CERTIFIED QUIZ

Introduction to C Language

C is a general-purpose, procedural programming language developed in 1972 by Dennis Ritchie at Bell Labs. It has been widely used for developing system software like operating systems, as well as various application programs. C provides a balance between low-level memory access and high-level language features, making it a foundational language in computer science.

Structure of a C Program

A typical C program consists of the following parts:
- Preprocessor Directives: Commands for the preprocessor (e.g., `#include`, `#define`).
- main() Function: Entry point for program execution.
- Variable Declarations: Declare variables used in the program.
- Statements and Expressions: Perform operations and computations.
- Return Statement: End the program with a return value.

Example:
c
#include
int main() {
printf("Hello, World!");
return 0;
}

Data Types in C

C provides various data types to represent different kinds of data:
- int: Represents integers (e.g., `int a = 5;`).
- float: Represents floating-point numbers (e.g., `float b = 3.14;`).
- double: Represents double-precision floating-point numbers.
- char: Represents single characters (e.g., `char c = 'A';`).
- void: Represents absence of a data type, often used in functions that do not return a value.

Modifiers such as `short`, `long`, `signed`, and `unsigned` can alter the size and range of data types.

Variables and Constants

Variables are used to store data that can be modified during program execution. They are declared with a specific data type.

Example:
c
int age = 25;
float pi = 3.1415;

Constants are fixed values that cannot be changed during execution. Constants are defined using the `const` keyword or `#define` preprocessor directive.

Example:
c

```
const int MAX_SIZE = 100;
#define PI 3.14
```

Operators in C

C supports a wide range of operators for performing operations on variables and data:
- Arithmetic Operators: `+`, `-`, ``, `/`, `%` (addition, subtraction, multiplication, division, modulus).
- Relational Operators: `==`, `!=`, `>`, `<`, `>=`, `<=` (comparison between values).
- Logical Operators: `&&`, `||`, `!` (logical AND, OR, NOT).
- Bitwise Operators: `&`, `|`, `^`, `~`, `<<`, `>>` (bit-level operations).
- Assignment Operators: `=`, `+=`, `-=`, `=`, `/=` (assign values or perform operations during assignment).
- Increment/Decrement Operators: `++`, `--` (increase or decrease the value of a variable by 1).
- Conditional (Ternary) Operator: `? :` (shorthand for if-else statements).

Control Structures

C offers several control structures for decision-making and looping:

- if-else: Conditional statements that execute code based on a condition.
Example:
c
```
if (a > b) {
printf("a is greater than b");
} else {
printf("b is greater than or equal to a");
}
```

- switch-case: Allows the selection of one among many options based on the value of a variable.
Example:
c
```
switch (grade) {
case 'A':
printf("Excellent!");
break;
case 'B':
printf("Good");
break;
default:
printf("Invalid grade");
}
```

- Loops: Repeatedly execute a block of code while a condition is true.
- `for` loop:
c
```
for (int i = 0; i < 10; i++) {
printf("%d\n", i);
}
```

- `while` loop:
c
```
while (x > 0) {
x--;
}
```

- `do-while` loop (guarantees at least one execution):
c
do {
x--;
} while (x > 0);

Functions in C

Functions in C help modularize the code by allowing you to define reusable blocks of code.

Syntax:
c
return_type function_name(parameters) {
// function body
return value;
}

Example:
c
int add(int a, int b) {
return a + b;
}

Functions can also have void as a return type if they don't return any value. C supports function prototypes, which provide the function signature before its definition.

Pointers

Pointers are variables that store the memory address of another variable. They provide powerful capabilities for memory manipulation and are essential for dynamic memory allocation, arrays, and function arguments.

Syntax:
c
int ptr;

Example:
c
int a = 10;
int ptr = &a;
printf("Address of a: %p", ptr);

- `ptr` dereferences the pointer, giving the value stored at the memory address.
- `&var` gives the address of the variable.

Arrays in C

An array is a collection of elements of the same type, stored in contiguous memory locations.

Syntax:
c
data_type array_name[size];

Example:
c
int numbers[5] = {1, 2, 3, 4, 5};

Arrays are zero-indexed, meaning the first element has an index of 0. C supports multidimensional arrays as well.

Strings in C

Strings are arrays of characters in C, and they are terminated with a null character (`\0`).

Example:
c
char str[] = "Hello, C!";

C provides the standard library `string.h` for common string functions such as `strlen()`, `strcpy()`, and `strcmp()`.

Structures and Unions

Structures allow grouping variables of different data types under a single name.

Syntax:
c
struct Person {
char name[50];
int age;
};

- Unions are similar to structures but store only one field at a time.

Syntax:
c
union Data {
int i;
float f;
};

Unions provide memory efficiency since all members share the same memory location.

Dynamic Memory Allocation

C provides functions from `stdlib.h` to allocate and manage memory dynamically during runtime:

- `malloc()`: Allocates a block of memory.
- `calloc()`: Allocates memory for an array of elements.
- `free()`: Frees previously allocated memory.
- `realloc()`: Reallocates memory to a different size.

Example:
c

```
int ptr = (int) malloc(5 sizeof(int));
if (ptr == NULL) {
printf("Memory not allocated.");
} else {
// Memory successfully allocated
free(ptr);
}
```

File Handling in C

C provides functions to work with files, allowing data to be read from or written to files.

- `fopen()`: Open a file.
- `fclose()`: Close a file.
- `fread()`: Read data from a file.
- `fwrite()`: Write data to a file.
- `fprintf()`: Write formatted output to a file.
- `fscanf()`: Read formatted input from a file.

Example:
c
```
FILE fp = fopen("file.txt", "w");
if (fp != NULL) {
fprintf(fp, "Hello, file!");
fclose(fp);
}
```