CS -01

PROBLEM SOLVING METHODOLOGIES AND PROGRAMMING IN C

By Rachel

UNIT 1 – PART 2

TOPIC 3

INTRODUCTION OF C LANGUAGE (HISTORY AND OVERVIEW)

History of C Language

C is a structured, high-level, machine independent language.
It runs under a variety of operating system and hardware platforms.

BCPL – Basic Combined Programming Language

ANSI – American National Standards Institute

ISO – International Standards Organization

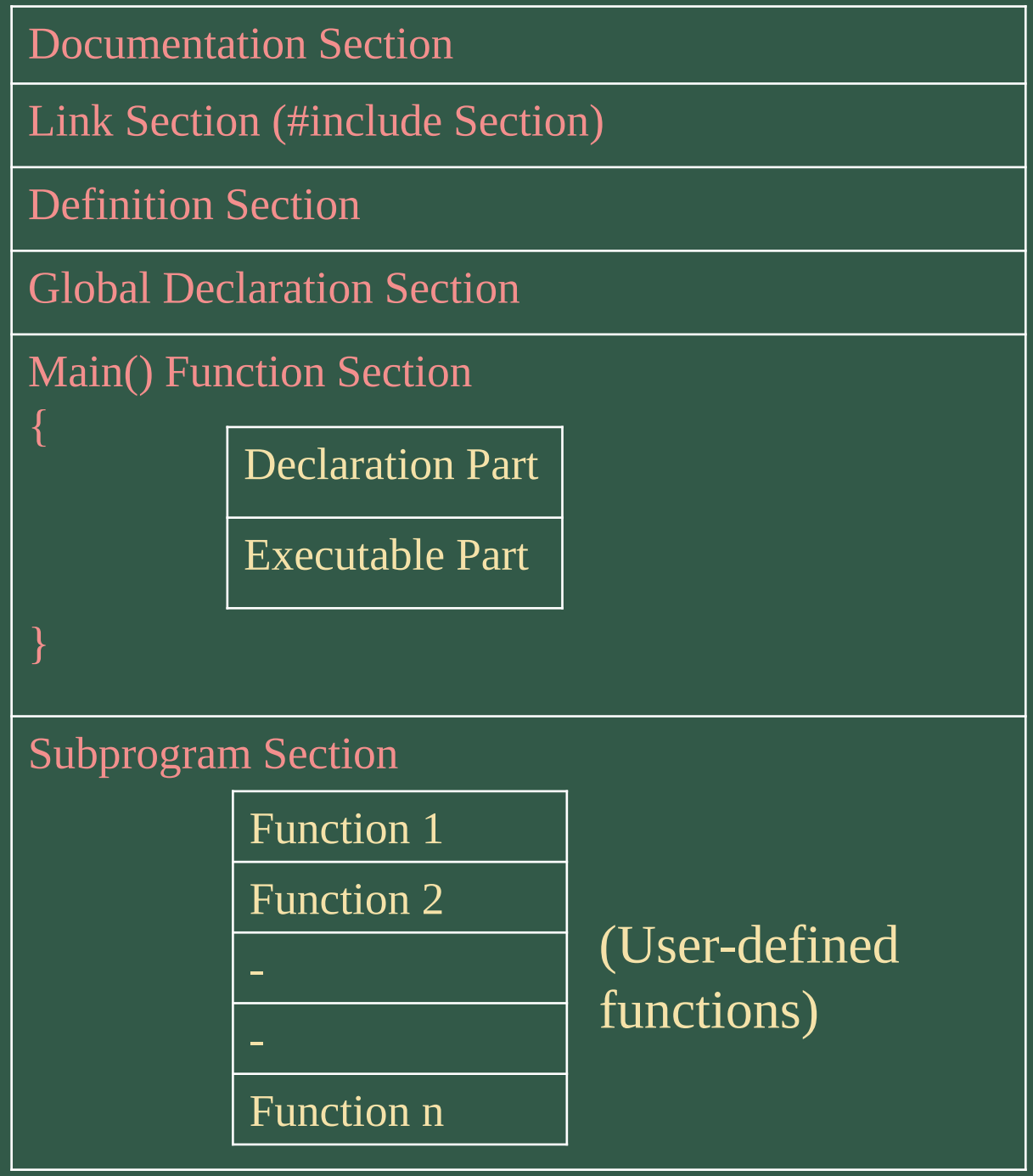
History of C

- **C** was evolved from ALGOL, BCPL and B.
- **C** was developed by **Dennis Ritchie** at the Bell Laboratories in 1972.
- Added new features and concepts like "data types".
- It was developed along with the UNIX operating system.
- It was strongly integrated with the UNIX operating system.
- In 1983 American National Standards Institute (ANSI) appointed a technical committee to define a standard for C. The committee approved a version of C in December **1989** which is now known as **ANSI C**.
- In 1990 International Standards Organization (ISO) has approved C and this version of C is referred to as **C89.**

Features of C Language

- It has rich set of built in functions.
- Operators can be used to write any program.
- C is suitable for making system software and business packages.
- C programs are efficient and fast.
- It has variety of data types and powerful operators.
- There are 32 keywords in C.
- It is portable language. Program written in one computer can be used by others.
- It is well suited for structured programming.
- It can be viewed as a group of building blocks called functions.

Structure of C Program



1. Documentation Section

- Also called Comments section.
- Helps to understand what the complete program is about.
- C compiler doesn’t execute any statement in comment.
- The format is /*……*/
- Example :

```
/* This is an example program */
```

2. Link Section

- It provides instruction to the compiler to link the functions of the Header files from the system library.
- Header files contains some particular functions which are necessary for the execution of C program.
- Example : #include <stdio.h>
  This tells the compiler to include information about the standard input/output library file.

3. Definition Section

- It defines all symbolic constants.
- Example : #define PI 3.14
  This defines the value of PI as 3.14 which will remain the same throughout the program.

4. Global Declaration Section

