Unit 2: Programming Technique

2.1. Introduction to Program Technique

- C is a general-purpose, procedural computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system.
- C is the most widely used computer language.
- It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

2.2. Top down and Bottom up Approach

Top down Approach

- The basic task of a top-down approach is to divide the problem into tasks and then divide tasks into smaller sub-tasks and so on.
- Each part of it then refined into more details, defining it in yet more details until the entire specification is detailed enough to validate the model.
- It break the problem into parts, Then break the parts into parts soon and now each of part will be easy to do.

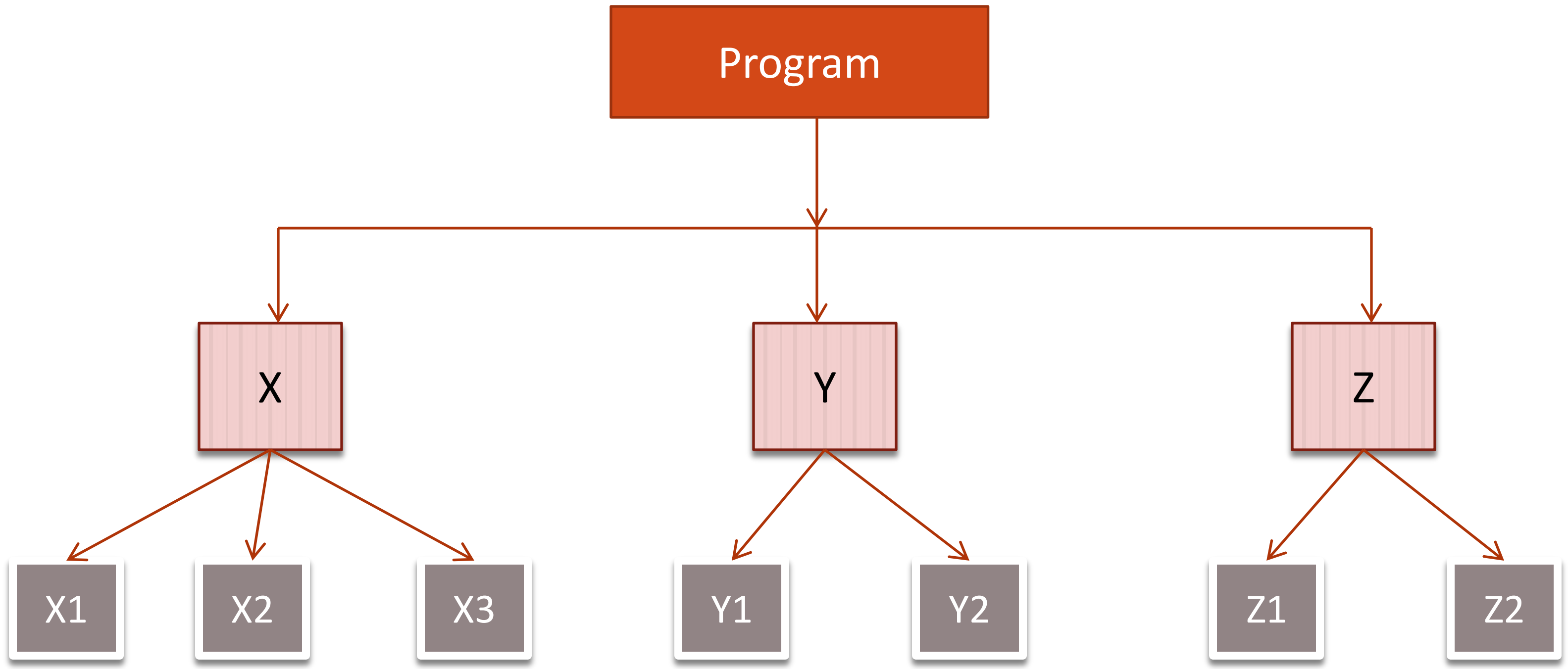
2.2. Top down and Bottom up Approach

Top down Approach

- C programming language supports this approach for developing projects.
- It is always good idea that decomposing solution into modules in a hierarchal manner.
- In this approach, first we develop the main module and then the next level modules are developed.
- This procedure is continued until all the modules are developed.

