c_programming_session_3

اللهمَّ علّمنا ما ينفعنا، وانفعنا بما علمتنا، وزدنا علمًا. وافتح علينا فتحًا عظيمً.

Tags: c programming
Status: #Adult

C Programming Session 3 - Arrays, Pointers, and Algorithms

Memory Organization

Memory Types Overview

Memory Type	Volatile	Non-Volatile
Characteristics	Data lost when power removed	Data persists without power
Examples	RAM	ROM, Flash, EEPROM

Non-Volatile Memory Types

	Masked ROM (MROM)	OTP ROM / PROM	EPROM	EEPRC
Programming	Manufacturer	User (one-time)	User(UV light+programmer)	User(e in-circ
Reprogram	No	No	Yes (UV light, entire chip)	Yes (el / smal
Erasure Method	N/A (permanent)	N/A (permanent)	UV Light	Electri
In-Circuit Erase	No	No	No (must remove chip)	Yes
Cost (per bit)	Lowest(high vol)	Low(moderate vol)	Moderate	High

	Masked ROM (MROM)	OTP ROM / PROM	EPROM	EEPRC
Read Speed	Fast	Fast	Fast	Mode
Write Speed	(factory prog)	Fast (one-time)	Slow	Slow
Typical Cycles	N/A	N/A	Few thousands	100k t
Common Uses	Firmware mass-prod	Firmware_smallruns, dev.	Older firmware, prototype	Config data,s data l

Memory Sections

ROM Sections

```
.BSS      - Uninitialized global variables
.vector   - Vector table addresses
.code     - Program instructions
.data     - Global variable initialization values
.rodata   - Read-only data (constants)
```

RAM Sections

```
.data     - Initialized global variables
.BSS      - Uninitialized global variables (runtime)
.stack    - Function calls, local variables
.heap     - Dynamic memory allocation
```

Context Switching in Function Calls

```
int main() {
    calc();          // 1. Jump from main
    // 6. Continue execution
    /*Code*/
}

void calc() {        // 2. Jump to calc
    sum();           // 3. Jump from calc
    // 5. Return to calc
}

void sum() {         // 4. Jump to sum
    // Function body
}                    // 5. Return from sum
```

Inline Functions

```
inline int add(int a, int b) {
    return a + b;
}
```

- Pros: No function call overhead
- Cons: Increased code size (function code copied at each call site)

Arrays

Array Declaration and Initialization

```
// Basic syntax
datatype name[size] = {initialization};

// Examples
int array1[5] = {0};              // {0, 0, 0, 0, 0}
int array1[5] = {1, 5, 7, 8, 9};   // {1, 5, 7, 8, 9}

// Accessing elements
int y = array1[3];  // y = 8 (index starts from 0)
```

Important Array Rules

- Index Range: 0 to n-1 (where n is array size)
- Index Type: Must be constant (not variable)
- Data Types: Can be primitive or non-primitive types

Searching Algorithms

Linear Search

Feature	Linear Search
Method	Check each element sequentially
Time Complexity	O(n)
Requirement	No sorting required
Best for	Small arrays, unsorted data

```c
#include <stdio.h>

int linear_search(int target, int arr[], int size);

int main() {
    int array[10] = {1, 2, 7, 8, 19, 30, 41, 67, 69, 100};
    int target;

    printf("Enter number to search: ");
    scanf("%d", &target);

    int result = linear_search(target, array, 10);

    if (result != -1) {
        printf("Number found at index %d\n", result);
    } else {
        printf("Number not found\n");
    }

    return 0;
}

int linear_search(int target, int arr[], int size) {
    for (int i = 0; i < size; i++) {
        if (target == arr[i]) {
            return i;  // Return index if found
        }
    }
    return -1;  // Return -1 if not found
}
```

Binary Search

Feature	Binary Search
Method	Divide array in half, compare with middle
Time Complexity	O(log n)
Requirement	Array must be sorted
Best for	Large sorted arrays

```
#include <stdio.h>

int binary_search(int target, int arr[], int size);

int main() {
    int array[10] = {1, 2, 7, 8, 19, 32, 45, 67, 69, 100};  // Sorted
array
    int target;

    printf("Enter number to search: ");
    scanf("%d", &target);

    int result = binary_search(target, array, 10);

    if (result != -1) {
        printf("Number found at index %d\n", result);
    } else {
        printf("Number not found\n");
    }

    return 0;
}

int binary_search(int target, int arr[], int size) {
    int start = 0;
    int end = size - 1;

    while (start <= end) {
        int mid = (start + end) / 2;

        if (target == arr[mid]) {
            return mid;
        } else if (target > arr[mid]) {
            start = mid + 1;
        } else {
            end = mid - 1;
        }
    }

    return -1;  // Not found
}
```

