Structure of the C Program

C program's fundamental composition comprises six distinct sections, facilitating readability, modification, documentation, and comprehension in a specific format. Adhering to the outlined structure below is imperative for the successful compilation and execution of a C program.

One of the key features of C is its structured programming paradigm. This means that programs are organized into logical blocks or functions, making it easier to manage and understand the code. This structure not only improves readability but also enhances maintainability, as individual components can be modified or updated without affecting the entire program.

In C, a program typically consists of functions, data types, variables, control structures, and header files. Functions are blocks of code that perform specific tasks and can be called from other parts of the program. Data types define the type of data that a variable can hold, such as integers, floating-point numbers, characters, and more.

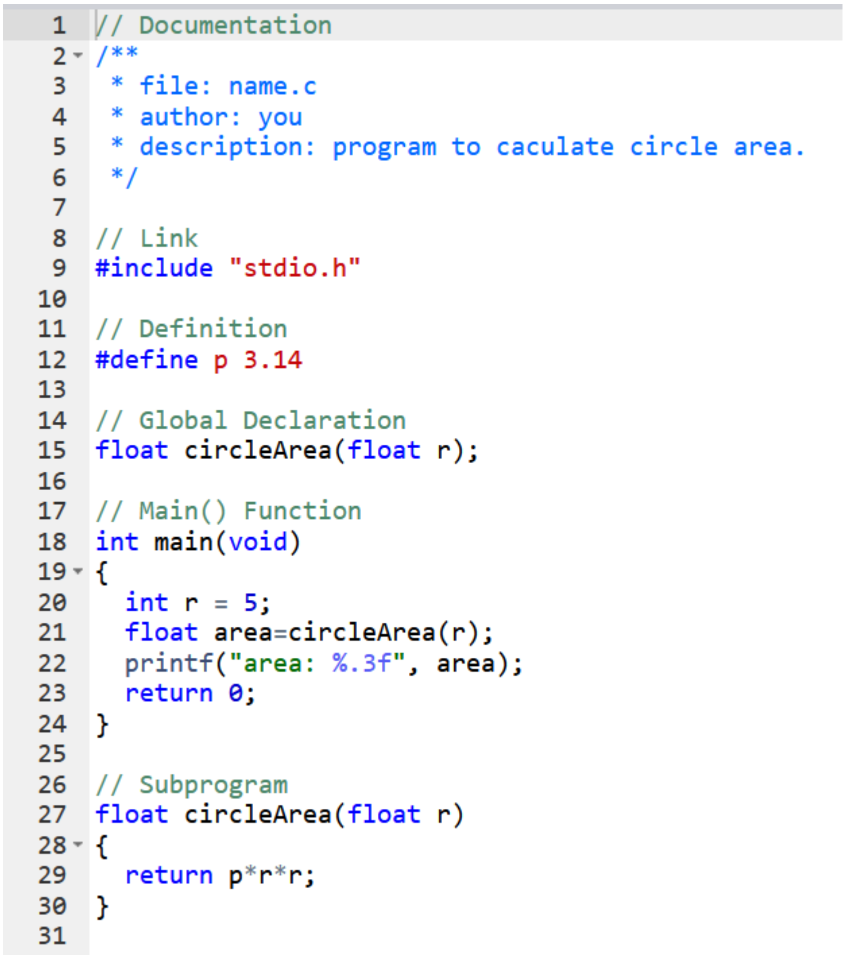
Variables are used to store and manipulate data within a program. They must be declared with a specific data type before they can be used. Control structures like loops and conditional statements allow for precise control of program flow, enabling tasks to be repeated or executed conditionally.

Header files provide essential libraries and declarations that extend the functionality of the C language. They are included at the beginning of a program using #include statements. Standard header files, such as for input and output operations, and for memory allocation and other utilities, are fundamental to writing C programs.

Furthermore, C requires a main function as the entry point for execution. This function is where the program begins its execution and typically calls other functions as needed. It is essential to adhere to the correct syntax and structure in order for the program to compile and run successfully.

