinement of the imperative paradigm adding subroutines (or procedures).

OBJECT ORIENTED PROGRAMMING PARADIGM

An approach to the solution of problems in which all

computations are performed in the context of objects

DECLARATIVE POGRAMMING PARADIGM

Functional Programming ,Logic Programming Paradigm-The use of factsand rules to represent information

CONCURRENT PROGRAMMING.

A technique in which two or more processes start, run in an interleaved fashion through context switching and complete in an overlapping time period by managing access to shared resources

Ms. Aasha Shaikh

Why Study Programming Languages?

- To improve the ability to develop effective algorithms. •

To Increase the ability to express ideas

- To improve the use of familiar languages.

• To increase the vocabulary of useful programming constructs. • To improve the background for choosing appropriate languages

• To make it easier to learn a new language.

• To make it easier to design a new language.

• To simulate useful features in languages that lack them. • To make better use of language technology wherever it appears.

• For overall advancement of computing

Ms. Aasha Shaikh

Different Programming languages

4

6 7

Solutio

n

C/C++

Operating Systems and System Tools

Java

Enterprise Application Development

Python

Artificial Intelligence & Machine Learning.

C#

Application & Web Development **R**

Data Analysis.

JavaScript

Rich Interactive Web Development.

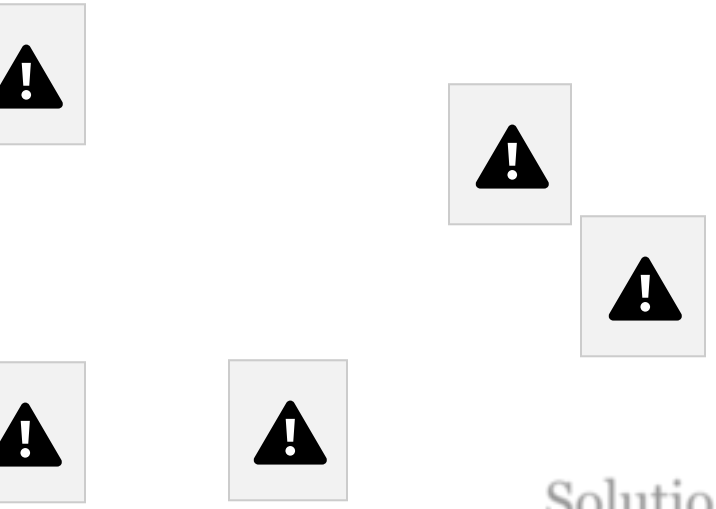
Golang

Server-Side Programming

Ms. Aasha Shaikh

Different Programming

languages



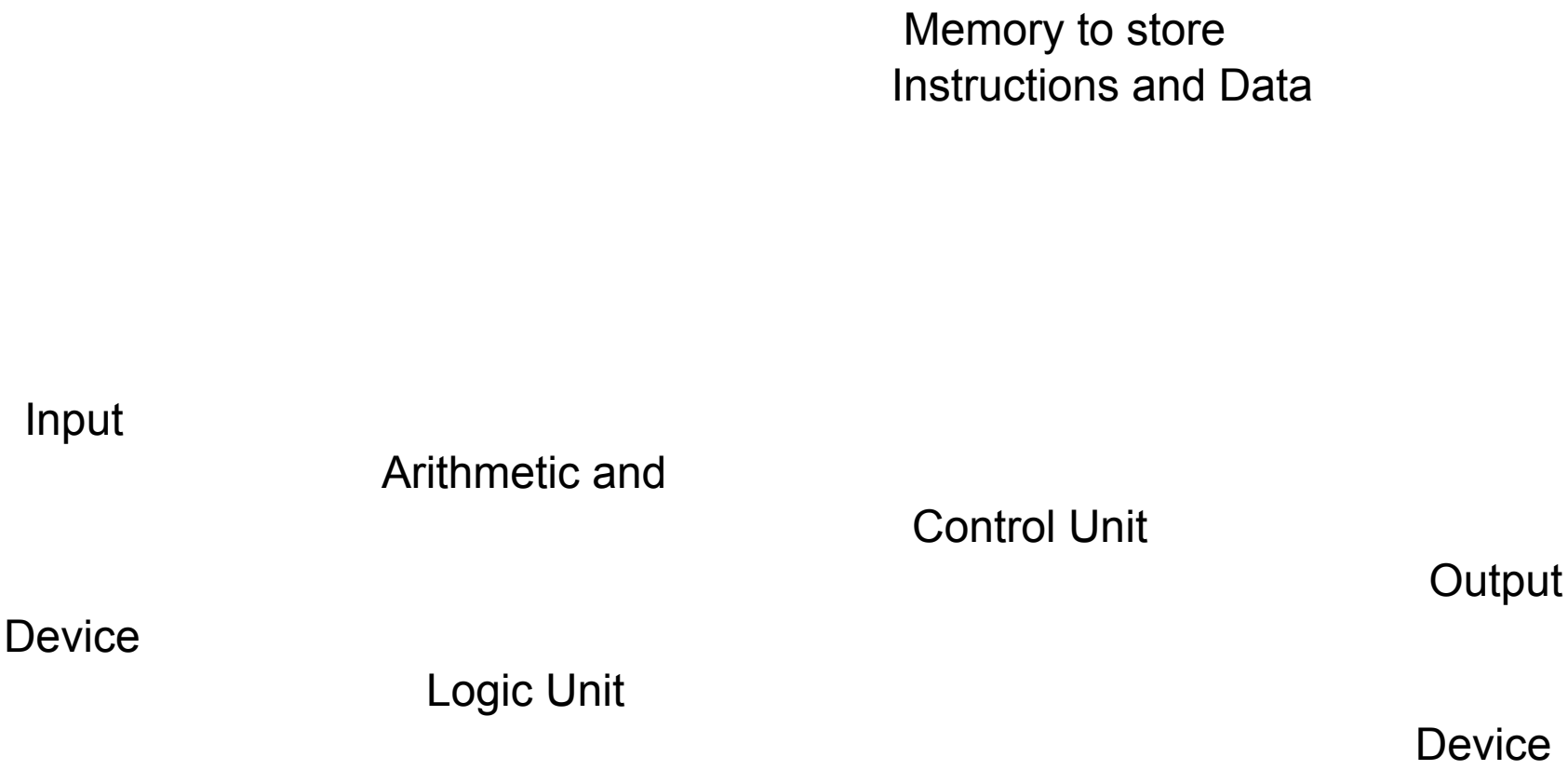
Solutio

n

Ms. Aasha Shaikh

Why Use Programming Language?

- We use von Neumann machines
- We need a programming language to talk to computer.



Low Level Machine Language

Ex. Light bulb is controlled by a processor running a program in main memory.

Machine Instruction	Machine Operation
00000000	Stop Program