Pointers

Pointer Declaration and Usage

```
// Pointer declaration (all equivalent)
int *ptr;
int* ptr;
int * ptr;

// Basic pointer operations
int x = 10;
int *ptr;
ptr = &x;    // ptr points to address of x
*ptr = 5;    // Write: change value at ptr's address (x becomes 5)
int y = *ptr; // Read: get value at ptr's address
```

Pointer Arithmetic

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;  // Points to first element

ptr[2] = 10;     // Equivalent to *(ptr + 2) = 10
                 // Sets arr[2] = 10

// Pointer arithmetic
ptr++;           // Move to next element
ptr--;           // Move to previous element
ptr = ptr + 2;   // Move forward by 2 elements
ptr = ptr - 2;   // Move backward by 2 elements
```

Array Sum Using Pointers

```c
#include <stdio.h>

int sum_array(int arr[], int size);

int main() {
    int array[10] = {1, 2, 7, 8, 19, 32, 45, 67, 69, 1000};
    int result = sum_array(array, 10);
    printf("Sum: %d\n", result);
    return 0;
}

int sum_array(int arr[], int size) {
    int *ptr = arr;
    int sum = 0;

    for (int i = 0; i < size; i++) {
        sum += ptr[i];  // or sum += *(ptr + i);
    }

    return sum;
}
```

Pass by Value vs Pass by Reference

Pass by Value

```c
#include <stdio.h>

void increment_by_value(int num) {
    num = num + 10;  // Modifies local copy only
    printf("Inside function: %d\n", num);
}

int main() {
    int x = 5;
    printf("Before: %d\n", x);
    increment_by_value(x);  // Pass copy of x
    printf("After: %d\n", x);   // x unchanged

    // Output:
    // Before: 5
    // Inside function: 15
    // After: 5

    return 0;
}
```

Pass by Reference (Address)

```
#include <stdio.h>

void increment_by_address(int *ptr) {
    *ptr = *ptr + 10;  // Modifies original variable
    printf("Inside function: %d\n", *ptr);
}

int main() {
    int x = 5;
    printf("Before: %d\n", x);
    increment_by_address(&x);  // Pass address of x
    printf("After: %d\n", x);   // x is modified

    // Output:
    // Before: 5
    // Inside function: 15
    // After: 15

    return 0;
}
```

Pointer Problems to Avoid

Dangling Pointer

```
int *func(void) {
    int x = 10;     // Local variable
    return &x;      // WRONG: x is destroyed when function ends
}                   // Returned address points to deallocated memory

void main(void) {
    int *ptr = func();
    printf("%d", *ptr);  // Undefined behavior - accessing invalid memory
}
```

Wild Pointer

```
void main(void) {
    int *ptr;           // Uninitialized pointer
    printf("%d", *ptr); // WRONG: ptr points to random memory location
}
```

Memory Address Operations

Pointer Arithmetic Example

```
// Given array starting at address 1000, each int = 4 bytes
int arr[10];

&arr[0] → 1000  // Address of first element
&arr[1] → 1004  // Address of second element
&arr[2] → 1008  // Address of third element
&arr[5] → 1020  // Address of sixth element
```

Quiz: Pointer Operations

Given:

- `z = 5` at address 999
- `x = 10` at address 1000
- `y = 8` at address 1001
- `ptr = 1000` at address 2000

```
printf("%d", x);        // Output: 10
printf("%p", &x);       // Output: 1000
printf("%p", ptr);      // Output: 1000
printf("%d", *ptr);     // Output: 10
printf("%p", &ptr);     // Output: 2000

// Pointer arithmetic
ptr++;              // ptr now points to y (address 1001)
ptr--;              // ptr now points to z (address 999)
ptr = ptr + 2;      // ptr points 2 positions after current
ptr = ptr - 2;      // ptr points 2 positions before current
```

Practice Tasks

Task 1: Palindrome Number Check

Check if a number reads the same forwards and backwards (e.g., 12321, 9009).

```
#include <stdio.h>
int palindrome(int y);
int reverse_num(int n);
int main() {
    int number;
    do // علشان اجرب اكتر من رقم ورا بعض عادي
    {
    printf("enter the number to check palindrome : ");
    scanf("%d",&number);
        if (palindrome(number)) {
            printf("The number %d is: Palindrome\n", number);
        } else {
            printf("The number %d is: Not Palindrome\n", number);
        }
    } while (1);
}

int palindrome(int y){
    int x = reverse_num(y);
    return x == y;
}

int reverse_num(int n){
    int reverse = 0;
    int reminder;
    while (n != 0){
        reminder = n% 10;
        reverse =reverse *10 + reminder;
        n = n /10;
    }
    return reverse;
}
```