2.2. Top down and Bottom up Approach

Top down Approach



2.2. Top down and Bottom up Approach

Top down Approach

Advantages:

- Breaking problems into parts help us to identify what needs to be done.
- At each step of refinement new parts will become less complex and therefore easier to solve.
- Parts of solution may turn out to be reusable.
- Breaking problems into parts allows more than one person to solve the problem.

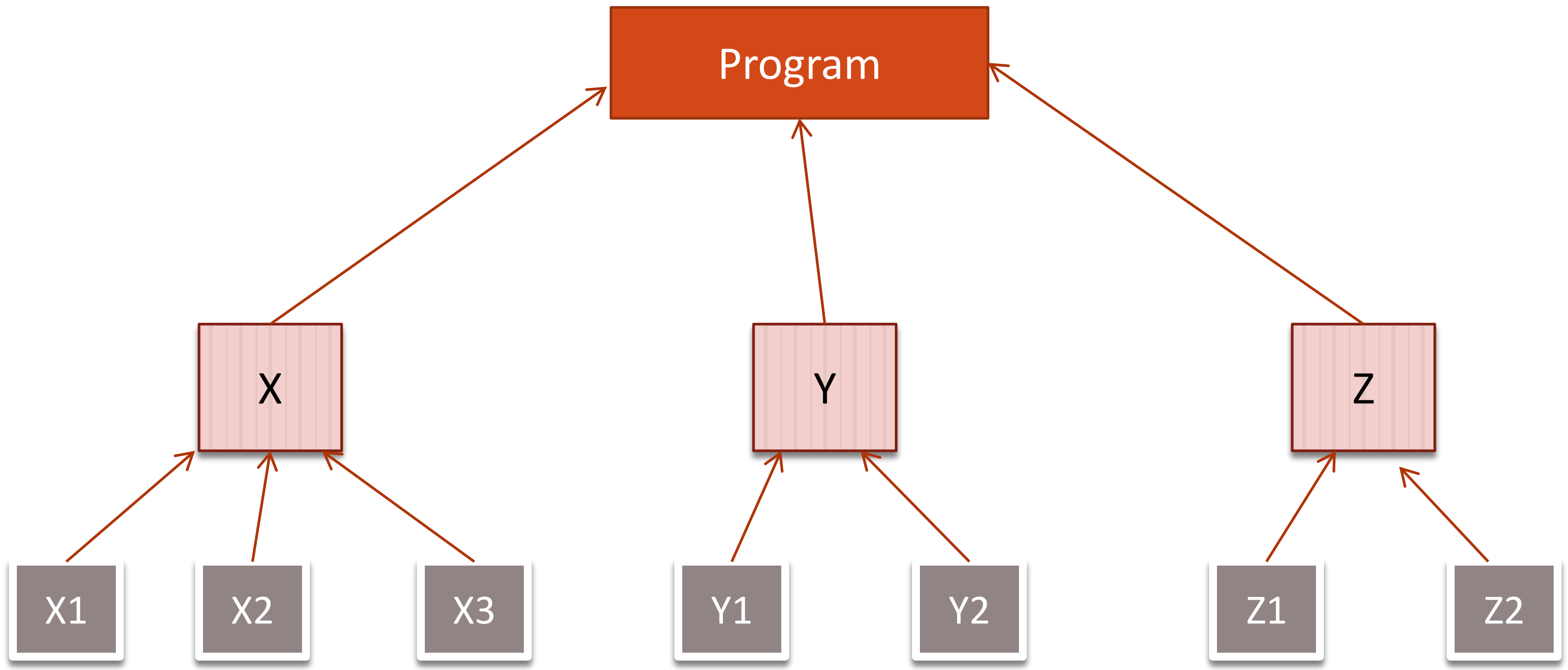
2.2. Top down and Bottom up Approach

Bottom-Up Design Model:

- In this design, individual parts of the system are specified in details.
- The parts are the linked to form larger components, which are in turn linked until a complete system is formed.
- This approach is exactly opposite to the top-down approach.
- In this approach, bottom level modules developed first (Lower level module developed, tested and debugged).
- Then the next module developed, tested and debugged.
- This process is continued until all modules have been completed.
- This approach is good for reusability of code.
- Object oriented language such as C++ or java uses bottom up approach where each object is identified first.