00000001	Turn bulb fully on
00000010	Turn bulb fully off
00000100	Dim bulb by 10%
00001000	Brighten bulb by 10%
00010000	If bulb is fully on, skip over next

	instruction
00100000	If bulb is fully off, skip over next instruction
01000000	Go to start of program (address 0)

- Machine language or machine code is the basic low level programming language of computers.
- It is a sequence of binary

digits (bits, 0s and 1s) comprehended only by computers but not by humans.

controls a processor, causing it to add, compare, move data from one place to another, etc.

- This machine code directly

High Level Programming Language

• Programming Language is a notation of writing programs (a set of instructions) to describe algorithm and data structures. • Program is a sentence of programming language and a finitary specification of computation.

- Machine language is processor specific and difficult to understand by humans, hence high level, English like programming languages such as C, C++, java were invented.
- High level programming language is not understood by

machine, hence we need to use another program like compiler or interpreter which will convert high level program code into machine code.

Ms. Aasha Shaikh

Compilation Process

Compiler

a program that converts instructions into a machine-code or **lower-level** form so that they can be read and executed by a computer.

Loader/Linker

a program used with a compiler or assembler to
provide links to the **libraries** needed for an **executable**
program

Assembler

Preprocessor

In computer science,
a **preprocessor** (or **precompiler**) is a program that processes its input data to produce output that is used as input to another program. The output is said to be a **preprocessed** form of the input data, which is often used by some subsequent programs like compilers.
a program for converting instructions written in low-level symbolic code into machine code.

