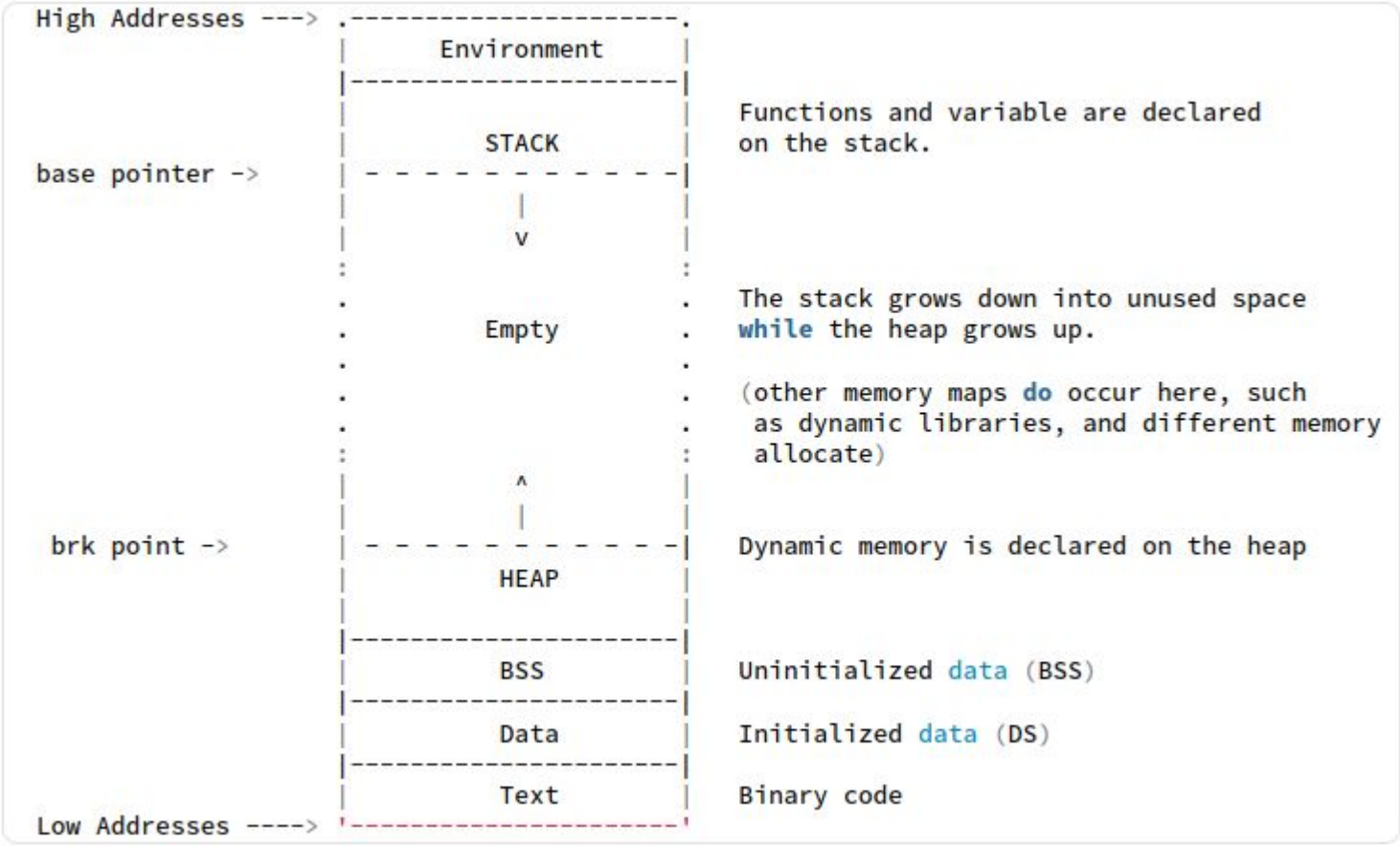
Dynamic Memory Allocation

Memory Layout



Agenda

- Types of memory in C
  - Static
  - Stack
  - Heap
  - Automatic
- Functions used for dynamic memory allocation
  - Malloc
  - Calloc
  - Realloc
  - free

Types of mem

In C, there are several types of memory allocation techniques available. Here are the most common ones:

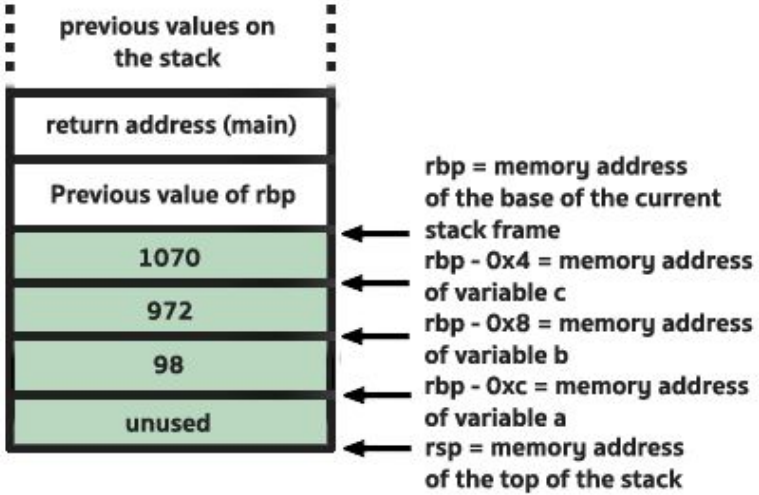
1. Stack Allocation:

2. Heap Allocation:

3. Static Allocation:

4. Automatic Allocation:

1. Stack Allocation



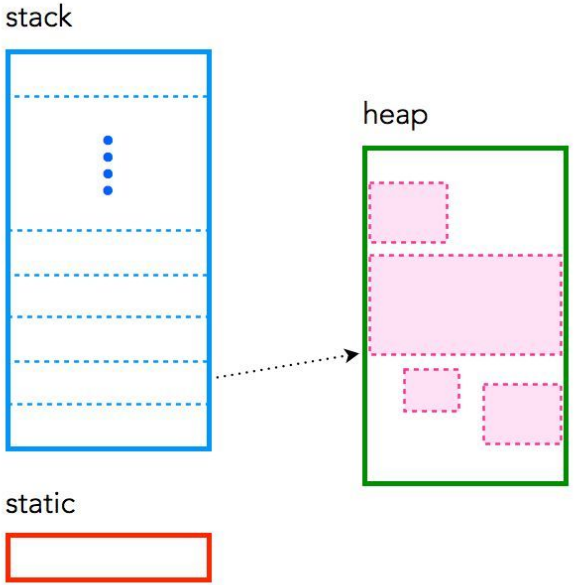
In stack allocation, memory is allocated from the **stack**, which is a region of memory used for *local variables and function calls*. When a function is called, the compiler automatically allocates memory for local variables on the stack, and when the function returns, that memory is automatically freed. Stack allocation is fast and efficient but **limited in size**.

Use cases of Stack Memory Allocation:

Local Variables: Stack memory allocation is primarily used for local variables within functions. When a function is called, the compiler allocates memory for local variables on the stack. The memory is automatically freed when the function returns. Stack allocation is efficient and provides automatic memory management.

Function Call Stack: Stack memory allocation is also used to manage the function call stack, which keeps track of function calls, local variables, and return addresses. Each function call creates a new stack frame, which is pushed onto the stack, and when the function returns, the stack frame is popped, and the previous function resumes execution.

2. Heap Allocation:



Heap allocation involves allocating memory dynamically from the heap, which is a larger region of memory managed by the program. Memory allocated on the heap persists until explicitly **free**d by the programmer. Heap allocation provides flexibility in terms of memory size and lifetime but requires **manual memory management.**

cont

In C, you can use the `**malloc()**`, `**calloc()**`, `**realloc()**`, and `**free()**` functions to **allocate** and **deallocate** memory on the heap.

- **malloc()** allocates a block of memory of a specified size and returns a pointer to the first byte of the allocated block.
- **calloc()** is similar to **malloc()**, but it also initializes the allocated memory to zero.
- **realloc()** allows you to resize a previously allocated block of memory.
- **free()** deallocates a block of memory previously allocated with **malloc() calloc(),** or **realloc()**.

Use cases of Heap Memory Allocation:

Dynamic Data Structures: Heap memory allocation is commonly used for creating dynamic data structures such as *linked lists, trees, or dynamically-sized arrays*. Using functions like malloc() or calloc(), you can allocate memory on the heap to store these structures, and they can grow or shrink as needed during program execution.

Large Data Objects: Heap allocation is suitable for large data objects that may exceed the limitations of the stack. Since the heap provides a larger memory space, it can accommodate larger objects that may not fit within the stack's limited size.

Persistent Data: Heap memory can be used to store data that needs to persist beyond the scope of a single function or block. By allocating memory on the heap, you can control the lifetime of the data and access it from multiple parts of the program.

3. Static Allocation:

Static allocation involves declaring variables with the `**static**` keyword, which allocates memory in a static data area. Static variables have a lifetime that spans the entire program execution and retain their values across function calls. Memory for static variables is allocated and deallocated automatically by the compiler. E.g global variables

Use cases of Static Memory Allocation:

Global Variables: Static memory allocation is commonly used for global variables. Global variables are declared outside of any function, and they retain their values throughout the execution of the program. They are allocated in the static memory area and are accessible from any part of the program.

Static Local Variables: Static memory allocation is also used for static local variables within a function. Static local variables have a lifetime that extends across multiple function calls, and their values persist between function invocations.

Constant Data: Static memory allocation is suitable for storing constant data, such as string literals or arrays with fixed values. These constants are stored in the static memory area and can be accessed throughout the program.

4. Automatic Allocation:

Automatic allocation is used for variables declared within a block or function without the `static` keyword. These variables are allocated on the stack and have a limited lifetime within the scope where they are defined. Memory is automatically allocated and deallocated when the block or function is entered and exited, respectively.

It's important to note that both stack and static allocations are determined at compile-time, while heap allocation allows for dynamic memory management at runtime. It's essential to manage dynamically allocated memory carefully to avoid memory leaks or accessing freed memory.

Memory leak

Memory Leak is a situation when we get some memory on the heap and do not free it when we are done using it. so our application is holding on to some unused memory in the heap, but why do we call it this situation **Memory Leak** and why does it happen due to improper use of dynamic memory only, due to improper use of heap only and not some other section of application memory.

Memory leaks in C happen for three core reasons:

1. we do not free the memory that is no longer needed
2. we do try to free the memory but we do not have the reference to it (dangling pointer)
3. we try to free the memory using the wrong function

Resources

1. Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc()
2.

See you at the next session!