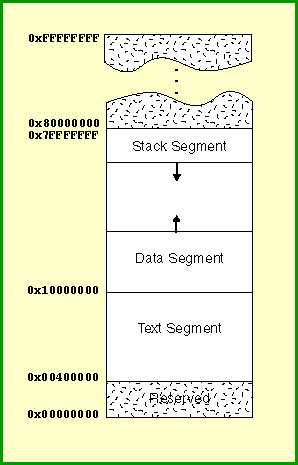
**Dynamic Memory Allocation**

**Dynamic memory allocation** is when an executing program requests that the operating system give it a range of addresses in main memory that are not yet being used. The program then uses this memory for its own purpose. Usually the purpose is to add a node to a data structure. In object oriented languages dynamic memory allocation is used to get the memory for a new object.

The memory comes from above the static part of the data segment. Programs may both request memory and also return previously dynamically allocated memory. The heap may have "holes" in it where previously allocated memory has been made free and is now available for other purposes.

A new dynamic request for memory might return a range of addresses out of one of the holes. But it might not use up all the hole, so further dynamic requests might be satisfied out of the original hole. Keeping track of allocated and deallocated memory is complicated. A modern operating system does this as some of its services.

**SPIM Dynamic Memory**

A stack is an easyly managed structure. Only a few memory addresses are needed to keep track of it. (Some of these addresses are in the stack pointer and in the frame pointer registers.) As a program executes the stack grows up and down and up and down as subroutines are called and exited. The heap is more like a book shelf. Books are constantly being taken off the shelf from various locations, leaving gaps, and then later returned, filling the gaps.

Here is how a SPIM program requests a block of memory from SPIM's heap:

li $a0,xxx # $a0 contains the number of bytes you need.

# This must be a multiple of four.

li $v0,9 # code 9 == allocate memory

syscall # Invoke the trap handler.

# $v0 <-- the address of the first byte

# of the dynamically allocated block

You don't know in advance what range of addresses you will get back for the allocate memory request. The trap handler returns the first address of a contiguous block of ($a0) number of bytes. (This is similar to a call to malloc() in "C".)

**Dynamic allocation and Static system:**

There are two types of memory in our machine, one is Static Memory and another one is Dynamic Memory; both memories are managed by our Operating System. Our operating system helps us in the allocation and deallocation of memory blocks either during compile-time or during the run-time of our program.  
When the memory is allocated during compile-time it is stored in the Static Memory and it is known as Static Memory Allocation, and when the memory is allocated during run-time it is stored in the Dynamic Memory and it is known as Dynamic Memory Allocation.

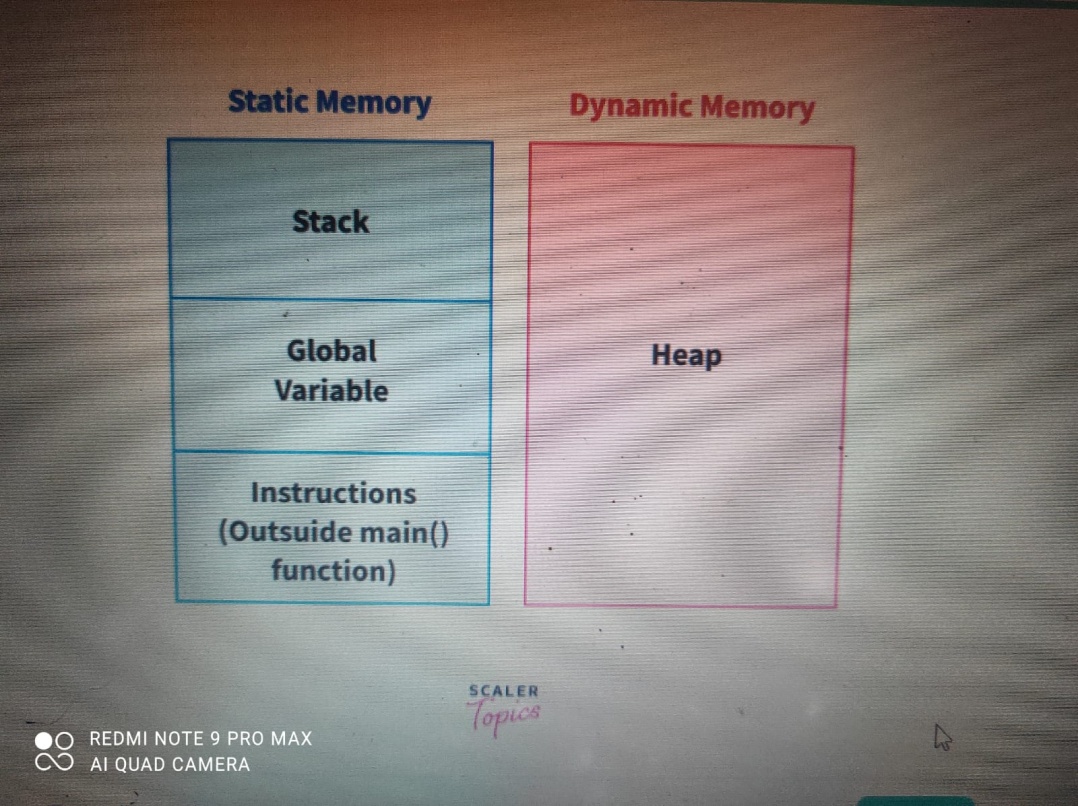
**Now, let us see some differences between Static Memory Allocation and Dynamic Memory Allocation:**

