

Lecture 1

11 September 2024

09:06 AM

Lecture 2

11 September 2024 09:06 AM

Recap-

1. Procedural Programming focuses on Functions which are like the steps
2. Object Oriented Programming focuses on Classes/Real World entity which are like objects

Examples

Procedural Oriented Programming	Object Oriented Programming
Student Wakes Up -> Gets Ready -> Attend Class Teacher Wakes Up -> Get Ready -> Prepare -> Give Lectures	Student Class Name, Roll No, Branch Functions -Wakes Up, Gets Ready, Attend lecture, Write Exam Teacher Class Name, ID, Branch, Wakes Up, Get Ready, Prepare, Give Lectures, Evaluate
There is no ownership in POP If(Gives_Lecture){ Attend_Class; } There is no description for different programs. There are only functions.	The function attend lecture is dependent on the teacher Eg. If(Teacher.Gives_Lectures == True){ Student.Attend_lecture; } If(student.write_exam==True){ Teacher.evaluate; } If instead of student we write teacher, then the programming language will show an error.

Procedural Oriented Programming	Object Oriented Programming
Functions are steps which are executed in a particular sequence.	Object is a real world entity, which can be defined by characteristics and functions.
Procedural Programming is not very organised as any function can be associated to any entity.	Object Oriented is more organized; Because we structure the functions in different entities
<div><h3>•PROCEDURAL-ORIENTED PROGRAMMING LANGUAGES:</h3><ul style="list-style-type: none">•Procedural-oriented programming languages are based upon procedural call concepts.•This kind of programming language divides the entire program into small portions called routines or functions.•This kind of programming language makes it easy for the user to easily track the flow of control of the program and code re-usability can be done throughout the program.•Some of the procedural-oriented programming languages are....<div><div><p>•FORTRAN</p><p>•ALGOL</p><p>•COBOL</p><p>•BASIC</p><p>•PASCAL</p><p>•C</p></div><div><pre>3 int product(int x, int y); 4 5 int main(void) 6 { 7 c = product(a,b); 8 9 printf("%i\n",c); 10 11 return 0; 12 } 13</pre><p>Function Prototype - int is the return type and int x and int y are the function arguments</p><p>Main Function - int is always the return type and there are no arguments, hence the (void). curly braces {} mark the start and end of the main function</p><p>Function call - product(a,b); a and b are global variables the function is passed. Here the values returned by the function are assigned to the variable c</p><p>Function Definition - contains the function statement return a * y;</p></div></div><div><pre>#include <stdio.h> // function declaration int addNumbers(int a, int b); int main() { int num1 = 5, num2 = 10, sum; // function call sum = addNumbers(num1, num2); printf("Sum of %d and %d is %d", num1, num2, sum); return 0; // function definition int addNumbers(int a, int b) { int result = a + b; return result; }</pre><p>Calling of function - returns the result to the calling function</p><p>Header file - returns return address</p></div></div>	<div><h3>•OBJECT-ORIENTED PROGRAMMING LANGUAGES:</h3><ul style="list-style-type: none">• Object-oriented programming languages are a kind of language, which mainly depends on objects.• In this type of programming language, the entire program is divided into parts, which are known as objects.• These objects are used to implement some of the programming methods like polymorphism, inheritance, abstraction, etc.• The main advantage of object-oriented programming languages is that these kinds of languages are faster and easier to execute, and easy to maintain.<p>Some of the object-oriented programming languages are....</p><div><p>Scala C++ Java Python C# ruby</p><pre>graph TD PackageName[Package Name] --> ClassVariable[Class Variable] ClassVariable --> ClassDog[Class Dog] ClassVariable --> ClassCat[Class Cat] ClassVariable --> ClassBird[Class Bird] ClassDog --> UsedForTransportation1[Used for Transportation] ClassCat --> UsedForTransportation2[Used for Transportation] ClassBird --> UsedForTransportation3[Used for Transportation] UsedForTransportation1 --> IncludeVehicle1[Include vehicle] UsedForTransportation2 --> IncludeVehicle2[Include vehicle] UsedForTransportation3 --> IncludeVehicle3[Include vehicle]</pre></div></div>
Communication between the functions	Communication between the objects

Function Declaration - Tells us the datatype and the name of its function.
Function Definition - Tell us the functionality and the logic of the function.

High Level Language and Low Level Language.