Preprocessor

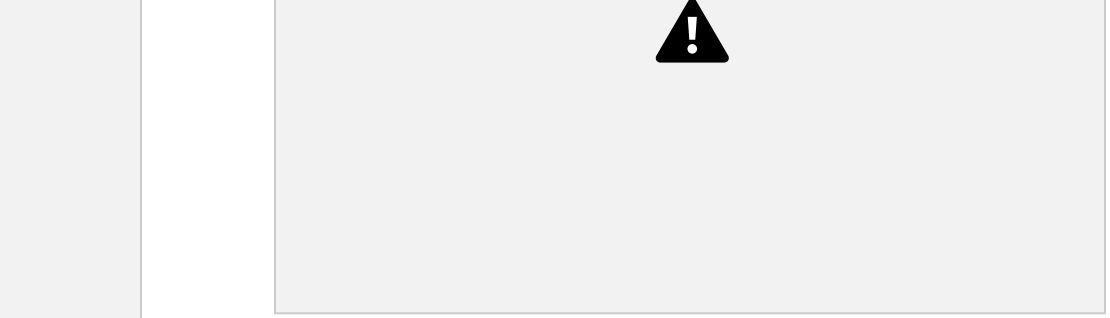
Compilation Process

I. Comments Removal

ii. Macros Expansion

iii. File inclusion

In computer science, a **preprocessor** (or **precompiler**) is a program that processes its input data to produce output that is used as input to another program. The output is said to be a **preprocessed** form of the input data, which is often used by some subsequent programs like compilers.



gcc -e Testcode.c

Ms. Aasha Shaikh

Preprocessor

Compilation Process

Compiler

a program that converts instructions

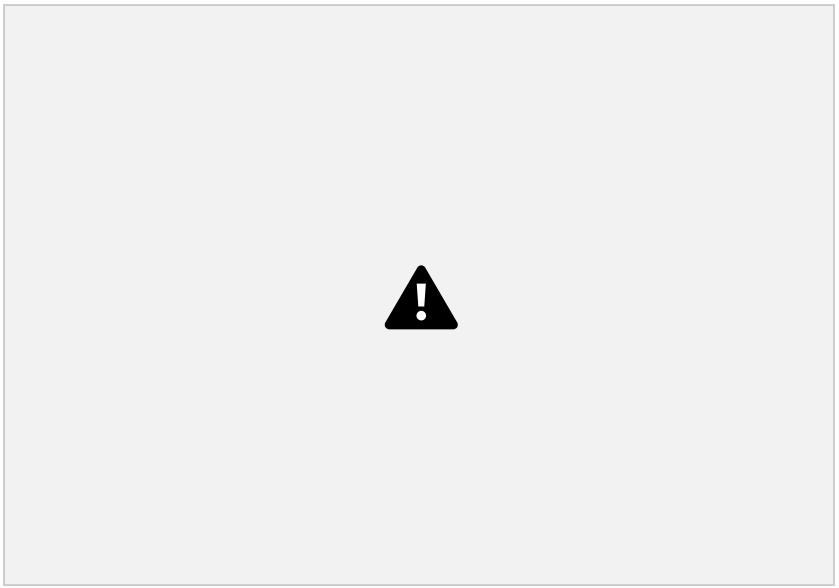
into a machine-code or **lower-level**

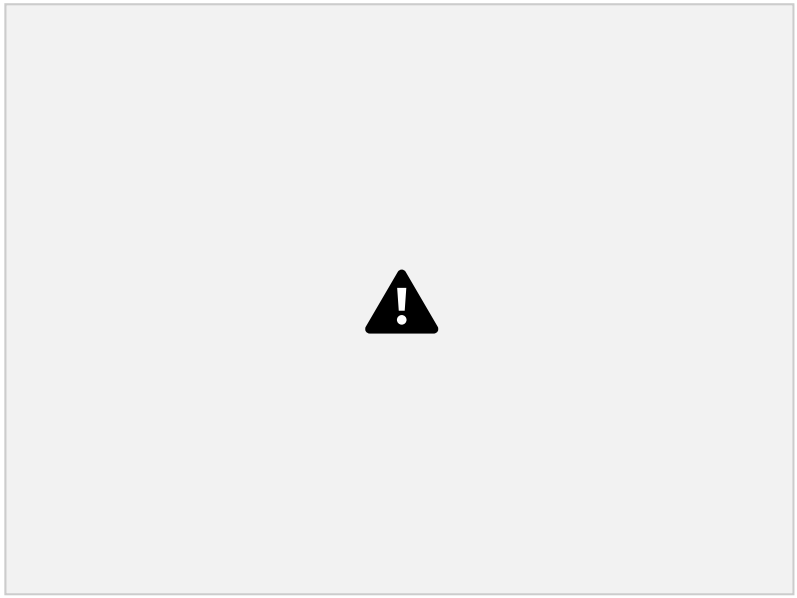
form so that they can be read and executed by a computer.