2.2. Top down and Bottom up Approach

Bottom-Up Design Model:



2.2. Top down and Bottom up Approach

Bottom-Up Design Model:

Advantage:

- Make decisions about reusable low level utilities then decide how there will be put together to create high level construct.
- Contrast between Top down design and bottom up design.

2.2. Top down and Bottom up Approach

S.NO.	TOP DOWN APPROACH	BOTTOM UP APPROACH
1.	In this approach We focus on breaking up the problem into smaller parts.	In bottom up approach, we solve smaller problems and integrate it as whole and complete the solution.
2.	Mainly used by structured programming language such as COBOL, Fortan, C etc.	Mainly used by object oriented programming language such as C++, C#, Python.
3.	Each part is programmed separately therefore contain redundancy.	Redundancy is minimized by using data encapsulation and data hiding.
4.	In this the communications is less among modules.	In this module must have communication.
5.	It is used in debugging, module documentation, etc.	It is basically used in testing.
6.	In top down approach, decomposition takes place.	In bottom up approach composition takes place.
7.	In this top function of system might be hard to identify.	In this sometimes we can not build a program from the piece we have started.
8.	In this implementation details may differ.	This is not natural for people to assemble.

2.3. Cohesion and Coupling

Cohesion

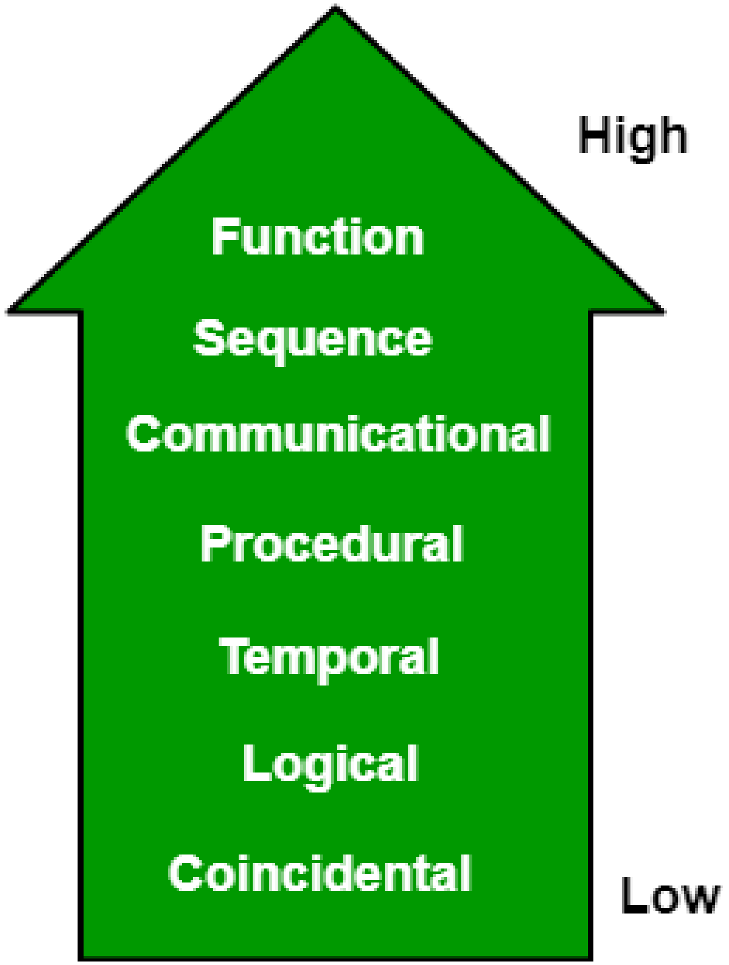
- Cohesion in programming development determines the degree to which the element inside a module belongs together.
- It measures the strength of relationships between pieces of functionality within a given module
- A good software design will have high cohesion.

