c_programming_session_2

اللهمَّ علِّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمًا.

Tags: c programming
Status: #Adult

C Programming Session 2 - Loops and Functions

Overview

Session 2 covers loop structures, user-defined functions, recursion, and multi-file project organization.

Loop Structures

For Loop

General syntax:

```
for (initialization; condition; increment/decrement) {
    // code block
}
```

Example: Print Numbers 1-10

```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

Example: Count by 3s (0 to 100 and back)

```c
int main() {
    int i;

    // Count up by 3s
    for (i = 0; i < 100; i += 3) {
        printf("%d\n", i);
    }
    // Count down by 3s
    for (i = 100; i >= 0; i -= 3) {
        printf("%d\n", i);
    }
    return 0;
}
```

Example: Multiplication Table

```c
int main() {
    int number;
    printf("Enter your number: ");
    scanf("%d", &number);

    for (int i = 1; i <= 12; i++) {
        printf("%d * %d = %d\n", number, i, number * i);
    }
    return 0;
}
```

While vs Do-While Loops

Feature	While Loop	Do-While Loop
Condition Check	Before execution	After execution
Minimum Executions	0	1
Use Case	When condition might be false initially	When at least one execution is required

Comparison Example

```c
int main() {
    int i = 1;

    // Do-while: executes first, then checks condition
    do {
        printf("a");
        i++;
    } while (i < 5);

    // While: checks condition first, then executes
    while (i < 8) {
        printf("b");
        i++;
    }

    // Output: aaaabbb
    return 0;
}
```

Infinite Loops

```c
// Method 1: While loop
while (1) {
    // infinite loop
}

// Method 2: Do-while loop
do {
    // code
} while (1);

// Method 3: For loop
for (;;) {
    printf("A");
}
```

Break vs Continue

Keyword	Action	Effect
`break`	Exit loop completely	Terminates loop execution
`continue`	Skip current iteration	Jumps to next iteration

```c
for (int i = 0; i <= 8; i++) {
    if (i > 5) {
        break;     // Exit when i > 5
    }
    printf("%d", i);
}
// Output: 012345

for (int i = 0; i <= 8; i++) {
    if (i > 5) {
        continue;  // Skip when i > 5
    }
    printf("%d", i);
}
// Output: 012345
```

Practical Examples

Factorial Calculation

```c
int main() {
    int x, fact = 1;
    printf("Enter number: ");
    scanf("%d", &x);

    int i = 1;
    while (i <= x) {
        fact *= i;
        i++;
    }

    printf("%d factorial = %d\n", x, fact);
    return 0;
}
```

Sum of 10 Numbers

```c
int main() {
    int x, sum = 0;

    for (int i = 1; i <= 10; i++) {
        printf("Enter number %d: ", i);
```

```
        scanf("%d", &x);
        sum += x;
    }

    printf("Sum is: %d\n", sum);
    return 0;
}
```

Variable Swapping Techniques

Method 1: Using Temporary Variable

```
int main() {
    int x, y, temp;
    printf("Enter x: ");
    scanf("%d", &x);
    printf("Enter y: ");
    scanf("%d", &y);

    printf("Before: x = %d, y = %d\n", x, y);

    temp = x;
    x = y;
    y = temp;

    printf("After: x = %d, y = %d\n", x, y);
    return 0;
}
```

Method 2: Using Arithmetic

```
x = x + y;
y = x - y;
x = x - y;
```

Method 3: Using XOR

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Software Development Lifecycle (SDLC)

examples : Waterfall ,V-cycle , Agile

V-Cycle Model (Common in Embedded Systems)

```
Requirements Analysis       ↔              Acceptance Testing
    System Design           ↔          Integration Testing
        Architecture Design ↔      Module Testing
            Module Design   ↔ Unit Testing
                            ↓
                          Code
```

Key Phases:

1. Requirements Gathering: Customer needs analysis
2. System Design: HSI (HW/SW Interface), SRS (SW Requirements Specification)
3. Component Design: CDD (Component Design Document)
4. Development: Code implementation
5. Testing: Unit → Integration → System → Acceptance . {VATS tester}
6. Deployment: Customer delivery

User-Defined Functions

Function Structure