Sections of the C Program

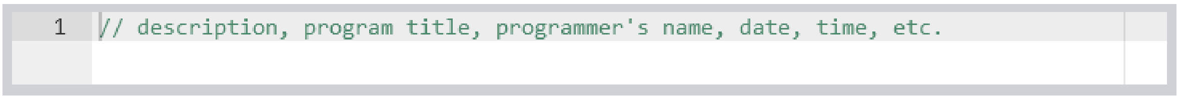
1. Documentation
2. Preprocessor Section
3. Definition
4. Global Declaration
5. Main() Function
6. Sub Programs



```c
// Documentation
/**
 * file: name.c
 * author: you
 * description: program to caculate circle area.
 */

// Link
#include "stdio.h"

// Definition
#define p 3.14

// Global Declaration
float circleArea(float r);

// Main() Function
int main(void)
{
  int r = 5;
  float area=circleArea(r);
  printf("area: %.3f", area);
  return 0;
}

// Subprogram
float circleArea(float r)
{
  return p*r*r;
}

```

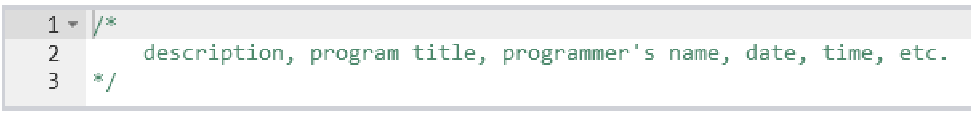
1. Documentation

This section encompasses the program's description, its title, as well as the date and time it was created. These details are typically provided at the beginning of the program in the form of comments. Documentation can take the form of:



```c
// description, program title, programmer's name, date, time, etc.
```

OR



```c
/*
    description, program title, programmer's name, date, time, etc.
*/
```

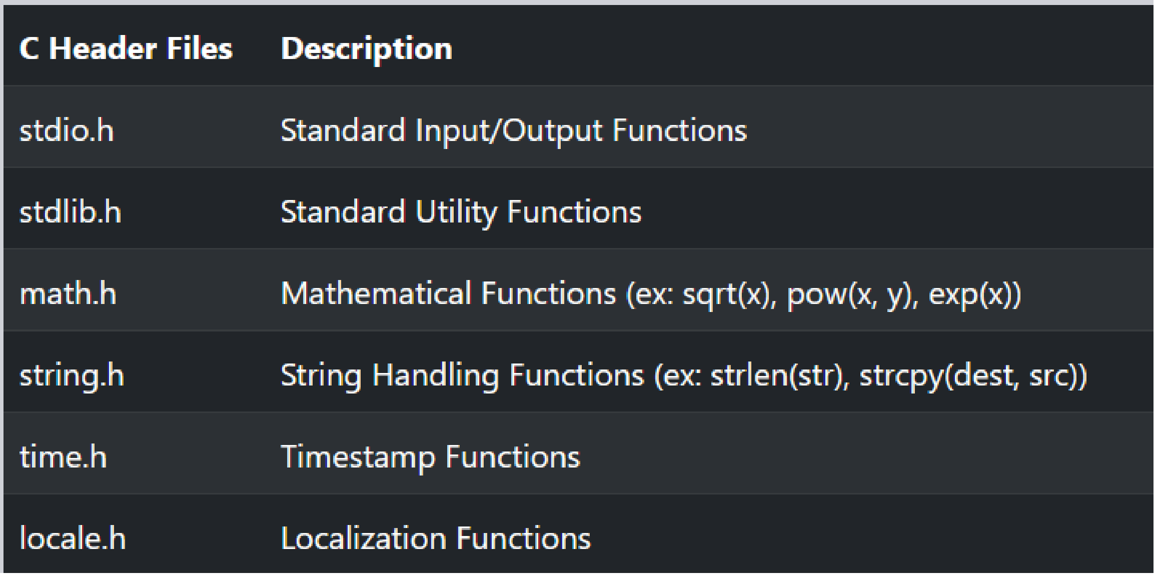
Any content written as comments serves as documentation for the program and does not affect the functionality of the provided code. Essentially, it provides an overview for the reader of the program.