| **Static Memory Allocation** | **Dynamic Memory Allocation** |
| --- | --- |
| Constant (Invariable) memory is reserved at **compile-time** of our program that **can't** be modified. | Dynamic (Variable) memory is reserved at **run-time** of our program that **can** be modified. |
| It is used at compile-time of our program and is also known as **compile-time memory allocation**. | It is used at run-time of our program and is also known as **run-time memory allocation**. |
| We **can't** allocate or deallocate a memory block during run-time. | We **can** allocate and deallocate a memory block during run-time. |
| [Stack](https://www.scaler.com/topics/c/dynamic-memory-allocation-in-c/#Stack) space is used in Static Memory Allocation. | [Heap](https://www.scaler.com/topics/c/dynamic-memory-allocation-in-c/#Heap) space is used in Dynamic Memory Allocation. |
| It **doesn't provide** reusability of memory while the program is running. So, it is **less** efficient. | It **provides** reusability of memory while the program is running. So, it is **more** efficient. |

When we execute a C Program, it requires to reserve some memory in the machine to store its variables, functions, instructions and the program file itself. However, we also have a memory segment that we can use dynamically as much memory as the system have and that too during the run-time of a program. So, now let us see the components/segments of memory that are used during the execution of a C program.

There are further four components in which our system's memory is divided:

1. Stack Segment (Static Memory)
2. Global Variables Segment (Static Memory)
3. Instructions / Text Segment (Static Memory)
4. Heap Segment (Dynamic Memory)



The amount of memory allocated for *Stack, Global Variables, and Instructions / Text* during compile-time is invariable and cannot be reused until the program execution finishes. However, the *Heap* segment of the memory can be used at run-time and can be expanded until the system's memory exhausts.

Let's look at some important points related to all the memory segments :

**Instructions / Text**

* Text outside the main() function is stored in the **Static Memory**.
* These instructions are stored during compile-time of a program.

**Global Variables**

* Global variables also known as static variables and can be declared by two methods,
  + Using static keyword, ex. static int i = 0;
  + Declaring variable outside main() or any other function.
* These variables are stored in the *Static Memory* during the compile-time of our program.

**Stack**

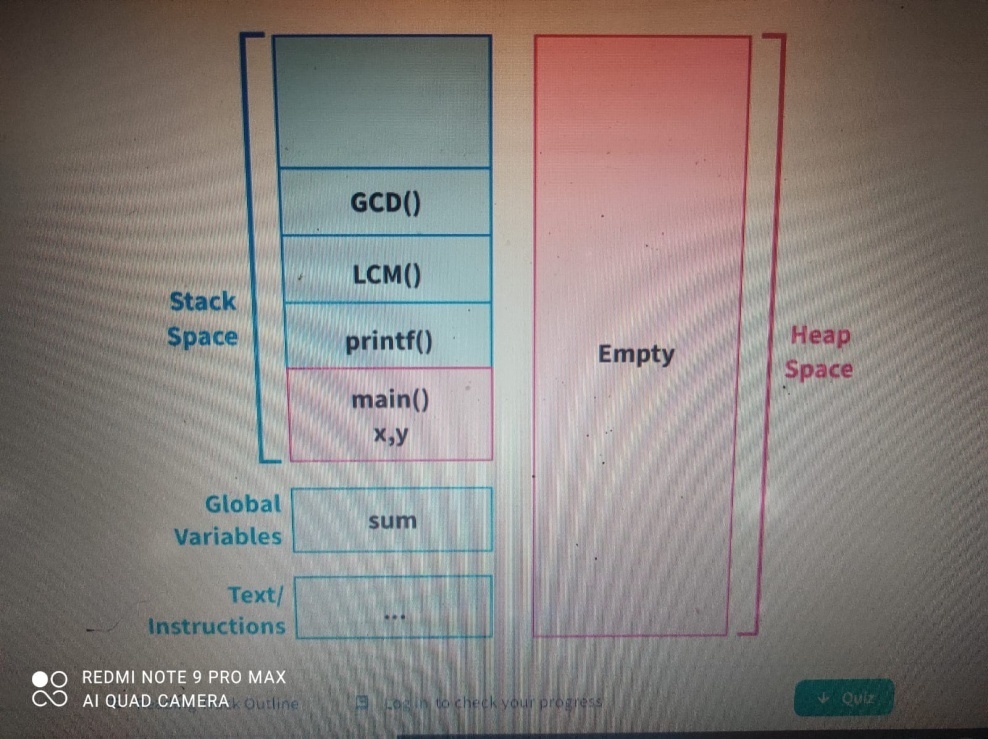
* Stack is a constant space (memory) allocated by our operating system to store local variables, function calls and local statements that are present in the function definition. It is the major part of the **Static Memory** in our system.
* There are some drawbacks of stack space as follows:
  + The memory allocated for stack can't grow during the run-time of an application.
  + We can't allocate or deallocate memory during execution.