```c
return_type function_name(parameters) {
    // function body
    return value; // if not void
}
```

Example: Maximum Function

```c
#include <stdio.h>

int Get_Max(int x, int y);

int main() {
    int x, y, result;
```

```
    printf("Enter x: ");
    scanf("%d", &x);
    printf("Enter y: ");
    scanf("%d", &y);

    result = Get_Max(x, y);

    if (result == 0) {
        printf("Both are equal\n");
    } else {
        printf("Max = %d\n", result);
    }

    return 0;
}

int Get_Max(int x, int y) {
    if (x > y) {
        return x;
    } else if (x < y) {
        return y;
    } else {
        return 0; // Equal values
    }
}
```

Global vs Local Variables

Scope	Global Variables	Local Variables
Declaration	Outside all functions	Inside functions
Accessibility	All functions	Current function only
Lifetime	Program duration	Function execution
Memory	Static memory area	Stack
Default Value	0 (initialized)	Garbage (uninitialized)

Multi-File Projects

File Organization

File Type	Extension	Contents
Header File	`.h`	Function prototypes, global variable declarations
Source File	`.c`	Function implementations, main code

Example Project Structure

```
project/
├── main.c          // Main program
├── functions.h     // Function prototypes
├── functions.c     // Function implementations
└── globals.h       // Global variables and constants
```

Include Syntax

```c
#include "functions.h"     // User-defined headers (local files)
#include <stdio.h>         // System headers (standard library)
```

Recursion

Definition

A function that calls itself directly or indirectly.

> ⚠ **Note for Embedded Systems**
>
> Recursion is generally not recommended in embedded C due to:
>
> - Stack memory limitations
> - Unpredictable memory usage
> - Potential stack overflow

Factorial Example (Educational Purpose)

```c
#include <stdio.h>

int factorial(int n);
```

```c
int main() {
    int n, result;
    printf("Enter number: ");
    scanf("%d", &n);

    result = factorial(n);
    printf("Factorial of %d = %d\n", n, result);

    return 0;
}

int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Practice Tasks

Task 1: Cube Calculator

Create a function to calculate the cube of a number.

```c
#include <stdio.h>

int Get_cube(int x);

int main() {
    int x, y;
    printf("enter x value : ");
    scanf("%d", &x);
    y = Get_cube(x);
        printf("cube = %d\n", y);
}

int Get_cube(int x) {
   return x*x*x;
}
```

Task 2: Star Triangle Pattern

```c
#include <stdio.h>
int star_triangle(int n);
int inv_star_triangle(int n);
int x, y,n;

int main() {
   printf("Enter the number of rows: ");
   scanf("%d", &n);

   x = star_triangle(n);
   y = inv_star_triangle(n);
}

int star_triangle(int n) {
    // First loop for printing all rows
    for (int i = 0; i < n; i++) {
        // First inner loop for printing leading white spaces
        for (int j = 0; j < (n-i) ; j++) {
             printf(" ");
        }
        // Second inner loop for printing stars *
        for (int k = 0; k < (i) + 1; k++) {
           printf("* ");
        }
        printf("\n");
    }
}

int inv_star_triangle(int n) {
    for (int i = 0; i < n; i++) {
    // Print leading spaces
        for (int j = 0; j < i + 1; j++) {
            printf(" ");
        }
    // Print stars
        for (int k = 0; k < n - i; k++) {
           printf("* ");
        }
        printf("\n");
    }
}
```

```
Enter the number of rows: 3
   *
  * *
 * * *
 * * *
  * *
   *
```