```
enter the number to check palindrome : 12321
The number 12321 is: Palindrome
enter the number to check palindrome : 888
The number 888 is: Palindrome
```

```
enter the number to check palindrome : 1234
The number 1234 is: Not Palindrome
```

Task 2: Scalar Multiplication

Multiply two arrays element-wise:

```
(a1, a2, a3) · (b1, b2, b3) = a1*b1 + a2*b2 + a3*b3
```

```
#include <stdio.h>
int scalar_arr(int y);
int x, y,sum=0;
int array1[10] = {1, 4, 5, 6, 8, 9, 11, 13, 17, 19};
int array2[10] = {2, 3, 4, 5, 6, 10, 15, 17, 19, 20};

int main() {
    x = scalar_arr(y);
    printf("the scalar product is %d",x);
}
//1*2 + 3*4 + 4*5 + 5*6 + 6*8 + 10*9 + 15*11 + 17*13 + 19*17 + 20*19 = 1291
int scalar_arr(int y){
    int*ptr1 =array1;
    int*ptr2 =array2;
    for (int i = 0; i < 10 ; i++){
        sum = sum + (ptr1[i] * ptr2[i]);
    }
    return sum;
}
```

```
the scalar product is 1291
```

Task 3: Bubble Sort Implementation

Implement bubble sort algorithm using functions and loops.

```
#include <stdio.h>
void bubble_sort(int array[] , int size);
void print_array(int array[], int size);
int array1[10] = {1, 40, 55, 67, 8, 9, 110, 13, 70, 19};
int main() {
    bubble_sort(array1,10);
    printf("the sorted array is : ");
    print_array(array1, 10);
}

void bubble_sort(int array[] , int size){
    for (int i = 0; i < size -1 ; i++){
        for (int k = 0; k < size - i -1; k++){
            if (array[k]> array[k+1]){
                int t = array[k];
                array[k] = array[k+1];
                array[k+1] = t;
            }
            else{
            }
        }
    }
}

void print_array(int array[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}
```

```
the sorted array is : 1 8 9 13 19 40 55 67 70 110
```

Task 4: Binary to Decimal Conversion

Convert binary numbers to decimal representation.

```
#include <stdio.h>
#include<math.h>
int bin_to_dec(int n);

int main() {
    int number;
    int y;
    do {// علشان اجرب اكتر من رقم ورا بعض عادي
        printf("enter the binary to convert : ");
        scanf("%d",&number);
        y = bin_to_dec(number);
        printf("binary %d converted to decimal is : %d\n",number,y);
    } while (1);
}

int bin_to_dec(int n){ //n = 101
    int decimal_num = 0;
    int reminder;
    int i = 0;
    while (n != 0){
        reminder = n% 10;
        n = n /10;
        decimal_num += reminder * pow(2, i);
        i++;
    }
    return decimal_num;
}
//1. rem = 1 >>decimal_num =1
//2. rem = 0 >>decimal_num =1+0
//3. rem = 1 >>decimal_num =1+0+4
```

```
enter the binary to convert : 101
binary 101 converted to decimal is : 5
enter the binary to convert : 1111111
binary 1111111 converted to decimal is : 127
enter the binary to convert : 10100011010
binary 2147483647 converted to decimal is : 2559
```

Task 5: Arithmetic Sequence Sum

Given parameters (start=2, count=7, step=3):
Calculate: 2 + 5 + 8 + 11 + 14 + 17 + 20 = 77

```
#include <stdio.h>
int arthimatic_sum(int x,int y,int z);

int main() {
    int num1,num2,num3;
    int y;
    do { // علشان اجرب اكتر من رقم ورا بعض عادي
        printf("Enter start : ");
        scanf("%d",&num1);
        printf("Enter number of terms : ");
        scanf("%d",&num2);
        printf("Enter step : ");
        scanf("%d",&num3);
        y = arthimatic_sum(num1,num2,num3);
        printf("arthimatic sum (start %d, # terms %d, step %d) is : %d\n",num1,num2,num3,y);
    } while (1);
}

int arthimatic_sum(int x,int y,int z){
    int sum=0;
    int i =0;
    while (i < y){
        sum += x + z * i;
        i++;
    }
    return sum;
}
```

```
Enter start : 2
Enter number of terms : 7
Enter step : 3
arthimatic sum (start 2, # terms 7, step 3) is : 77
```

Algorithm Complexity Comparison

Algorithm	Time Complexity	Space Complexity	Best Use Case
Linear Search	O(n)	O(1)	Unsorted small arrays
Binary Search	O(log n)	O(1)	Sorted large arrays
Bubble Sort	$O(n^2)$	O(1)	Educational, small arrays

Next Session Preview

- Data type modifiers (signed, unsigned, short, long)
- Structures and unions
- Bit manipulation and bit fields

- Advanced memory concepts