Testcode.i

inbuilt *compiler software* to convert the intermediate (.i) file into an **Assembly file** (.s) having assembly level instructions (low-level code)

Compiling phase in C uses an





Testcode.i

Testcode.s 8/3/2022 Lecture 0: Introduction to PCPF

Ms. Aasha Shaikh

Preprocessor

Compilation Process

It takes basic instructions from an

Testcode.s

assembly code file into
and converts them

machine type known as
the **object code**.

Assembler

a program for
converting instructions
written in low level
symbolic code into
machine code.

Testcode.i

Compile

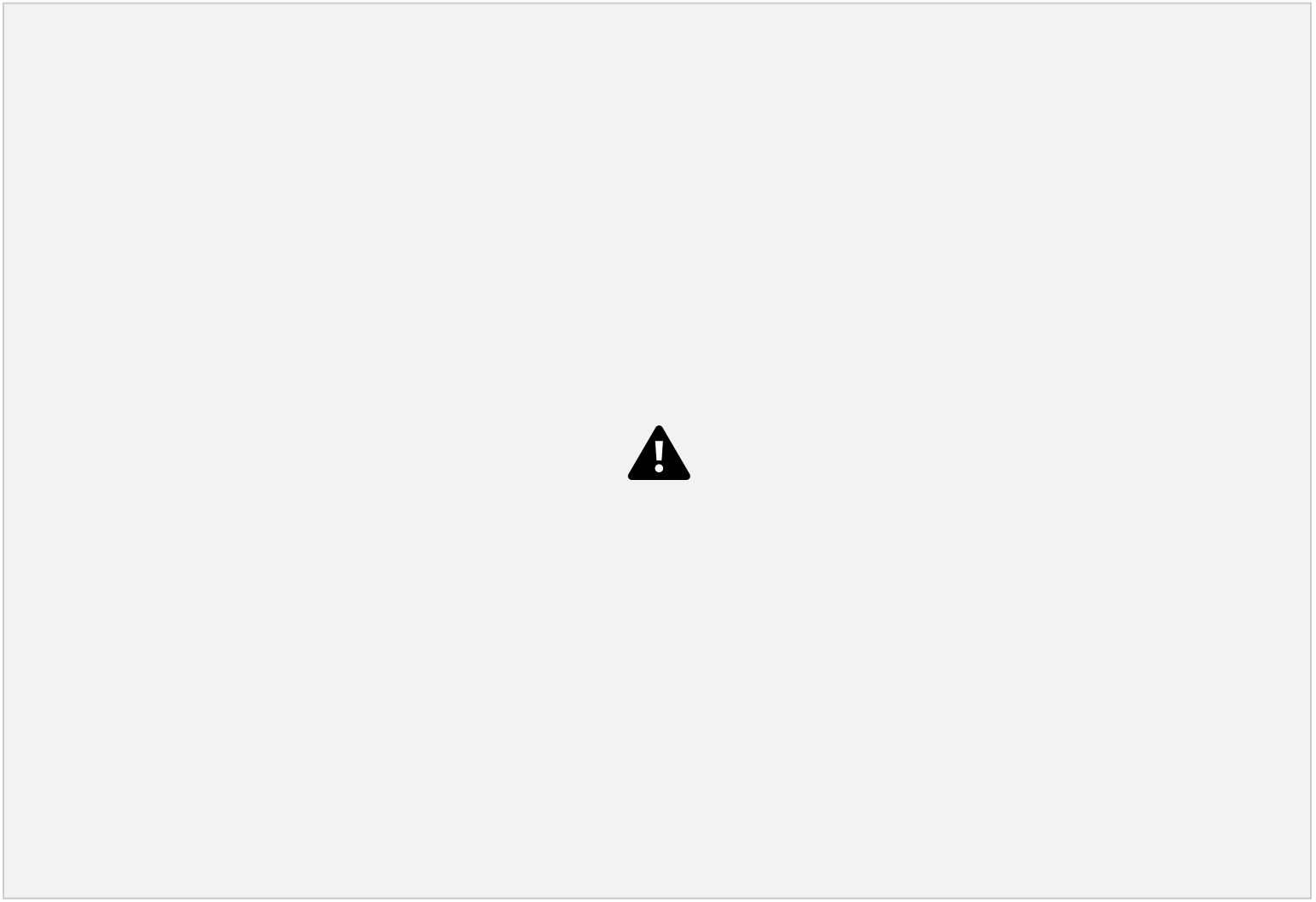
r

gcc -c Testcode.c
binary/hexadecimal
code specific to the

Testcode.s

Testcode.o

Assembler



Ms. Aasha Shaikh

Compilation Process

Preprocessor

Relocatable code

Assembler

Static library
(.lib, .a)