- Some variables can be used in more than one function. Such variables are called global variables. They should be declared in this section.
- Example :

```
void interest(int i, int p, float r);
```

This tells that the function named interest will have 2 integer values named i and p and a float value named r and which will not return any value to the main function.

5. The main() function Section

- Every C program must have a specific function named main. The execution starts at the beginning of the main function.
- It has two parts namely declaration part and execution part.
- Declaration part - To declare all variables used in the executable part.
- Executable part - The program execution begins at the opening brace ({) and ends with the closing brace (}).
- All statements in the declaration part and executable part ends with the semicolon (;).

6. Sub-program Section

- It contains all the User Defined Functions called in main().
- User Defined Functions are generally placed immediately after main(), but they may appear in any order.

Example C Program

```c
#include <stdio.h>
main()
{
    /*Printing Begins*/
    printf("Hello! Welcome to the C world!");
    /*Printing ends*/
}
```

Program Output

```
Hello! Welcome to the C world!
```

TOPIC 4

DIFFERENCE BETWEEN TRADITIONAL C AND MODERN C

Traditional C	Modern C
1. Traditional C was developed by Dennis Ritchie at Bell Labs in 1970s.	1. Modern C also called C99 was developed by Standardization Committee in 1999.
2. It has limited set of keywords and syntax elements.	2. It has enhanced and standardized set of keywords and syntax elements.
3. It is less compatible with limited standards.	3. It is much compatible as it allows developers to write portable code that can be easily compiled and executed on various platforms.
4. It will not support common language specification.	4. It supports common language specification and programmers from different backgrounds can work on the same codebase.
5. It supports basic data types such as integers, floating point numbers and characters.	5. It supports additional data types like void and wchar_t(wide characters)
6. It has basic type checking feature.	6. It has improved type checking feature with stricter rules for type conversions and promotions.
7. Function prototypes are not mandatory and a function could be defined without explicitly declaring its parameters.	7. Function prototypes are mandatory which enhances code readability and facilitates error detection during compilation.