Types of Cohesion

1. Functional cohesion (Most Required)
2. Sequential cohesion
3. Communicational cohesion
4. Procedural cohesion
5. Temporal cohesion
6. Logical cohesion
7. Coincidental cohesion (Least Required)

2.3. Cohesion and Coupling

Cohesion



2.3. Cohesion and Coupling

Cohesion

1.Functional cohesion

- Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

2.Sequential cohesion

- A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

2.3. Cohesion and Coupling

Cohesion

3. Communicational Cohesion

- A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

4. Procedural cohesion

- A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal. e.g. the algorithm for decoding a message.

2.3. Cohesion and Coupling

Cohesion

5. Temporal cohesion

- When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

6. Logical cohesion

- A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

7. Coincidental cohesion

- A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

2.3. Cohesion and Coupling

Coupling

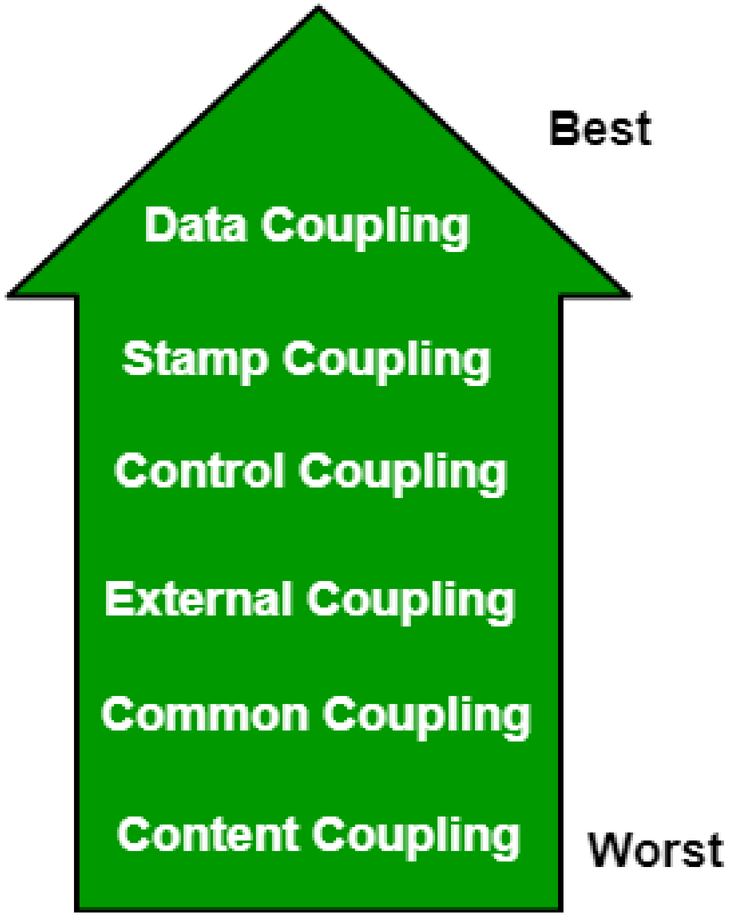
- Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Types of Coupling:

- Data Coupling:
- Stamp Coupling
- Control Coupling
- External Coupling
- Common Coupling
- Content Coupling

2.3. Cohesion and Coupling

- **Coupling**



2.3. Cohesion and Coupling

Coupling

1. Data Coupling:

- If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled.

2. Stamp Coupling

- Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc.
- For example, passing structure variable in C or object in C++ language to a module.

2.3. Cohesion and Coupling

Coupling

3. Control Coupling:

- Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.
- Example- sort function that takes comparison function as an argument.

4. External Coupling:

- In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

2.3. Cohesion and Coupling

Coupling

5. Common Coupling:

- Two modules are common coupled if they share information through some global data items.
- The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.

6. Content Coupling:

- In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module.
- This is the worst form of coupling and should be avoided.