**Heap**

* Heap is the **Dynamic Memory** of our program, It can also be imagined as a big free pool of memory available to us. The space occupied by the heap section is not fixed, it can vary during the run-time of our program, and there are functions to perform allocation and deallocation of memory blocks during run-time.
* Heap space can grow as long as we do not run out of the system's memory itself, however it is not in the best interest of a programmer to exhaust the system's memory, so we need to be really careful while using the heap space in our program.

**Note :** A machine's memory is one of the most useful resources available to us that can be used as both *Statically* and *Dynamically*. So, we have to make sure to manage it well and not waste it during the execution of our program.

**Explanation of Memory Allocation Process :**

****

* The first rectangular box represents the memory reserved for text/instructions, second box as memory reserved for global variables section, third box as the space reserved for stack and fourth box represents the heap space.
* When we run our program, first the main() function is called, then some amount of memory from the stack frame is allocated for the execution of main() function and as sum is a global variable, it has been given a space in the global segment.
* *Heap* space remains empty throughout the execution of the program as there is no allocation of memory during run-time.
* The amount of memory allocated for a function in the stack can be called as a stack frame. All of the local variables, arguments and the information in a function are stored within the stack frame allocated to that function, and as soon as a function returns something or reaches its last statement, it is cleared from the stack memory.
* When the main() finishes, program also finishes. In the end, space occupied by our global variables also gets cleared.

## Introduction to Dynamic Memory Allocation in C

**Dynamic Memory Allocation** is a process in which we allocate or deallocate a block of memory during the run-time of a program.

There are four functions malloc(), calloc(), realloc() and free() present in <stdlib.h> header file that are used for Dynamic Memory Allocation in our system. It can also be referred to as a procedure to use Heap Memory in which we can vary the size of a variable or Data Structure (such as an Array) during the lifetime of a program using the library functions.

Dynamic Memory Allocation is considered as a very important concept in the field of Data Structures and is used in almost every Data Structures like Linked Lists, Stacks, Dynamic Arrays, Queue, etc.

## C malloc() Method

malloc() is a method in C which is used to allocate a memory block in the heap section of the memory of some specified size (in bytes) during the run-time of a C program. It is a library function present in the <stdlib.h> header file.

### Syntax of malloc()

* General Syntax:

(cast-data-type \*)malloc(size-in-bytes);

malloc() function takes size in bytes as an argument and returns a void pointer, so we have to typecast the malloc() function to the required data type.

malloc() does not initialise the allocated memory block, so initially it contains a garbage value.

* Defined in header <stdlib.h> as : void\* malloc( size\_t size );
  + size : number of bytes to be allocated
* Example Syntax:

int \*ptr = (int \*)malloc(sizeof(int));

In the above statement, we have assigned and allocated a block of size 4-bytes (64-bit compiler) with cast data type as integer to an integer pointer ptr during the run-time of our program.

We can simply pass number of bytes instead of using sizeof() in argument, both are correct, but sizeof() automatically takes care of the type of compiler and operating system that can cause changes in sizes of different data types on different compilers and operating systems, so it is recommended to use sizeof() function in Dynamic Memory Allocation.