2. Preprocessor Section

In the preprocessor section of the program, there are declarations for all the necessary header files. These files serve as bridges to incorporate enhanced code from external sources into

program. Before the compilation process, copies of these various files are integrated into program.
Example:

```c
#include "stdio.h"
#include "stdlib.h"
```



C Header Files	Description
stdio.h	Standard Input/Output Functions
stdlib.h	Standard Utility Functions
math.h	Mathematical Functions (ex: sqrt(x), pow(x, y), exp(x))
string.h	String Handling Functions (ex: strlen(str), strcpy(dest, src))
time.h	Timestamp Functions
locale.h	Localization Functions

3. Definition

The #define preprocessor is employed to establish a constant value that persists throughout the program. Whenever the compiler encounters this defined name, it is substituted with the actual piece of code it represents.

Example:

```c
#define p 3.14
```

4. Global Declaration

The global declaration section encompasses global variables, function declarations, and static variables. Variables and functions declared in this scope are accessible throughout the entire program.
Example:

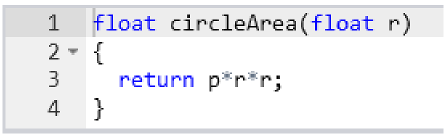
```c
int x = 10;
```

5. Main Function

C programs include a main function. The return type of the main() function can be either int or void. void main() informs the compiler that the program will not yield any value, while int main() indicates that the program will return an integer value.

6. Sub Programs

This section of the program is where user-defined functions are invoked. When called from either the main function or outside it, the program's control transitions to the respective function. These functions are defined based on the programmer's specific needs. Example:



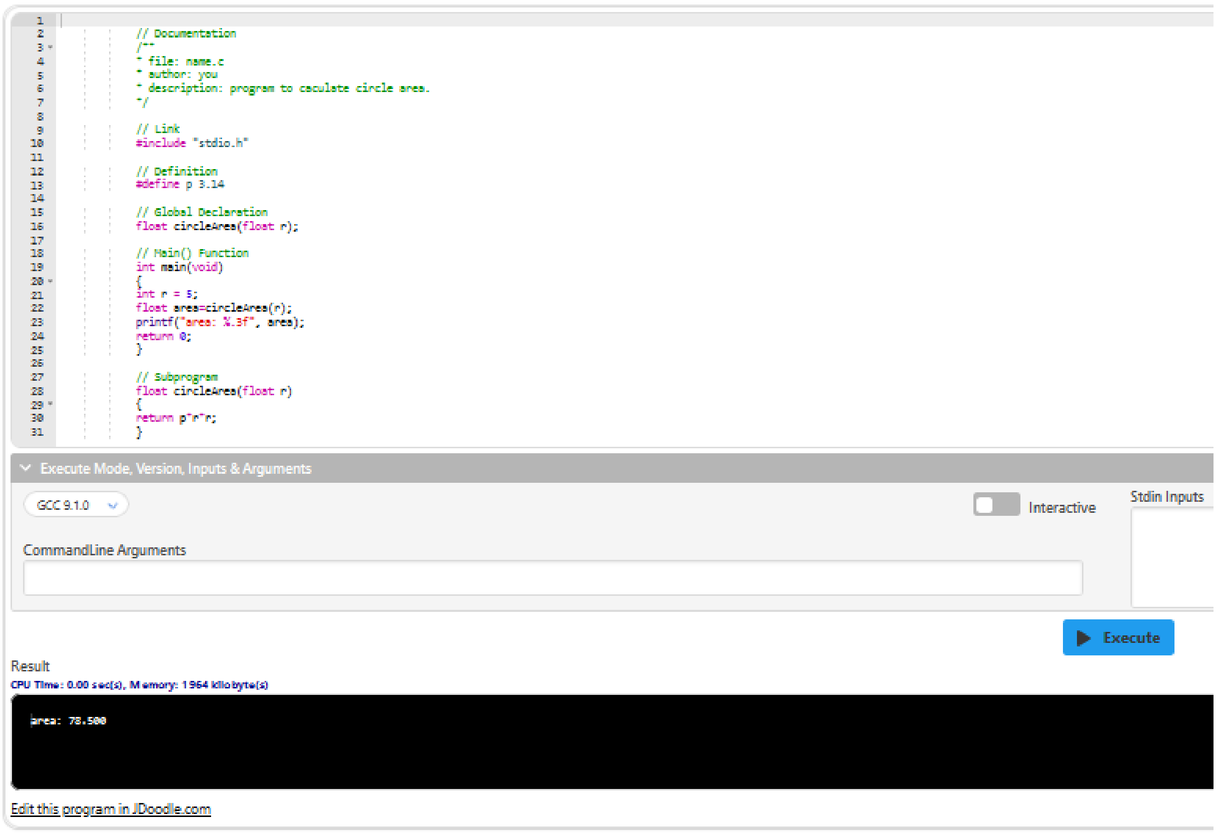
```
float circleArea(float r)
{
  return p*r*r;
}
```