High-Level Language	Low-Level Languages
High-Level Languages are simple to pick up and comprehend.	Low-level languages may be difficult to learn and comprehend.
Because they need translation software, they run slower than lower-level languages.	They work at a bottleneck pace.
They allow for a great deal more abstraction.	They don't allow much if any, abstraction.
At the hardware level, they don't have a lot of options.	They are extremely near to the hardware and may assist in the development of software at the hardware level.
Hardware expertise is not necessary for creating programs.	Hardware expertise is required while creating programs.
The programs are simple to change.	It's tough to change programs.
Several instructions may be executed by a single statement.	The assertions may be directly translated to instructions on the CPU.

High Level Language is same on all devices

Low level language maybe specific for a singular device because of different versions CPU,

Context for Tomorrow's Class

Linker - Links two or more functions and explains the sequence.

Loader - Loads the functions into the CPU.

Compilers

Lecture 3

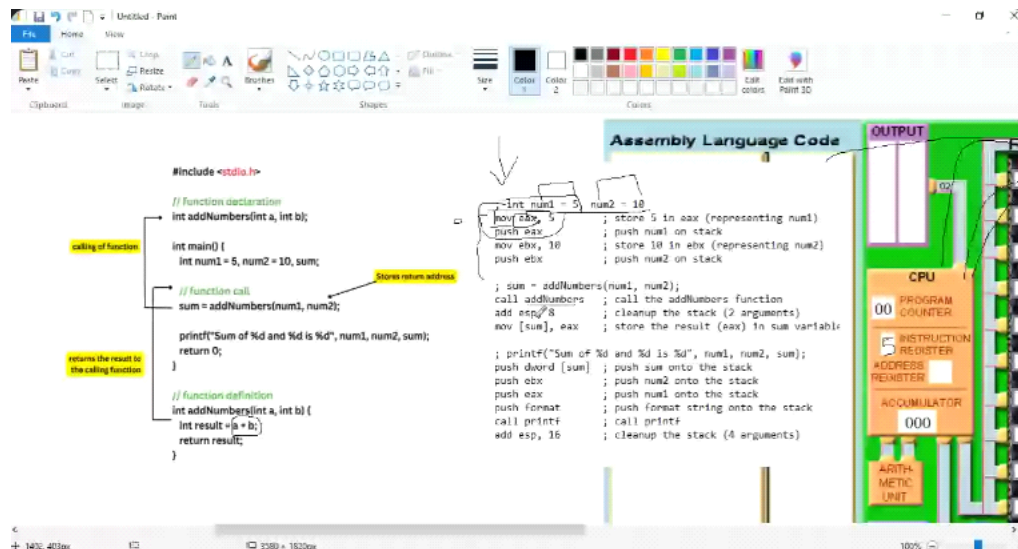
11 September 2024

09:07 AM

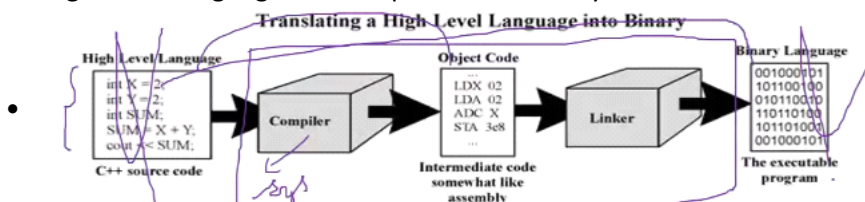
Lecture 4

12 September 2024 08:53 AM

1. Assembly language acts as a middle ground between our code and memory language(binary)
2. Execution of code is happening on CPU.

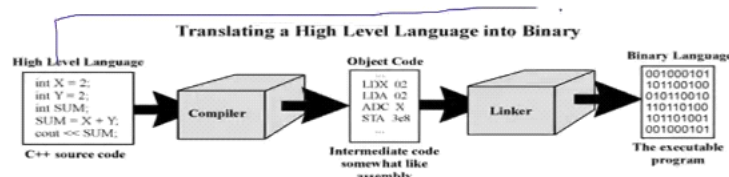


- High Level Language --> Compiler --> Assembly Code --> Linker--> Binary Code



SUMMARY OF THE COMPILATION PROCESS:

- **Lexical Analysis:** Tokenizes the source code.
- **Syntax Analysis:** Builds a syntax tree based on grammar.
- **Semantic Analysis:** Ensures logical correctness and builds a symbol table.
- **Intermediate Code Generation:** Produces an abstract, low-level code.
- **Optimization:** Improves performance of the intermediate code.
- **Target Code Generation:** Translates intermediate code into machine-specific code.
- **Assembly and Linking:** Converts to machine code and combines with other modules.
- **Loading and Execution:** Runs the final binary on the machine.



