

Call by Value

1 Introduction

Call by value is a method of passing arguments to a function where a copy of the actual parameter is passed. Changes made to the parameter inside the function do not affect the original variable.

1.1 Code

```
1 #include <stdio.h>
2 int modify(int x)
3 {
4     x = x + 10;
5     printf("Inside function: %d\n", x);
6 }
7 int main()
8 {
9     int a = 5;
10    printf("Before function call: %d\n", a);
11    modify(a);
12    printf("After function call: %d\n", a);
13    return 0;
14 }
```

1.2 Output

```
Before function call: 5
Inside function: 15
After function call: 5
```

1.3 Conclusion

Call by value in C is a method of passing arguments where a copy of the actual variable is sent to the function. This ensures that any modifications made inside the function do not affect the original variable. It provides data security by preventing unintended changes but can be memory-intensive for large data structures. Call by value is useful when the function should work with temporary copies without altering the original values.

Call by Reference

2 Introduction

Call by reference is a method of passing arguments to a function where the function receives the memory address of the actual parameter instead of a copy. This allows the function to modify the original variable.

2.1 C Code

```
1 #include <stdio.h>
2 int modify(int *x)
3 {
4     *x = *x + 10;
5     printf("Inside function: %d\n", *x);
6 }
7 int main()
8 {
9     int a = 5;
10    printf("Before function call: %d\n", a);
11    modify(&a);
12    printf("After function call: %d\n", a);
13    return 0;
14 }
```

2.2 Output

```
Before function call: 5
Inside function: 15
After function call: 15
```

2.3 Conclusion

Call by Reference is a method in which a function receives the memory address (reference) of an argument rather than a copy of its value. This allows the function to directly modify the value of the argument.

Recursion

3 Introduction

Recursion in C is a process in which a function calls itself directly or indirectly in order to solve a problem. Each recursive call typically works on a smaller or simpler version of the problem, and the function will eventually reach a base case that terminates the recursion. Recursion is often used for problems that can be broken down into smaller subproblems of the same type, such as calculating factorials, Fibonacci numbers, or traversing trees.

3.1 C Code

```
1 #include <stdio.h>
2 int factorial(int n)
3 {
4     if (n == 0) return 1;
5     return n * factorial(n - 1);
6 }
7 int main()
8 {
9     int num = 5;
10    printf("Factorial of %d is %d\n", num, factorial(num));
11    return 0;
12 }
```

3.2 Output

```
Factorial of 5 is 120
```

3.3 Conclusion

Recursion in C is a programming technique in which a function calls itself to solve a problem. It is particularly useful for problems that can be broken down into smaller, similar sub-problems, such as calculating factorials, Fibonacci numbers, or tree traversal.

Pointer with Array

4 Introduction

Pointer variables and arrays in C are closely related concepts that work together efficiently in handling memory and data. Both pointers and arrays allow access to memory locations, but they do so in different ways.

4.1 C Code

```
1 #include <stdio.h>
2 int main()
3 {
4     int arr[] = {1, 2, 3, 4, 5};
5     int *ptr = arr;
6     for(int i = 0; i < 5; i++)
7     {
8         printf("%d ", *(ptr + i));
9     }
10    return 0;
11 }
```

4.2 Output



1 2 3 4 5

4.3 Conclusion

Using pointers with arrays improves efficiency and allows direct memory manipulation, which can be useful for various operations.

Structure

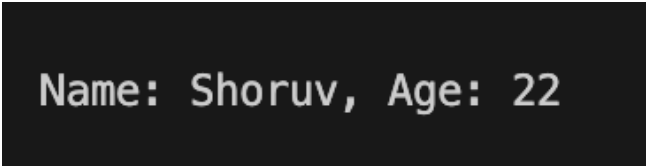
5 Introduction

Structure in C is a user-defined data type that allows grouping different data types under a single name. It is useful when we need to store multiple related variables of different types together.

5.1 C Code

```
1 #include <stdio.h>
2 struct Student
3 {
4     char name[50];
5     int age;
6 };
7 int main()
8 {
9     struct Student s1 = {"Shoruv", 22};
10    printf("Name: %s, Age: %d\n", s1.name, s1.age);
11    return 0;
12 }
```

5.2 Output

The output of the C program is displayed on a black background with white text, showing the name and age of the student.

5.3 Conclusion

Structures provide a way to organize and manage related data efficiently, making them useful in complex applications.

File Input/Output

6 Introduction

File Input/Output in C allows reading from and writing to files, enabling data storage beyond program execution. The C standard library provides functions for handling files through the `stdio.h` header.

6.1 C Code

```
1 #include <stdio.h>
2 int main()
3 {
4     FILE *fptr = fopen("test.txt", "w");
5     fprintf(fptr, "Hello, File I/O!\n");
6     fclose(fptr);
7     return 0;
8 }
```

6.2 Output (Content of test.txt)



Hello, File I/O!

6.3 Conclusion

File I/O is essential for handling data storage and retrieval, allowing programs to work with external files efficiently. It enables persistent data management, making it possible to read, write, and manipulate large data sets beyond the runtime of a program. Using different file modes and operations, C provides a flexible approach to working with both text and binary files. Proper file handling ensures data integrity, reduces memory usage, and enhances overall program functionality.