2.3. Cohesion and Coupling

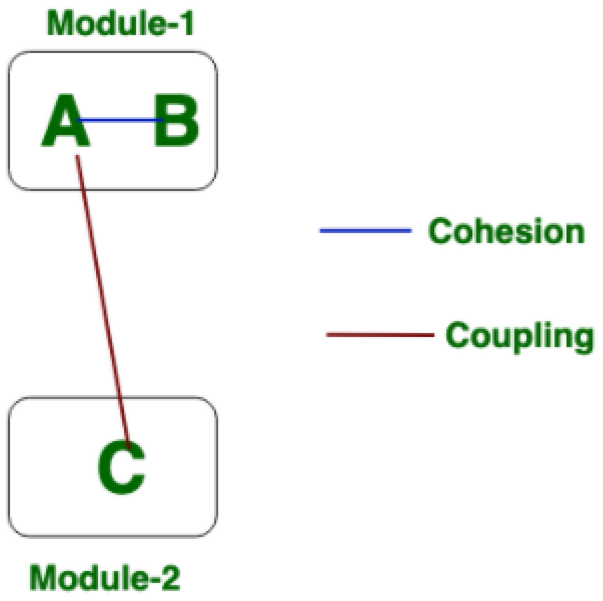
Differences between cohesion and coupling

Cohesion:

- Cohesion is the indication of the relationship within module. It is concept of intra-module. Cohesion has many types but usually highly cohesion is good for software.

Coupling:

- Coupling is also the indication of the relationships between modules. It is concept of Inter-module. Coupling has also many types but usually low coupling is good for software.



Differences between cohesion and coupling

COHESION	COUPLING
Cohesion is the concept of intra module.	Coupling is the concept of inter module.
Cohesion represents the relationship within module.	Coupling represents the relationships between modules.
Increasing in cohesion is good for software.	Increasing in coupling is avoided for software.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
Highly cohesive gives the best software.	Where as loosely coupling gives the best software.
In cohesion, module focuses on the single thing.	In coupling, modules are connected to the other modules.

2.4.Structured Programming

A C program is divided into different sections. There are six main sections to a basic c program.

The six sections are,

- Documentation
- Link
- Definition
- Global Declarations
- Main functions
- Subprograms

2.4.Structured Programming

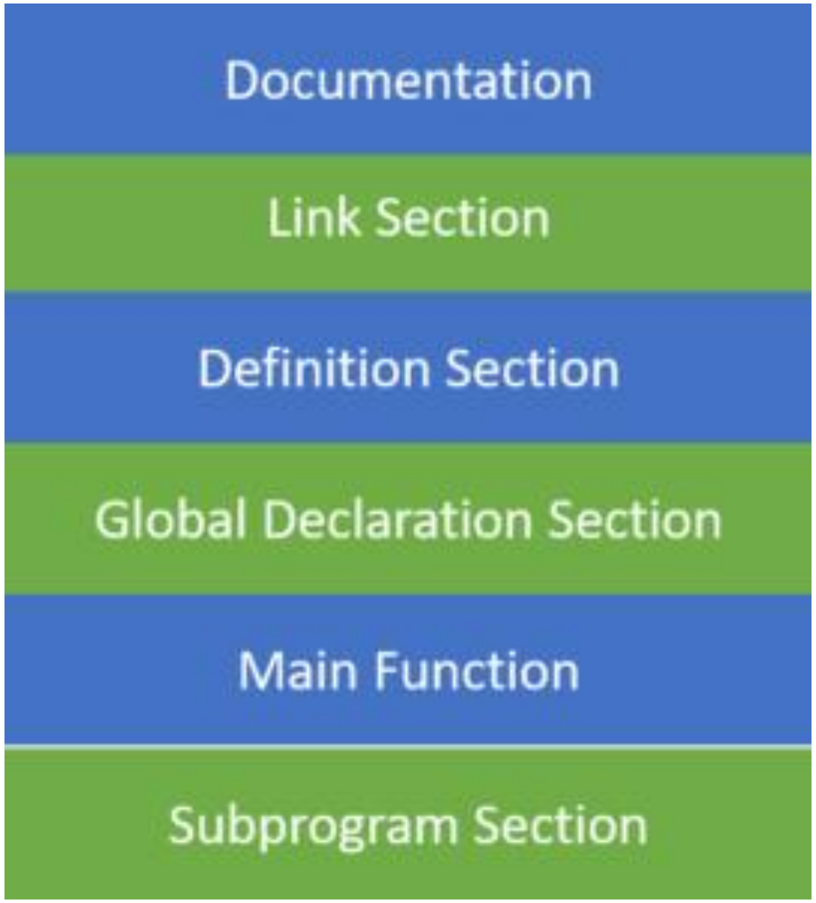


Figure: Basic Structure Of C Program

2.4.Structured Programming

1. Documentation Section

- The documentation section is the part of the program where the programmer gives the details associated with the program.
- It usually gives the name of the program, the details of the author and other details like the time of coding and description. It gives anyone reading the code the overview of the code.