**OR:**

**Memory Allocation:** Memory allocation is a process by which computer programs and services are assigned with physical or virtual memory space. The memory allocation is done either before or at the time of program execution. There are two types of memory allocations: 

1. [Compile-time or Static Memory Allocation](https://www.geeksforgeeks.org/difference-between-static-allocation-and-heap-allocation/)
2. [Run-time or Dynamic Memory Allocation](https://www.geeksforgeeks.org/what-is-dynamic-memory-allocation/)

**Static Memory Allocation:** Static Memory is allocated for declared variables by the compiler. The address can be found using the [*address of*](https://www.geeksforgeeks.org/address-function-c-cpp/) operator and can be assigned to a pointer. The memory is allocated during compile time.

**Dynamic Memory Allocation:** Memory allocation done at the time of execution(run time) is known as **dynamic memory allocation**. Functions [calloc() and malloc()](https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/) support allocating dynamic memory. In the Dynamic allocation of memory space is allocated by using these functions when the value is returned by functions and assigned to pointer variables.

**Tabular Difference Between Static and Dynamic Memory Allocation in C:**

|  |  |  |
| --- | --- | --- |
| S.No | Static Memory Allocation | Dynamic Memory Allocation |
| 1 | In the static memory allocation, variables get allocated permanently, till the program executes or function call finishes. | In the Dynamic memory allocation, variables get allocated only if your program unit gets active. |
| 2 | Static Memory Allocation is done before program execution. | Dynamic Memory Allocation is done during program execution. |
| 3 | It uses [stack](https://www.geeksforgeeks.org/stack-data-structure/) for managing the static allocation of memory | It uses [heap](https://www.geeksforgeeks.org/heap-data-structure/) for managing the dynamic allocation of memory |
| 4 | It is less efficient | It is more efficient |
| 5 | In Static Memory Allocation, there is no memory re-usability | In Dynamic Memory Allocation, there is memory re-usability and memory can be freed when not required |
| 6 | In static memory allocation, once the memory is allocated, the memory size can not change. | In dynamic memory allocation, when memory is allocated the memory size can be changed. |
| 7 | In this memory allocation scheme, we cannot reuse the unused memory. | This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it. |
| 8 | In this memory allocation scheme, execution is faster than dynamic memory allocation. | In this memory allocation scheme, execution is slower than static memory allocation. |
| 9 | In this memory is allocated at compile time. | In this memory is allocated at run time. |
| 10 | In this allocated memory remains from start to end of the program. | In this allocated memory can be released at any time during the program. |
| 11 | **Example:** This static memory allocation is generally used for [array](https://www.geeksforgeeks.org/introduction-to-arrays/). | **Example:** This dynamic memory allocation is generally used for [linked list](https://www.geeksforgeeks.org/data-structures/linked-list/). |

# Memory Allocation:

**Memory allocation** is an action of assigning the physical or the virtual memory address space to a process (its instructions and data). The two fundamental methods of memory allocation are static and dynamic memory allocation.

The static memory allocation method assigns the memory to a process, **before its execution**. On the other hand, the dynamic memory allocation method assigns the memory to a process, **during its execution**.

In this section, we will be discussing what is memory allocation, its types (static and dynamic memory allocation) along with their advantages and disadvantages. So let us start.

### Content: Static and Dynamic Memory Allocation

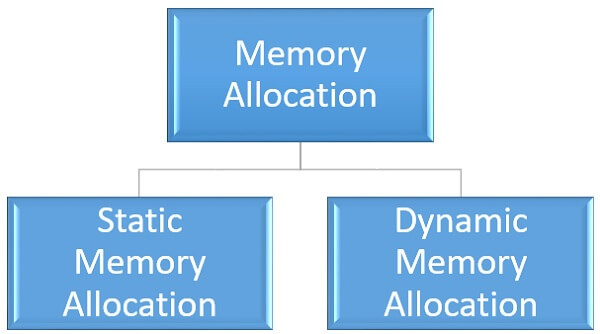
1. [Memory Allocation](https://binaryterms.com/static-and-dynamic-memory-allocation.html#MemoryAllocation)
2. [Types of Memory Allocation](https://binaryterms.com/static-and-dynamic-memory-allocation.html#TypesofMemoryAllocation)
3. [Advantages and Disadvantages of Static and Dynamic Memory Allocation](https://binaryterms.com/static-and-dynamic-memory-allocation.html#AdvantagesandDisadvantagesofStaticandDynamicMemoryAllocation)
4. [Key Takeaways](https://binaryterms.com/static-and-dynamic-memory-allocation.html#KeyTakeaways)

## Memory Allocation

To get a process executed it must be first placed in the memory. Assigning space to a process in memory is called memory allocation. Memory allocation is a general aspect of the term **binding.**

Let us understand binding with the help of an example. Suppose, there is an **entity** in a program, with a set of attributes. Now, a **variable** of this entity will have values for this **set of attributes**. For storing these values, we must have memory allotted to these attributes.

