

Structured Languages

Rahul Deodhar

You already know

- Basics of computer
- Database
 - FoxPro / Oracle
 - DBMS / RDBMS
- Operating System
 - DOS / Novel/Unix
- Applications (Spreadsheets / Word processor)
- Basics of programming languages
 - C,
- Compiler/Interpreter

Syllabus

- Basics
 - Basic data types
 - Arrays
 - Functions
 - Pointers & References
 - Structures
 - Constructor
Destructor
 - Operator Overloading
- OOPS basics
 - Why OOPs?
 - Advantages
- OOPs advanced
 - Classes & Objects
 - Inheritance
 - Polymorphism
 - Public and private
 - Functions
 - Data variables
 - Container classes
 - Virtual functions

Syllabus

- Java Basics
 - Why Java
 - Advantages
 - Platform independence
- Language basics
 - Java Application string & string buffer
 - Input / Output
 - Syntax
 - Data types
 - Functions
- Java Exception handling
- Multi-threading sessions
- Java .net (Networking and Security Networking with URLs)
- Database access JDBC and Sql.
- JFC swing
- Java 20 drag and drop

Choice!

Types of Programming languages

Machine Language

- Strings of numbers giving machine specific instructions
- Example:

Assembly Language

- English-like abbreviations representing elementary computer operations (translated via assemblers)
- Example:

High Level Language

- Codes similar to everyday English
- Use mathematical notations (translated via compilers)
- Example:

```
grossPay = basePay + overTimePay
```

Structured Programming

Structured Programming

- Structured programming (1960s)
 - Disciplined approach to writing programs
 - Clear, easy to test and debug, and easy to modify
- Pascal
 - 1971: Niklaus Wirth
- Ada
 - 1970s - early 1980s: US Department of Defense (DoD)
 - Multitasking
 - Programmer can specify many activities to run in parallel
- C

Questions?

Some High Level Languages

Brief Introduction

Some High-level Languages

- FORTRAN
 - FORmula TRANslator
 - 1954-1957: IBM
 - Complex mathematical computations
 - Scientific and engineering applications
- COBOL
 - COmmon Business Oriented Language
 - 1959: computer manufacturers, government and industrial computer users
 - Precise and efficient manipulation of large amounts of data
 - Commercial applications
- Pascal
 - Prof. Niklaus Wirth
 - Academic use

History of C

- C
 - Evolved by Ritchie
 - Used to develop UNIX
 - Used to write modern operating systems
 - Hardware independent (portable)
 - By late 1970's C had evolved to "Traditional C"
- Standardization
 - Many slight variations of C existed, and were incompatible
 - Committee formed to create a "unambiguous, machine-independent" definition
 - Standard created in 1989, updated in 1999

The C Standard Library

- C programs consist of pieces/modules called functions
 - A programmer can create his own functions
 - Advantage: the programmer knows exactly how it works
 - Disadvantage: time consuming
 - Programmers will often use the C library functions
 - Use these as building blocks
 - Avoid re-inventing the wheel
 - If a premade function exists, generally best to use it rather than write your own
 - Library functions carefully written, efficient, and portable

Object Technology

- Reusable software components that model items in the real world
- Meaningful software units
 - Date objects, time objects, paycheck objects, invoice objects, audio objects, video objects, file objects, record objects, etc.
 - Any noun can be represented as an object
- Very reusable
- More understandable, better organized, and easier to maintain than procedural programming
- Favor modularity

C++

- Superset of C developed by Bjarne Stroustrup at Bell Labs
- "Spruces up" C, and provides object-oriented capabilities
- Object-oriented design very powerful
 - 10 to 100 fold increase in productivity
- Dominant language in industry and academia
- Because C++ includes C, some feel it is best to master C, then learn C++

Java

- Java is used to
 - Create Web pages with dynamic and interactive content
 - Develop large-scale enterprise applications
 - Enhance the functionality of Web servers
 - Provide applications for consumer devices (such as cell phones, pagers and personal digital assistants)

Visual Basic

- BASIC
 - Beginner's All-Purpose Symbolic Instruction Code
 - Mid-1960s: Prof. John Kemeny and Thomas Kurtz (Dartmouth College)
- Visual Basic
 - 1991
 - Result of Microsoft Windows graphical user interface (GUI)
 - Developed late 1980s, early 1990s
 - Powerful features
 - GUI, event handling, access to Win32 API, object-oriented programming, error handling
 - Visual Basic .NET

Visual C++

- Visual C++
 - Microsoft's implementation of C++
 - Includes extensions
 - Microsoft Foundation Classes (MFC)
 - Common library
 - GUI, graphics, networking, multithreading, ...
 - Shared among Visual Basic, Visual C++, C#
- .NET platform
 - Web-based applications
 - Distributed to great variety of devices
 - Cell phones, desktop computers
 - Applications in disparate languages can communicate

C#

- C#
 - Pronounced “C-Sharp”
 - Anders Hejlsberg and Scott Wiltamuth (Microsoft)
 - Designed specifically for .NET platform
 - Roots in C, C++ and Java
 - Easy migration to .NET
 - Event-driven, fully object-oriented, visual programming language
 - Integrated Development Environment (IDE)
 - Create, run, test and debug C# programs
 - Rapid Application Development (RAD)
 - Language interoperability

Basics of Programming Languages

Basics of programming language

- Data types
- Syntax
 - Operations
 - Special characters
 - definitions
 - Calls
 - References
 - arguments
- Input / Output
- Functions and Loops
- Exception handling

Basics of Program

How program works

- Input
- Operations
 - Assumptions or standard variables
 - Algorithm
- Output
 - Nature of output
 - Name of variable
 - Data-type
 - Output format

Program basics

- Headers
- Definitions
- Inputs
- Functions etc.
- Outputs
- Other elements
 - Comments (line comment / block comment)

Program

- Pre-work
 - Define objectives
 - Inputs
 - Outputs
 - Algorithm
- Write program
- Compile and debug
- Executable program

A typical program working

Basic C++ Program

```
1 // A basic C++ program
2 // This is a line comment
3 #include <iostream> // Preprocessor Directive
4
5 // function main begins program execution
6 int main()
7 {
8     std::cout << "Welcome to C++!\n";
9
10    return 0; // indicate that program ended successfully
11
12 } // end function main
```

```
Welcome to C++!
```

Typical program working

1. Edit

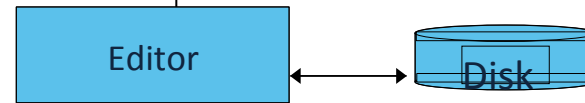
2. Preprocess

3. Compile

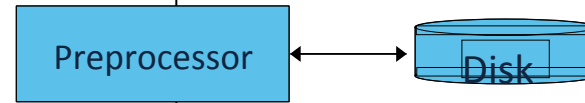
4. Link

5. Load

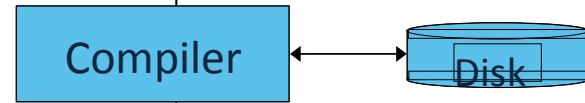
6. Execute



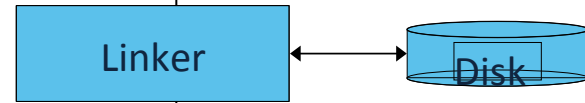
Program is created in the editor and stored on disk.



Preprocessor program processes the code.



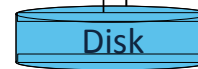
Compiler creates object code and stores it on disk.



Linker links the object code with the libraries



Loader puts program in memory.



CPU takes each instruction and executes it, possibly storing new data values as the program executes.