Faculty of Engineering - Cairo University

Credit Hours System

CUERT Report

on

C and Embedded C Questions

Submitted by:

Aya Ayman Mohamed Abdelmegid

Student Code:

1210205

Department:

CCE-C

Phone number:

+20 1028288368

1. Concept of Pointers, Their Types, and Usage in C

Overview of Pointers

Pointers are variables that store memory addresses, enabling direct manipulation of memory.

Types of Pointers

- **Null Pointer:** A pointer that does not point to any valid memory location.
- **Void Pointer:** A pointer that can hold addresses of any data type but cannot be dereferenced directly.
- **Function Pointer:** Stores the address of a function, enabling dynamic function calls.
- **Wild Pointer:** An uninitialized pointer that may point to any arbitrary memory location, often leading to undefined behavior and potential risks.

Usage of Pointers

- **Dynamic Memory Allocation:** Functions like malloc, calloc, and realloc utilize pointers to allocate and manage memory dynamically during runtime.
- **Array Manipulation:** Pointers facilitate manipulation of arrays.
- **Function Parameters:** Passing large data structures or arrays by reference using pointers avoids unnecessary data copying and improves memory stack performance.

---

2. Difference Between Pointers and Arrays in C

- **Arrays:** Represent block of memory for storing multiple elements of the same type. The name of an array serves as a constant pointer to the first element, but the array's base address cannot be reassigned.

- **Pointers:** Variables that hold memory addresses and can be reassigned to point to different locations, providing greater flexibility for dynamic memory management.

---

3. Concept of Pointer Arithmetic

Pointer arithmetic involves operations that adjust the memory addresses stored in pointers, allowing navigation through arrays and memory regions:

- **Incrementing/Decrementing Pointers:** Moves the pointer to the next element based on the size of the data type it points to (e.g., ptr + 1 for an int* pointer advances by sizeof(int) bytes).
- **Pointer Differences:** Subtracting two pointers within the same array yields the number of elements between them, providing insight into their relative positions.

---

4. Memory Layout of C Programs

- **Text Segment:** Contains the compiled code of the program. This segment is read-only to prevent accidental modification of executable code.
- **Data Segment:** Stores initialized global and static variables. This segment has a fixed size determined at compile time.
- **BSS Segment:** Contains uninitialized global and static variables, which are initialized to zero by default.
- **Heap:** Used for dynamic memory allocation, allowing variables to be created and destroyed during runtime.
- **Stack:** Manages function call information, local variables, and control flow, including return addresses and function parameters.

---

5. Difference Between Signed and Unsigned Data Types in C

- **Signed Data Types:** Capable of representing both positive and negative values. Their range is typically from -2^(n-1) to 2^(n-1) - 1, where n is the number of bits.
- **Unsigned Data Types:** Only represent non-negative values. Their range extends from 0 to (2^n) - 1, maximizing the positive value representation for a given number of bits.

---

6. Importance of Data Type Sizes in Embedded Systems

- **Memory Constraints:** Choosing appropriately sized data types prevents overflow, reduces memory usage, and ensures system stability.
- **Predictability:** Accurate data type sizing helps maintain predictable system behavior and efficient memory utilization, crucial for embedded applications with stringent performance requirements.

---

7. Difference Between a struct and a union in C

- **Struct:** A composite data type where each member has its own separate memory location, allowing for the grouping of variables of different types. The total size of a struct is the sum of its members' sizes in addition to padding.
- **Union:** A data type where all members share the same memory location, so only one member can be used at a time. This allows efficient memory usage when only one of several variables is needed at any given time. The size of the union is the size of the greatest element.

---

8. Concept of Bitwise Operators and Their Use

Bitwise operators perform operations directly on the binary representations of data, the available bitwise operations are AND &, OR |, XOR ^, NOT ~, Left Shift <<, Right Shift >>

---

9. Concept of Bitfields in C

Bitfields allow the allocation of a specific number of bits within a struct, enabling compact data storage:

- **Bitfields:** Useful for representing flags, small integers, or other compact data structures. They save memory by limiting the number of bits used for each field.

---

10. Concept of Endianness and Difference Between Little Endian and Big Endian

Endianness defines the byte order used for storing multi-byte data types:

- **Little Endian:** Stores the least significant byte at the smallest memory address (e.g., Intel processors).
- **Big Endian:** Stores the most significant byte at the smallest memory address (e.g., some network protocols and older processors).

---

11. Build Process of C Programs

- **Preprocessing:** Handles macros and file inclusions before actual compilation begins.
- **Compilation:** Translates preprocessed code into assembly language. → .s

- **Assembly:** Converts assembly code into machine code, producing object files. -> .o
- **Linking:** Combines object files and libraries into a single executable, resolving references and addresses. -> .exe, .hex, .bin

---

12. Concept of Context Switching in C

Firstly it saves the current address, then it jumps to the address of the function, executes the function, and then returns to the address saved in the link register (LR).