Example

```
/**
* File Name: Helloworld.c
* Author: Manthan Naik
* date: 09/08/2019
* description: a program to display hello world
*            no input needed
*/
```

Moving on to the next bit of this basic structure of a C program article,

2.4.Structured Programming

2. Link Section

- This part of the code is used to declare all the header files that will be used in the program.
- This leads to the compiler being told to link the header files to the system libraries.

Example

```
#include<stdio.h>
```

Moving on to the next bit of this basic structure of a C program article,

2.4.Structured Programming

3. Definition Section

- In this section, we define different constants. The keyword define is used in this part.

Example

```
#define PI=3.14
```

Moving on to the next bit of this basic structure of a C program article,

2.4.Strucuted Programming

4. Global Declaration Section

- This part of the code is the part where the global variables are declared.
- All the global variable used are declared in this part.
- The user-defined functions are also declared in this part of the code.

Example

```
float area(float r);
int a=7;
```

Moving on to the next bit of this basic structure of a C program article,

2.4.Structured Programming

5. Main Function Section

- Every C-programs needs to have the main function. Each main function contains 2 parts. A declaration part and an Execution part. The declaration part is the part where all the variables are declared. The execution part begins with the curly brackets and ends with the curly close bracket. Both the declaration and execution part are inside the curly braces.

Example

```
int main(void)
{
    int a=10;
    printf(" %d", a);
    return 0;
}
```