Loader/Linker

A program used with a compiler or assembler to

Testcode.s

libraries

provide links to the

executable

needed for an

program

Testcode.o

Compiler


Testcode.exe

Quiz

01 Name the popular Programing language?


02 Name any 2 type of computer programming paradigm?

03 True or false ? C is procedural programming language?

Arrange 4 stages of compilation process.

Linker/Loader 4

1

2

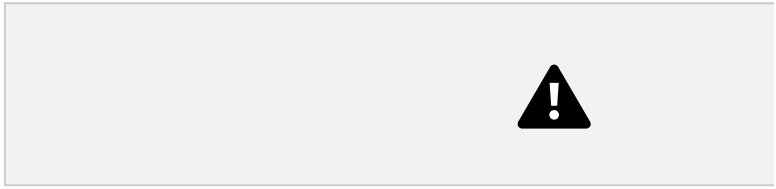
3
Assembler

8/3/2022 Lecture 0: Introduction to PCPF

Preprocessor Compiler

26

04



Ms. Aasha Shaikh

Phases of Compiler

Assembly code

Intermediate Code Generator

Symbol Table

Higher Level Language

Error Handling

Lexical Analyzer

Syntax Analyzer

Code optimizer

Semantic Analyzer

Target code Generation

Phases of Compiler

There are two phases of compilers: Analysis phase and Synthesis phase

- **Analysis phase :** creates an intermediate representation from the given source code.

- **Synthesis phase :** creates an equivalent target program from the intermediate representation.

- These 2 phases also commonly termed as the **front end** (Analysis phase)and the **back end**(Synthesis phase).

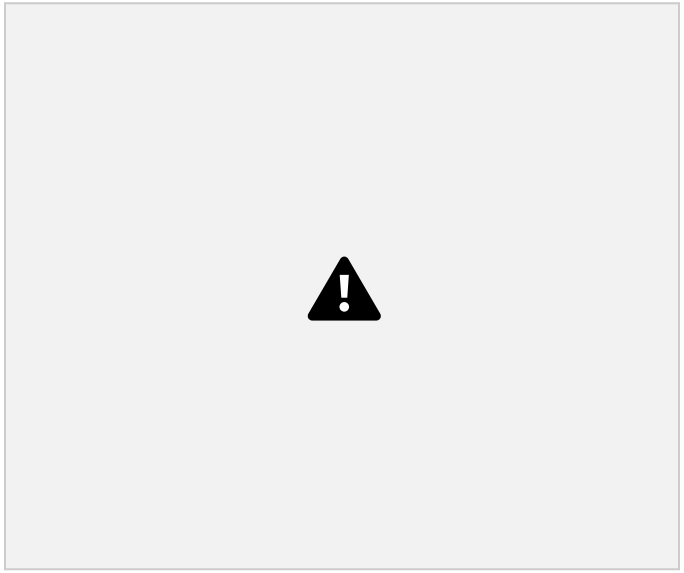
- Front-end constitutes the Lexical analyzer, semantic analyzer, syntax analyzer, and intermediate code generator. On the other hand, the back-end part consist of code optimizer and target code Generation.

Phases of Compiler

1- Lexical Analyzer

- In the lexical Analyzer phase entire source code get scanned by the compiler. It takes entire high-level language as input, reads its character and group them into lexemes/ tokens.

- It makes the entry of corresponding tokens into the symbol table and passes the token to next phase.



Ms. Aasha Shaikh

Lexical Analyzer

The primary function of lexical analyzer are:

- Identifying lexical units in source code.

- Ignoring comments in the source program.
- Removing white space, tab spaces, new-line character.

Example:

If this enters as an input: **x=a+b*c;**

No. of tokens =8

id=id+id*id

2- Syntax Analyzer

It takes all the tokens one by one and uses Context-Free Grammar to construct the parse tree.

Following is the list of tasks performed in syntax analysis phases:

- Take tokens from lexical analyzer as input.
- Checks the expression whether it is syntactically correct or not.
- Reports syntax error, if any
- Construct hierarchical structure known as syntax tree or parse tree.