C Terminology and Syntax

- **Statement:** A programming statement carries out a specific action in a program. It is concluded with a semicolon (;).
- **Preprocessor Directive:** A preprocessor directive begins with a hash sign (#). It is processed prior to the actual compilation of the program. Importantly, a preprocessor directive does not require a semicolon at the end.
- **Block:** A block is a collection of programming statements enclosed within braces { }.
- **Whitespaces:** Blank spaces, tabs, and newlines collectively constitute what we refer to as whitespaces.
- **Case Sensitivity:** C is case sensitive, meaning that "x1" and "X1" are distinct identifiers in the language.

Program execution

Execute the previous program within online compiler (JDoodle).



```c

// Documentation
/**
* file: name.c
* author: you
* description: program to caculate circle area.
*/

// Link
#include "stdio.h"

// Definition
#define p 3.14

// Global Declaration
float circleArea(float r);

// Main() Function
int main(void)
{
int r = 5;
float area=circleArea(r);
printf("area: %.3f", area);
return 0;
}

// Subprogram
float circleArea(float r)
{
return p*r*r;
}
```

Result

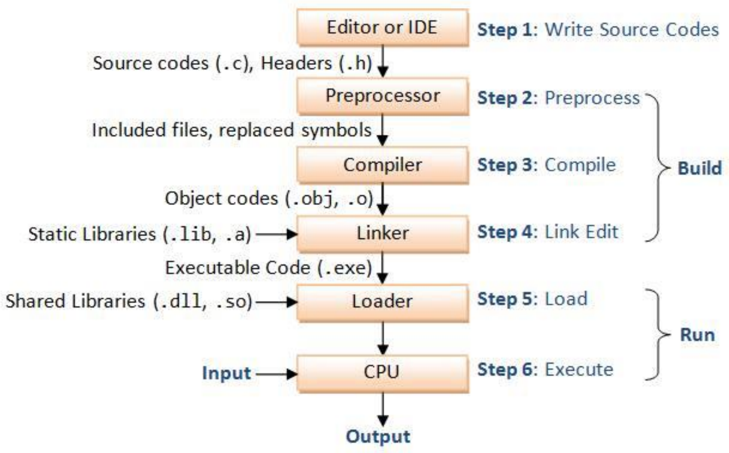
CPU Time: 0.00 sec(s), Memory: 1964 kilobyte(s)

```
area: 78.500
```

Edit this program in JDoodle.com

The Process of executing a C Program

- **Step 1:** Create the source codes (.c) along with any necessary header files (.h).
- **Step 2:** Apply preprocessing to the source codes based on the directives provided. Preprocessor directives, identified by a hash sign (#) like #include and #define, instruct the compiler to carry out specific actions (like incorporating another file or substituting symbols).
- **Step 3:** Compile the preprocessed source codes into object codes (.obj, .o).
- **Step 4:** Merge the compiled object codes with any other required object codes and library object codes (.lib, .a) to generate the executable code (.exe).
- **Step 5:** Load the resulting executable code into the computer's memory.
- **Step 6:** Execute the loaded executable code.



source: https://www3.ntu.edu.sg/home/ehchua/programming/cpp/c0_Introduction.html