Moving on to the next bit of this basic structure of a C program article,

2.4.Structured Programming

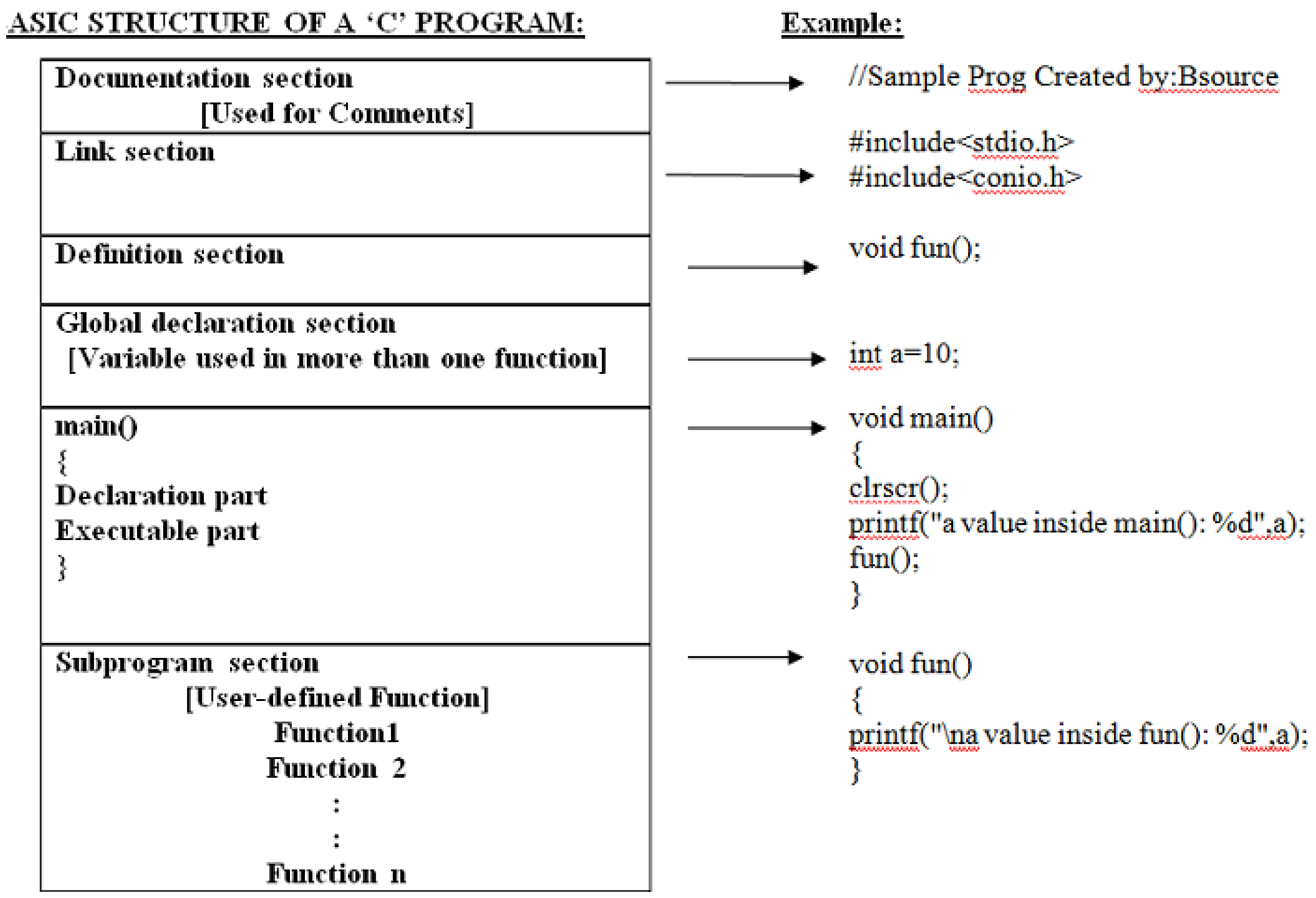
6. Sub Program Section

- All the user-defined functions are defined in this section of the program.

Example

```
int add(int a, int b)
{
    return a+b;
}
```

2.4.Structured Programming



ASIC STRUCTURE OF A 'C' PROGRAM:

Documentation section
[Used for Comments]

Link section

Definition section

Global declaration section
[Variable used in more than one function]

main()
{
Declaration part
Executable part
}

Subprogram section
[User-defined Function]
Function1
Function 2
:
:
Function n

Example:

```
//Sample Prog Created by:Bsource

#include<stdio.h>
#include<conio.h>

void fun();

int a=10;

void main()
{
clrscr();
printf("a value inside main(): %d",a);
fun();
}

void fun()
{
printf("\na value inside fun(): %d",a);
}
```

2.4.Structured Programming

Features of structured programming

- The structured program consists of well structured and separated modules.
- The entry and exit in a Structured program is a single-time event.
- It means that the program uses single-entry and single-exit elements.
- Therefore a structured program is well maintained, neat and clean program.
- This is the reason why the Structured Programming Approach is well accepted in the programming world.

2.4.Structured Programming

Advantages of Structured Programming Approach:

- Easier to read and understand
- User Friendly
- Easier to Maintain
- Mainly problem based instead of being machine based
- Development is easier as it requires less effort and time
- Easier to Debug
- Machine-Independent, mostly.

2.4.Structured Programming

Disadvantages of Structured Programming Approach:

- Since it is Machine-Independent, So it takes time to convert into machine code.
- The converted machine code is not the same as for assembly language.
- The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
- Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

2.5. Deterministic and Non-Deterministic Technique

- In **deterministic technique**, for a given particular input, the computer will always produce the same output going through the same states but in case of **non-deterministic technique**, for the same input, the compiler may produce different output in different runs.
- In fact non-deterministic techniques can’t solve the problem in polynomial time and can’t determine what is the next step.
- The non-deterministic techniques can show different behaviors for the same input on different execution and there is a degree of randomness to it.

2.5. Deterministic and Non-Deterministic Technique

$$\begin{tikzcd}
 & q_1 \\
q_0 \arrow[ur, "1"] \arrow[dr, "0"'] & \\
 & q_2
\end{tikzcd}$$

Deterministic Algorithm

$$\begin{tikzcd}
 & q_1 \\
q_0 \arrow[ur, "0"] \arrow[dr, "0"'] & \\
 & q_2
\end{tikzcd}$$

Non-Deterministic Algorithm

2.5. Deterministic and Non-Deterministic Technique

DETERMINISTIC ALGORITHM	NON-DETERMINISTIC ALGORITHM
For a particular input the computer will give always same output.	For a particular input the computer will give different output on different execution.
Can solve the problem in polynomial time.	Can’t solve the problem in polynomial time.
Can determine the next step of execution.	Cannot determine the next step of execution due to more than one path the algorithm can take.