Lexical Analysis :
 Syntax Analysis
 Semantic Analysis

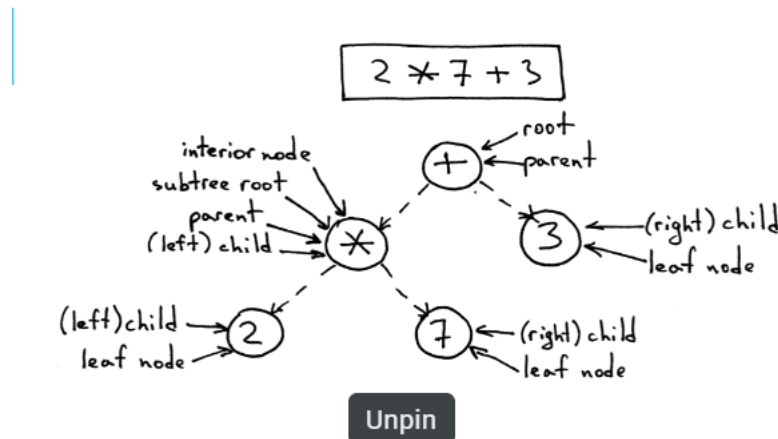
Lexical Analysis : Tokenizes the source code

1. Token - smallest unit code in a programming language.
2. Example `int x = 5;`
Tokens KEYWORD "int", IDENTIFIER "x", OPERATOR "=", CONSTANT "5", PUNCTUATION ";" are the 5 tokens.

Syntax Analysis - Builds a syntax tree based on grammar

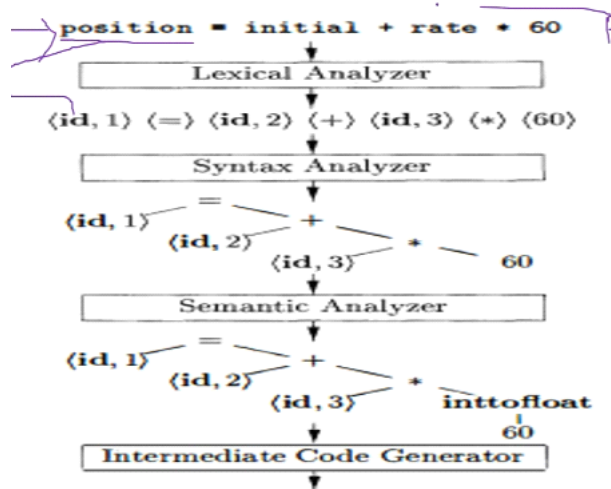
1. Syntax - Programming rules which are language specific.

•SYNTACTIC ANALYSIS



1. The parser checks the sequence of tokens, confirms the grammar rules of the language.
2. It constructs a parse tree that shows the syntactic structure of the program.

Semantic Analysis



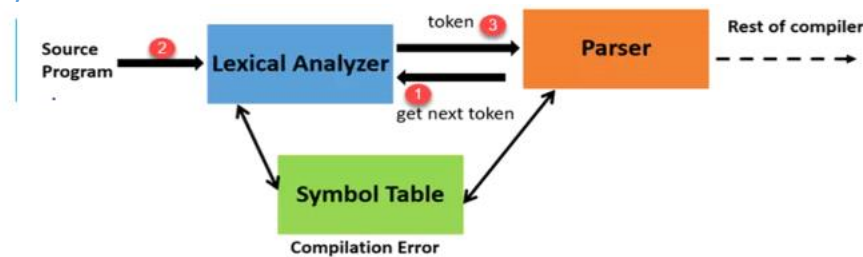
Converts int into float if, the value of data changes during operation.

Or if the value of data becomes float and if we only want int then semantic analyser converts it back to int.
Checks if the code is logically correct or not.

Lecture 5

13 September 2024 09:10 AM

Symbol table



Eg. `Int x =10; --> <id int>, <id x>, <id ==>, <id 10> <id ;>`

After performing the lexical analysing and after dividing into tokens, we put everything Into a table called Symbolic Table.

Example -

Identifier	Type	Scope	Memory Address	Additional Info
x	int	main	0x1000	Initialized to 10
y	float	main	0x1004	Initialized to 20.5
z	int	main	0x1008	Result of x + y

-> Symbol Table contains - name of identifier, the type of data, the scope of the data, its memory location and additional value associated with it.

Types -

To operate on any variable, the both variables should have the same data type. So we use Symbol Table to check the variable data type. If not we correct it through semantic analysis.