BITS Pilani

Lecture 5

Imperative programming Paradigm

User’s perspective of a program

- Solves a problem
- Processes data and does computations
- Set of instructions
- Written in high level language
- Collection of functions
- Machine independent
- Should produce correct results
- Should be fast in completing all computations

Designer’s perspective of a program

- A program is viewed as a sentence.
- A sentence is derived from the underlying grammar
- There are infinite number of finite programs
- A program has a syntactic structure
- A program should efficiently utilize the hardware resources
- A program should produce correct results

Behavior of a program

- Static: Source code is a simple sequence of instructions written in a file

  if (x<10) y=x+z; else y=x-z*2;

- Dynamic: Program in execution

  read x as 13 (at run time)

  y=x-z*2   (executed at run time)

  [y=x+z is not executed]

Program

- A program is a concise representation of the computation that occurs when the program runs.
- A large sequence of computations can be expressed in few lines of code.
- Computations can be repetitive to a known amount of time or to an unknown condition that satisfies at any point of time.

Difference between static form of program and its dynamic behavior

- Program in execution represents dynamic behavior of the static source code
- Dynamic computations are usually longer than the static source code

static code: for(i=0; i<100;i++) x=x+i;

dynamic behavior

```
        i=0
         x=x+i
        i=i+1
        x=x+I
        ……
        ….
        i=i+1                    //i=99
        x=x+1
```

Difference between static form of program and its dynamic behavior

- Static program may be longer but its dynamic computation can be smaller

Static code: if (x<10) x=x+y; else x = x-y;

Dynamic computation: x=x-y (read x as 13)

Structured programming

- The structure of the program text explains what the program does.
- Advantages of a structured program
    - Easy to read and modify
    - Easy to tune for efficiency

Imperative programming paradigm

- Imperative languages are **action** oriented in which the computations are viewed as sequence of actions.
- An action refers to **change in value of a variable** (read(a), a++, a=b+c or a=call_sum(b,c))
- A program state is defined as a **tuple** of variable values (a,b,c) which changes at each computation
- **Efficiency** of a program is the key to imperative paradigm

Imperative programs

- Programs written in imperative language consist of instructions that change the state of the program.
- Programs are imperative in the grammatical sense of imperative verbs (actions) that express a command

Imperative programming and Von Neumann architecture

- John Von Neumann documented the basic concepts of the stored program computers
- Instructions in an imperative language are written keeping in view the underlying machine details such as memory and processor
- The first imperative language was assembly language, then came FORTRAN, ALGOL, COBOL

Turing completeness and imperative languages

- A language is Turing complete if it can be used to implement any algorithm (Alan Turing)
- Imperative languages are Turing complete if they support integers, arithmetic operators, assignment, sequencing, looping and branching.

Program as change of state

- Memory locations as key to imperative programs
- Variable names are bound to memory locations
- Variable names and their values bound at run time
- Computation and execution flow statements

Modular structure of a program

- A program must have a driver function.
- There can be one or more functions in a program
- There may not be any other module other than the driver function.
- A module contains statements.
- Statements can be of different types such as declaration, I/O, assignment, function call etc.

Any restriction on functions usage in a program?

- User’s view: A function should be defined before its call
- Designer’s view: support for parameter passing technique, …
- Compiler developer’s view: Capture the definition and call through grammar rules.

What is a construct in a programming language?

- A construct is a piece of code that is derived (constructed) from the underlying grammar that defines the programming language.
- Some important constructs are expressions, statements (if, switch, for while etc.), modules, datatypes (arrays, records, strings, matrices etc.), parameter list,

An expression construct

- User view
  - Precedence of operator- what we want to compute (Plus-15th century, Multiplication-17th century)
  - Example: 12+3*2 is 18
- Designer view
  - Precedence of operators- what is imposed via grammar rules
  - Example whether 12+3*2 should be 30 or 18?