Fundamentals of Programming

By

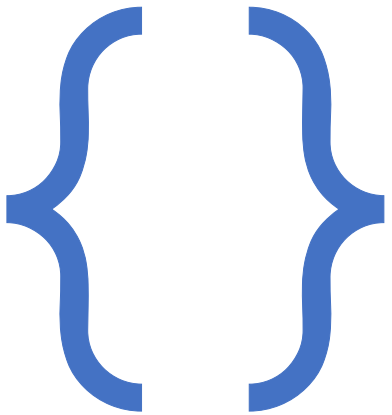
Prof Muhammad Iqbal Bhat

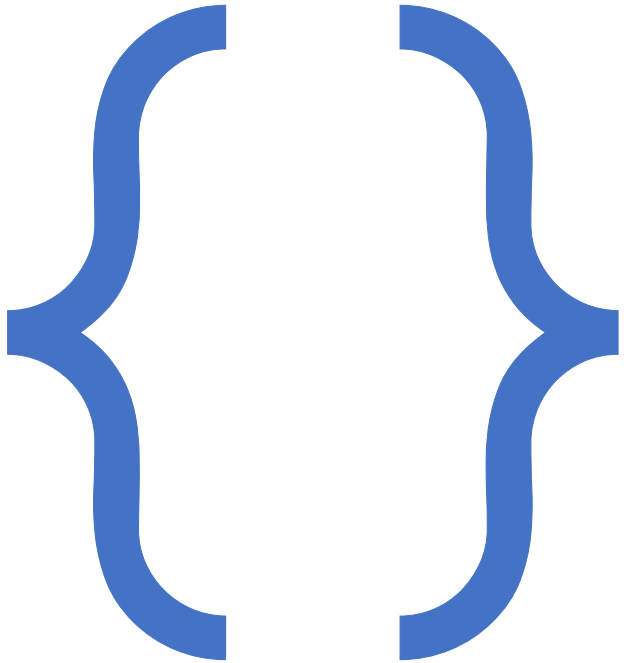
Government Degree College Beerwah

Topics

- Syntax and Semantics
- Source Code and Object Code:
- Datatypes
- Variables and Constants
- Declaration
- Structured Data Types
- Sequence Control
- Sequence Control between Statements

Syntax and Semantics





Syntax and Semantics

- Syntax and semantics are two fundamental concepts in programming languages. They define the structure and meaning of the language, respectively. Understanding the syntax and semantics of a programming language is essential for writing correct and efficient programs.

Syntax refers to the set of rules that govern the structure of a programming language

It defines the way in which programs are written and how the language is used.

A programming language syntax is typically defined by a set of grammar rules that dictate how different elements of the language can be combined to create valid statements.

Examples:

- Variable Declaration: Syntax: datatype variable_name; Example: int age;
- Function Declaration: Syntax: return_type function_name(parameter_list); Example: void print_message(char* message);
- Control Structure: Syntax: if (condition) { statement; } Example: if (age >= 18) { printf("You are an adult"); }

Semantics:

Semantics refers to the meaning of the programming constructs in a language.

It defines the behavior of the language and the way in which programs are executed.

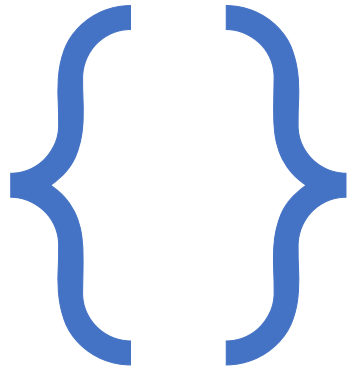
Semantics are closely related to syntax since they rely on the correct interpretation of the language's syntax.

Static vs Dynamic Semantics:

- Static semantics define the rules that govern the structure of a program before it is executed. They include rules for variable declarations, function definitions, and type checking.
- Dynamic semantics define the behavior of the program as it is executed. They include rules for control flow, memory management, and error handling.