2.6. Iterative and Recursive Logic

- A program is called recursive when an entity calls itself.
- A program is call iterative when there is a loop (or repetition).
- In simple terms, an *iterative* function is one that loops to repeat some part of the code, and a *recursive* function is one that calls itself again to repeat the code.
- Using a simple *for* loop to display the numbers from one to ten is an *iterative* process.
- Examples of *simple* recursive processes aren't easy to find, but creating a school timetable by rearranging the lessons, or solving the *Eight Queens Problem* are common examples.
- *Iteration* and *recursion* are probably best explained using an example of something that can be done using either technique.

2.6. Iterative and Recursive Logic

PROPERTY	RECURSION	ITERATION
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).

2.6. Iterative and Recursive Logic

Below are the detailed example to illustrate the difference between the two:

- **Time Complexity:** Finding the Time complexity of Recursion is more difficult than that of Iteration.
  - **Recursion**: Time complexity of recursion can be found by finding the value of the nth recursive call in terms of the previous calls. Thus, finding the destination case in terms of the base case, and solving in terms of the base case gives us an idea of the time complexity of recursive equations. Please see Solving Recurrences for more details.
  - **Iteration**: Time complexity of iteration can be found by finding the number of cycles being repeated inside the loop.

2.6. Iterative and Recursive Logic

- **Usage:** Usage of either of these techniques is a trade-off between time complexity and size of code. If time complexity is the point of focus, and number of recursive calls would be large, it is better to use iteration. However, if time complexity is not an issue and shortness of code is, recursion would be the way to go.
  - **Recursion**: Recursion involves calling the same function again, and hence, has a very small length of code. However, as we saw in the analysis, the time complexity of recursion can get to be exponential when there are a considerable number of recursive calls. Hence, usage of recursion is advantageous in shorter code, but higher time complexity.
  - **Iteration**: Iteration is repetition of a block of code. This involves a larger size of code, but the time complexity is generally lesser than it is for recursion.

2.6. Iterative and Recursive Logic

- **Overhead:** Recursion has a large amount of Overhead as compared to Iteration.
  - **Recursion**: Recursion has the overhead of repeated function calls, that is due to repetitive calling of the same function, the time complexity of the code increases manifold.
  - **Iteration**: Iteration does not involve any such overhead.
- **Infinite Repetition:** Infinite Repetition in recursion can lead to CPU crash but in iteration, it will stop when memory is exhausted.
  - **Recursion**: In Recursion, Infinite recursive calls may occur due to some mistake in specifying the base condition, which on never becoming false, keeps calling the function, which may lead to system CPU crash.
  - **Iteration**: Infinite iteration due to mistake in iterator assignment or increment, or in the terminating condition, will lead to infinite loops, which may or may not lead to system errors, but will surely stop program execution any further.

2.7. Modular Designing and Programming

- Modular programming is the process of subdividing a computer program into separate sub-programs.
- A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system.
- Some programs might have thousands or millions of lines and to manage such programs it becomes quite difficult as there might be too many of syntax errors or logical errors present in the program, so to manage such type of programs concept of **modular programming** approached.
- Each sub-module contains something necessary to execute only one aspect of the desired functionality.
- Modular programming emphasis on breaking of large programs into small problems to increase the maintainability, readability of the code and to make the program handy to make any changes in future or to correct the errors.

2.7. Modular Designing and Programming

Points which should be taken care of prior to modular program development:

- Limitations of each and every module should be decided.
- In which way a program is to be partitioned into different modules.
- Communication among different modules of the code for proper execution of the entire program.

2.7. Modular Designing and Programming

Advantages of Using Modular Programming Approach

- **Ease of Use :**This approach allows simplicity, as rather than focusing on the entire thousands and millions of lines code in one go we can access it in the form of modules. This allows ease in debugging the code and prone to less error.
- **Reusability :**It allows the user to reuse the functionality with a different interface without typing the whole program again.
- **Ease of Maintenance :** It helps in less collision at the time of working on modules, helping a team to work with proper collaboration while working on a large application.

Unit 2

Finished