---

13. Difference Between the Linker and the Linker Script

- **Linker:** A tool that combines object files and libraries into a final executable, handling symbol resolution and address allocation.
- **Linker Script:** A script that provides instructions to the linker on how to place code and data sections in memory, allowing for custom memory layout.

---

14. Basic Concepts of What Happens Before main() is Called in C

Before the main() function executes, several initializations take place in the startup_file:

- **Static Initializations:** Initializes global and static variables to their default values.
- **Constructor Functions:** In C++, global constructors are called to initialize global objects before main() is executed.
- **Calls main()**

15. Preprocessor Directives

Preprocessor directives are instructions to the compiler to preprocess code before actual compilation, some of them are:

- **#include:** Includes external files or libraries.
- **#define:** Defines macros or symbolic constants.
- **#ifdef:** Conditional compilation based on macro definitions.

16. Compare Between Inline Functions and Macro-like Functions in C

- **Inline Functions:** Request the compiler to insert the function's code directly at the call site, potentially improving performance by avoiding function call overhead.
- **Macros:** Preprocessor directives that perform text substitution before compilation, offering a way to define constants or inline code, but lacking type safety and scope control.

17. Expression *(&ADC_Readings + 1) - ADC_Readings:

This expression first calculates the address of the element immediately after the ADC_Readings array, then dereferences that address to get the value at that location. Finally, it subtracts the starting address of the ADC_Readings array to get the difference in bytes between the two addresses. This effectively gives you the size of one element in the ADC_Readings array, which should be the size of an unsigned char.

18. Interrupts:

Interrupts are signals sent by hardware or software to the processor, causing the current program execution to be suspended and a specific interrupt service routine (ISR) to be executed.

---

19. Interrupt Vector Table (IVT):

The Interrupt Vector Table (IVT) is a data structure that contains the memory addresses of the interrupt service routines (ISRs) for each interrupt source. When an interrupt occurs, the processor uses the IVT to jump to the corresponding ISR and execute the necessary interrupt-handling code.

References

1. Null Pointer in C - Wikipedia: https://en.wikipedia.org/wiki/Null_pointer
2. Void Pointers - Wikibooks: https://en.wikibooks.org/wiki/C_Programming/Pointer_Overview#Void_Pointers
3. Function Pointers in C - CProgramming: https://www.cprogramming.com/tutorial/function-pointers.html
4. Wild Pointers - GeeksforGeeks: https://www.geeksforgeeks.org/what-are-wild-pointers/
5. Dynamic Memory Allocation in C - Tutorialspoint: https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm
6. Pointers and Arrays - GeeksforGeeks: https://www.geeksforgeeks.org/difference-between-arrays-and-pointers/
7. Passing by Reference in C - Tutorialspoint: https://www.tutorialspoint.com/cprogramming/c_function_pass_by_reference.htm
8. Arrays vs. Pointers - GeeksforGeeks: https://www.geeksforgeeks.org/arrays-vs-pointers-in-c/
9. Pointer Arithmetic - Cplusplus.com: https://www.cplusplus.com/doc/tutorial/pointers/
10. Memory Layout of C Programs - GeeksforGeeks: https://www.geeksforgeeks.org/memory-layout-of-c-program/
11. Signed vs. Unsigned Data Types - GeeksforGeeks: https://www.geeksforgeeks.org/signed-and-unsigned-in-c/
12. Embedded Systems Data Types - Embedded.com: https://www.embedded.com/understanding-data-types-in-embedded-systems/

13. Struct vs. Union - Tutorialspoint: https://www.tutorialspoint.com/cprogramming/c_structures.htm
14. Unions in C - GeeksforGeeks: https://www.geeksforgeeks.org/union-in-c/
15. Bitwise Operators in C - GeeksforGeeks: https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/
16. Bitfields in C - GeeksforGeeks: https://www.geeksforgeeks.org/bit-fields-in-c/
17. Endianness Explained - GeeksforGeeks: https://www.geeksforgeeks.org/endianism/
18. C Build Process - Tutorialspoint: https://www.tutorialspoint.com/compilers/build-process-of-a-c-program
19. Context Switching - GeeksforGeeks: https://www.geeksforgeeks.org/context-switching/
20. Linker - GeeksforGeeks: https://www.geeksforgeeks.org/linker-and-loader/
21. Linker Scripts - GNU: https://www.gnu.org/software/binutils/manual/ld-2.35/html_node/Linker-Scripts.html
22. Before main() Execution - GeeksforGeeks: https://www.geeksforgeeks.org/what-happens-before-main-function-in-c/
23. Preprocessor Directives - Tutorialspoint: https://www.tutorialspoint.com/cprogramming/c_preprocessors.htm
24. Inline Functions vs Macros - GeeksforGeeks: https://www.geeksforgeeks.org/inline-functions-vs-macros-in-c/