Syntax Analyzer

Example of Syntax for

“a+b*c”

+

*

a

b c

Ms. Aasha Shaikh

3- Semantic Analyzer

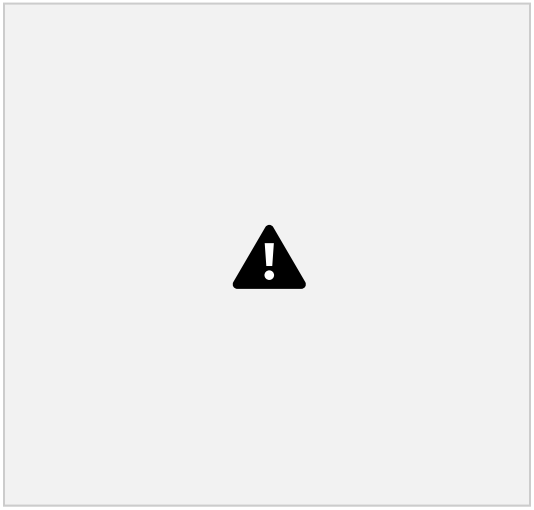
- Semantic Analyzer verifies if the parse tree is semantically correct or not, meaningful or not. If not, it produces a verified parse tree.
- It will check for mismatching data types, incompatible operands, a function called with improper arguments,

undeclared variables, etc.

- In our example,

  it will give same parse tree in output

  as it is already semantically correct.



4- Intermediate Code Generator

- It generates intermediate code, which is a form that can be readily executed by a machine.
- Example – Three address codes
- Intermediate code is converted to machine language using

the last two phases which are platform dependent. • The intermediate code for our example **x=a+b*c** would be: T1= b*c

T2= a+T1

X=T2

5- Code Optimizer

- It transforms the code so that it consumes fewer resources and produces more speed
- For the intermediate code:

T1= b*c

T2= a+T1

X=T2

- The optimized code can be:

T1= b*c

X= a+T1

Ms. Aasha Shaikh

Target Code Generator

Target Code Generator

- Target code generator generates machine understandable code.
- Target code generator generates machine understandable code. •

This is the final stage of compilation. The optimized code is

- This is the final stage of compilation. The optimized code is converted into relocatable machine code. Example, let's take

converted into relocatable machine code. Example, let's take assembly code as are Target code.

assembly code as are Target code.

- For our previous code: The target code / Assembly code can be:

• For our previous code: The target code / Assembly code can be:

T1=b*c X=a+T1

code can be:

a R0, b R1, c R2 a R0, b R1, c R2

Mul R1, R2

Mul R1, R2

Add R0, R2

Add R0, R2

Mov R2, X

Mov R2, X

Symbol Table

- Symbol Table is a data structure which is created and maintained by the compiler to keep track of variables i.e. it stores information about

the scope and bonding of various entities such as variale and function names, classes, object etc.

- A symbol table makes it easier for the complier to search the identifier record and retrieve it quickly

- The information is symbol Table are being collected by the analysis phase of the compiler and are used by the synthesis phases of the compiler.

Error Handling

Like Symbol Table all the phases of compiler are also connected to Error handler.The tasks of the Error Handler are to detect each error, report it to the user, and then make some recovery strategy and implement them to handle the error.

The primary functions od error Handler are:

- Error Detection
- Error Reporting
- Error Recovery

Compiler v/s Interpreter

Compiler	Interpreter
A compiler translates the entire source code in a single run.	An interpreter translates the entire source code line by line.

It consumes less time i.e.it is faster than an interpreter.	It consumes much more time than the compiler i.e., it is slower than the compiler.
It is more efficient.	It is less efficient.
CPU utilization is more.	CPU utilization is less as compared to the compiler.
Both syntactic and semantic errors can be checked simultaneously.	Only syntactic errors are checked.
The compiler is larger.	Interpreters are often smaller than compilers.

Compiler v/s Interpreter

Compiler	**Interpreter**
It is not flexible.	It is flexible.

The localization of errors is difficult.	The localization of error is easier than the compiler.
A presence of an error can cause the whole program to be re-organized.	A presence of an error causes only a part of the program to be re-organized.
The compiler is used by the language such as C, C++.	An interpreter is used by languages such as Java.

Activity

- WordCloud

  www.menti.com

  code: 80182306

- Write comment on Compiler v/s interpreter:

  https://padlet.com/aayshashaikh/PCP

F

Thank You