1. \*Explain the purpose of dynamic memory allocation in C. Discuss the differences between stack memory and heap memory.\*

Dynamic memory allocation in C is used to allocate memory at runtime using functions like malloc(), calloc(), etc. This is in contrast to static memory allocation, which occurs at compile time. The difference between stack memory (for static allocation) and heap memory (for dynamic allocation) includes the lifespan, management, and size of the memory. The stack is managed by the CPU and is limited in size, but fast. The heap is managed by the programmer, can be much larger, but is slower and requires manual cleanup.

2. \*\*Compare malloc and calloc. What is the key difference between them? Provide an example demonstrating their usage.\*\*

malloc allocates a single block of memory without initializing it, while calloc allocates multiple blocks and initializes them to zero. Example:

c

int \*ptr = (int\*)malloc(5 \* sizeof(int)); // allocates memory for 5 integers

int \*arr = (int\*)calloc(5, sizeof(int)); // allocates and initializes memory for 5 integers

3. \*\*Explain the purpose of realloc. Provide a scenario where resizing an existing memory block is necessary.\*\*

realloc is used to resize a previously allocated memory block. A scenario could be expanding an array when the number of elements to be stored exceeds the current capacity.

4. \*What is a dangling pointer? How can you avoid it?\*

A dangling pointer is a pointer that points to a memory location that has been freed/deallocated. To avoid this, pointers should be set to NULL after freeing the memory they point to.

5. \*\*Explain the difference between free and delete in C and C++. When should you use each?\*\*

free is used in C to deallocate memory allocated by malloc, calloc, or realloc. delete is used in C++ to deallocate memory allocated with new.

6. \*What happens if you forget to free dynamically allocated memory? How can memory leaks impact program performance?\*

If dynamically allocated memory is not freed, it leads to memory leaks, causing the application to use more memory than necessary, which can lead to reduced performance and eventually, in extreme cases, cause the application to crash due to memory exhaustion.

7. \*Explain the concept of an array of pointers. How can you use it to store addresses of dynamically allocated memory blocks?\*

An array of pointers is a collection of pointer variables in a contiguous memory layout, where each pointer can store the address of a dynamically allocated memory block. This is useful for creating a dynamically allocated multi-dimensional array.

8. \*\*What is the return type of malloc and calloc? Why is it void \*?\*\*

The return type is void \* because these functions return a pointer to a memory block that has not been typed yet, providing a generic pointer that can be cast to any desired data type.

9. \*Explain the difference between stack memory and heap memory in terms of storage duration, allocation, and deallocation.\*

This is similar to question 1. Stack memory has automatic storage duration (managed by the scope of functions), while heap memory's storage duration lasts until it is explicitly freed. Allocation and deallocation on the stack are managed by the system, but on the heap, they are managed by the programmer.

### Assignment 2

1. \*Explain the purpose of preprocessor directives in C. How do they differ from regular C statements?\*

Preprocessor directives provide instructions to the compiler to preprocess the information before actual compilation starts. They differ from regular C statements because they are not part of the C language itself and are not executed at runtime.

2. \*\*What is the purpose of the #include directive? Provide an example.\*\*

The #include directive is used to include the contents of a file into the current source file. Example:

c

#include <stdio.h> // includes the standard input-output library

3. \*Define macros in C. How are they useful?\*

Macros are defined using #define and represent code fragments that are expanded by the preprocessor. They are useful for constants, code reuse, and conditional compilation.

4. \*\*Explain the difference between #define and #undef.\*\*

#define is a preprocessor directive used to create macros, which are essentially placeholders for various values or blocks of code that are replaced by those values or code blocks when the program is compiled. It can be used to define constants or functions in a manner that allows easy changes and can be used for conditional compilation as well.

Example of #define:

c

#define PI 3.14159

#undef is used to undefine a macro that was previously defined using #define. This is useful for ensuring that a macro is not accidentally used where it's not intended, or to redefine it differently later in the code.

Example of using #undef:

c

#define PI 3.14159

// ... some code that uses PI

#undef PI

// PI is no longer defined from this point onward

By undefining a macro, you can ensure it doesn't affect subsequent code, which is useful if the macro was only needed for a particular section or could cause conflicts if used elsewhere.

5. \*\*What is conditional compilation? How does the #ifdef directive work?\*\*

Conditional compilation allows code to be compiled or omitted based on defined conditions. The #ifdef directive checks if a preprocessor constant is defined and compiles the code between #ifdef and #endif only if the constant is indeed defined.

6. \*Discuss the purpose of pragmas in C. Provide an example.\*

Pragmas in C provide additional information to the compiler, often for the purpose of optimization or other specific behaviors. An example is #pragma once, which indicates that the file should only be included once in a single compilation, a guard against multiple inclusions.

7. \*How can you avoid multiple inclusions of the same header file?\*

Multiple inclusions can be avoided using header guards with #ifndef, #define, and #endif directives, or by using #pragma once at the beginning of the header file.

8. \*\*Explain the concept of conditional compilation using #ifdef and #else. Provide an example.\*\*

Conditional compilation with #ifdef and #else allows for different sections of code to be compiled based on whether a preprocessor constant is defined or not. Example:

c

#ifdef DEBUG

printf("Debugging is enabled.\n");

#else

printf("Debugging is disabled.\n");

#endif

9. \*\*What is the purpose of the #error directive? How does it impact compilation?\*\*

The #error directive generates a compilation error with a specified message. It can be used to flag when a certain condition is not met during compilation. The compilation is stopped with an error message.

10. \*\*Discuss the significance of the #pragma once directive. How does it prevent multiple inclusions?\*\*

The #pragma once directive tells the compiler to include the file only once per compilation, which prevents problems related to multiple definitions and potential compile-time increase due to redundant processing of the same file.