Task 3: Pascal's Triangle

```c
#include <stdio.h>

int pascal_triangle(int n);
int inv_pascal_triangle(int n);
int x, y,n,c;
int main() {
   printf("Enter the number of rows: ");
   scanf("%d", &n);

   x = pascal_triangle(n);
   y = inv_pascal_triangle(n);
}

int pascal_triangle(int n) {
    // First loop for printing all rows
    for (int i = 0; i < n; i++) {
        // First inner loop for printing leading white spaces
        for (int j = 0; j < (n-i) ; j++) {
             printf(" ");
        }
        // Second inner loop for printing numbers
        for (int k = 0; k < (i) + 1; k++) {
            if (k == 0 || i == 0){
                c = 1;
            }
            else{
                c = c * (i - k + 1) / k;
            }
            printf("%d ",c);
        }
        printf("\n");
    }
}
```

```
int inv_pascal_triangle(int n) {
    for (int i = 0; i < n; i++) {
    // Print leading spaces
        for (int j = 0; j < i + 1; j++) {
            printf(" ");
        }
    // Print numbers
        for (int k = 0; k < n - i; k++) {
            if (k == 0){
                c = 1;
            }
            else{
                c = c * (n- i - k ) / k;
            }
            printf("%d ",c);
        }
        printf("\n");
    }
}
```

```
Enter the number of rows: 4
    1
   1 1
  1 2 1
 1 3 3 1
 1 3 3 1
  1 2 1
   1 1
    1
```

Task 4: Fibonacci Series

Generate the first N numbers of the Fibonacci sequence.

```
#include <stdio.h>

int fibonacci(int n);
int x,n,c;
int main() {
   printf("Enter the number of terms: ");
   scanf("%d", &n);
   x = fibonacci(n);
```

```
}

int fibonacci(int n){
    int val1 = 0;
    int val2 = 1;
    for (int i = 0; i <= n; i++){
        if(i == 0 || i == 1){
            c=i;
        }
        else{
            c = val1 + val2;
            val1 = val2;
            val2 =c;
        }
        printf("%d",c);
        printf(" ,");
    }
}
```

```
Enter the number of terms: 10
0 ,1 ,1 ,2 ,3 ,5 ,8 ,13 ,21 ,34 ,55 ,
```

Task 5: Number Reversal

Convert 12345 → 54321

```
#include <STDIO.H>

int reverse_num(int n);
int x,n;
int main() {
   printf("Enter the number to reverse: ");
   scanf("%d", &n);
   x = reverse_num(n);
   printf("reversed number is : %d", x);
}

int reverse_num(int n){
    int remainder = 0;
    int digits;
    while (n != 0){
        digits = n% 10;
        remainder =remainder *10 + digits;
```

```
        n = n /10;
    }
    return remainder;
}
```

```
Enter the number to reverse: 19760428
reversed number is : 82406791
```

Task 6: Decimal to Binary Conversion

Convert decimal numbers to binary without using arrays.

```
#include <stdio.h>

int dec_bin(int n);
int main() {
    int n;
    do{
        printf("Enter the decimal number: ");
        scanf("%d", &n);
        int result = dec_bin(n);
        printf("Binary representation: %d\n", result);
    } while (1);
}

int dec_bin(int z) {
    int binary_num = 0;
    int base = 1;
    while (z > 0) {
        binary_num += (z % 2) * base;
        base *= 10;
        z /= 2;
    }
    return binary_num;
}
```

```
Enter the decimal number: 50
Binary representation: 110010
Enter the decimal number: 82
Binary representation: 1010010
Enter the decimal number: 2
Binary representation: 10
```

```
Enter the decimal number: 8
Binary representation: 1000
Enter the decimal number: 16
Binary representation: 10000
```

Task 7: Number Pairs

For number 12, find all pairs that sum to 12:

```c
#include <STDIO.H>

int addative(int n);
int x,n,c;
int main (){
    printf("Enter the number : ");
    scanf("%d", &n);
    x = addative(n);
}

int addative(int n){
    for (int i = 1; i <= (n+1)/2; i++){
        c = n - i;
        printf("%d =  %d + %d\n",n,c,i);
    }
 }
```

```
Enter the number : 20
20 =  19 + 1
20 =  18 + 2
20 =  17 + 3
20 =  16 + 4
20 =  15 + 5
20 =  14 + 6
20 =  13 + 7
20 =  12 + 8
20 =  11 + 9
20 =  10 + 10
```

Next Session Topics

- Arrays and data structures

- Sorting algorithms
- Pointers and memory operations
- Advanced pointer operations