Examples;

- Variable Initialization: Syntax: datatype variable_name = value; Semantics: Assigns the value to the variable during declaration. Example: int age = 20;
- Function Call: Syntax: function_name(argument_list); Semantics: Executes the function with the specified arguments. Example: print_message("Hello World");



Source Code and Object Code:

Source Code:

Source Code:

- Source code is the human-readable version of a program written in a high-level programming language.
- It is written using a text editor or an integrated development environment (IDE).
- The source code contains the instructions and algorithms that are used to create the program.

Object Code:

- Object code is the machine-readable version of a program generated by a compiler.
- The object code is in the form of binary code that the computer's CPU can understand and execute.
- The object code is generated by translating the source code into machine code that can be executed directly by the computer.

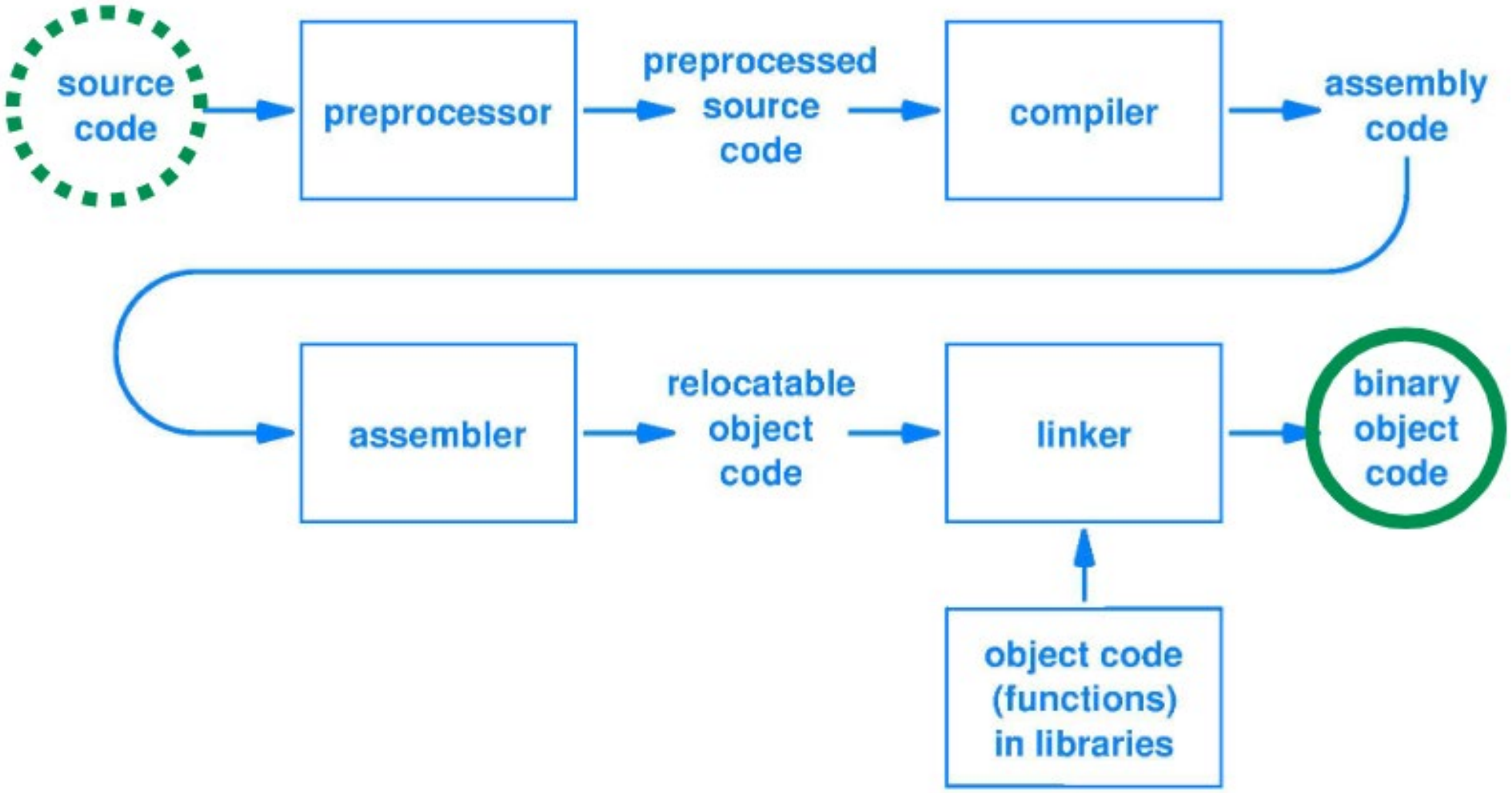
Source Code vs Object Code:

Source code is written in a high-level language, while object code is in machine code that the computer can directly execute.

Source code is written by humans, while object code is generated by a compiler or assembler.

Source code can be easily read and modified, while object code cannot be directly edited.

Source Code to Object Code Translation:



Data Types:

Datatypes define the type of data that can be stored and manipulated in a program.

Examples of datatypes include integers, floating-point numbers, characters, strings, and Boolean values.

Datatypes can be divided into two categories: primitive and composite.

Primitive datatypes are basic types, such as integers and characters, that cannot be further decomposed.

Composite datatypes are made up of multiple primitive datatypes, such as arrays and structures.

Data Types in C Language:

Data Type	Description	Size (in bytes)	Range
char	Character data type	1	-128 to 127 or 0 to 255 (unsigned)
int	Integer data type	2 or 4	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
short	Short integer data type	2	-32,768 to 32,767
long	Long integer data type	4 or 8	-2,147,483,648 to 2,147,483,647 or -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point data type	4	3.4E-38 to 3.4E+38
double	Double-precision floating-point data type	8	1.7E-308 to 1.7E+308
long double	Extended-precision floating-point data type	10 or 16	3.4E-4932 to 1.1E+4932
_Bool	Boolean data type	1	0 to 1

Constants:

A constant is a named value that cannot be changed throughout the program.

Constants are declared using the keyword "const", followed by the data type and name.

Constants are useful for values that should not be changed, such as mathematical constants or physical constants.

Constants in C Language:

Constant Type	Description	Example
Integer Constants	Integer values that cannot be changed during the program execution	42
Real Constants	Floating-point or double-precision values that cannot be changed during the program execution	3.14, 2.5e-3
Character Constants	Single characters enclosed in single quotes	'a', '$'
String Constants	A sequence of characters enclosed in double quotes	"Hello, World!"
Enumeration Constants	User-defined named values that represent a set of related integer constants	enum season { spring, summer, fall, winter };
Symbolic Constants	User-defined constants that are assigned a name, and are typically defined using the preprocessor directive #define	#define PI 3.14159

Declaration

The declaration refers to the process of introducing a variable or constant into a program.

A variable or constant must be declared before it can be used in a program.

Declaration involves specifying the data type, name, and, optionally, an initial value for the variable or constant.

Declarations in C Language:

Declaration Type	Description	Syntax
Variable Declaration	Declare a variable of a specific data type, with an optional initial value	datatype variable_name = initial_value;
Function Declaration	Declare a function with a return type, function name, and parameter list	return_type function_name(parameter_list);
Array Declaration	Declare an array with a specific data type, size, and optional initial values	datatype array_name[size] = {initial_values};
Pointer Declaration	Declare a pointer to a specific data type, with an optional initial value (address)	datatype *pointer_name = &variable_name;
Structure Declaration	Declare a structure with a specific name and member variables of different data types	struct structure_name { member_datatype1 member_variable1; member_datatype2 member_variable2; };
Union Declaration	Declare a union with a specific name and member variables that share the same memory space	union union_name { member_datatype1 member_variable1; member_datatype2 member_variable2; };
Enumeration Declaration	Declare an enumeration type with a specific name and named integer constants	enum enumeration_name { constant1, constant2, ... };

Questions?