So, the act of assigning the memory address to the attribute of the variable is called **memory allocation**. And the act of specifying/binding the values to the attributes of the variable is called **binding**. This action of binding must be performed before the variable is used during the execution of the program.

We have two types of memory allocation or we can say two methods of binding, static and dynamic binding.  
 

## Types of Memory Allocation

### 1. Static Memory Allocation

Static memory allocation is performed when the compiler compiles the program and generates object files. The linker merges all these object files and creates a single executable file. The loader loads this single executable file in the main memory, for execution. In static memory allocation, the size of the data required by the process must be known **before** the execution of the process initiates.

If the data sizes are not known before the execution of the process, then they have to be guessed. If the data size guessed is larger than the required, then it leads to **wastage** of memory. If the guessed size is smaller, then it leads to inappropriate execution of the process.

The static memory allocation method does not need any memory allocation operation during the execution of the process. All the memory allocation operation required for the process is done before the execution of the process has started. So, it leads to **faster** execution of a process.

Static memory allocation provides more **efficiency** when compared to dynamic memory allocation.

### 2. Dynamic Memory Allocation

Dynamic memory allocation is performed while the program is in execution. Here, the memory is allocated to the entities of the program when they are to be used for the **first time** while the program is running.

The actual size, of the data required, is known at the run time so, it allocates the **exact** memory space to the program thereby, reducing the memory wastage.

Dynamic memory allocation provides **flexibility** to the execution of the program. As it can decide what amount of memory space will be required by the program. If the program is large enough then a dynamic memory allocation is performed on the different parts of the program, which is to be used currently. This reduces memory wastage and improves the performance of the system.

Allocating memory dynamically creates an overhead over the system. Some allocation operations are performed repeatedly during the program execution creating more overheads, leading in **slow** execution of the program.

Dynamic memory allocation does not require special support from the operating system. It is the responsibility of the programmer to design the program in a way to take advantage of dynamic memory allocation method.

Thus the dynamic memory allocation is flexible but slower than static memory allocation.

### Advantages of static and dynamic memory allocation

#### Static Memory Allocation

1. Static memory allocation provides an **efficient**way of assigning the memory to a process.
2. All the memory assigning operations are done before the execution starts. So, there are no **overheads** of memory allocation operations at the time of execution of the program.
3. Static memory allocation provides **faster**execution, as at the time of execution it doesn’t have to waste time in allocation memory to the program.

#### Dynamic Memory Allocation

1. Dynamic memory allocation provides a **flexible**way of assigning the memory to a process.
2. Dynamic memory allocation **reduces**the memory **wastage**as it assigns memory to a process during the execution of that program. So, it is aware of the exact memory size required by the program.
3. If the program is large then the dynamic memory allocation is performed on the different parts of the program. Memory is assigned to the part of a program that is currently in use. This also reduces memory wastage and indeed improves **system performance**.

### Disadvantages of static and dynamic memory allocation

#### Static Memory Allocation

1. In static memory allocation, the system is **unaware**of the memory requirement of the program. So, it has to guess the memory required for the program.
2. Static memory allocation leads to memory **wastage**. As it estimates the size of memory required by the program. So, if the estimated size is larger, it will lead to memory **wastage**else if the estimated size is smaller, then the program will execute **inappropriately**.

#### Dynamic Memory allocation

1. Dynamic memory allocation method has an **overhead**of assigning the memory to a process during the time of its execution.
2. Sometimes the memory allocation actions are repeated several times during the execution of the program which leads to more **overheads**.
3. The overheads of memory allocation at the time of its execution **slowdowns**the execution to some extent.

**Process Stack management:**

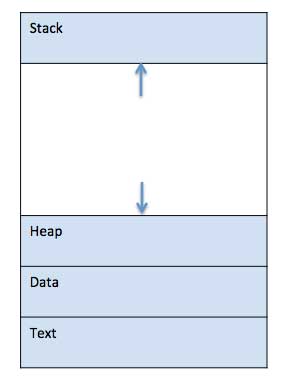
## Process:

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



|  |  |
| --- | --- |
| **S.N.** | **Component & Description** |
| 1 | **Stack**  The process Stack contains the temporary data such as method/function parameters, return address and local variables. |
| 2 | **Heap**  This is dynamically allocated memory to a process during its run time. |
| 3 | **Text**  This includes the current activity represented by the value of Program Counter and the contents of the processor's registers. |
| 4 | **Data**  This section contains the global and static variables. |

AD

## Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

#include <stdio.h>

int main() {

printf("Hello, World! \n");

return 0;

}

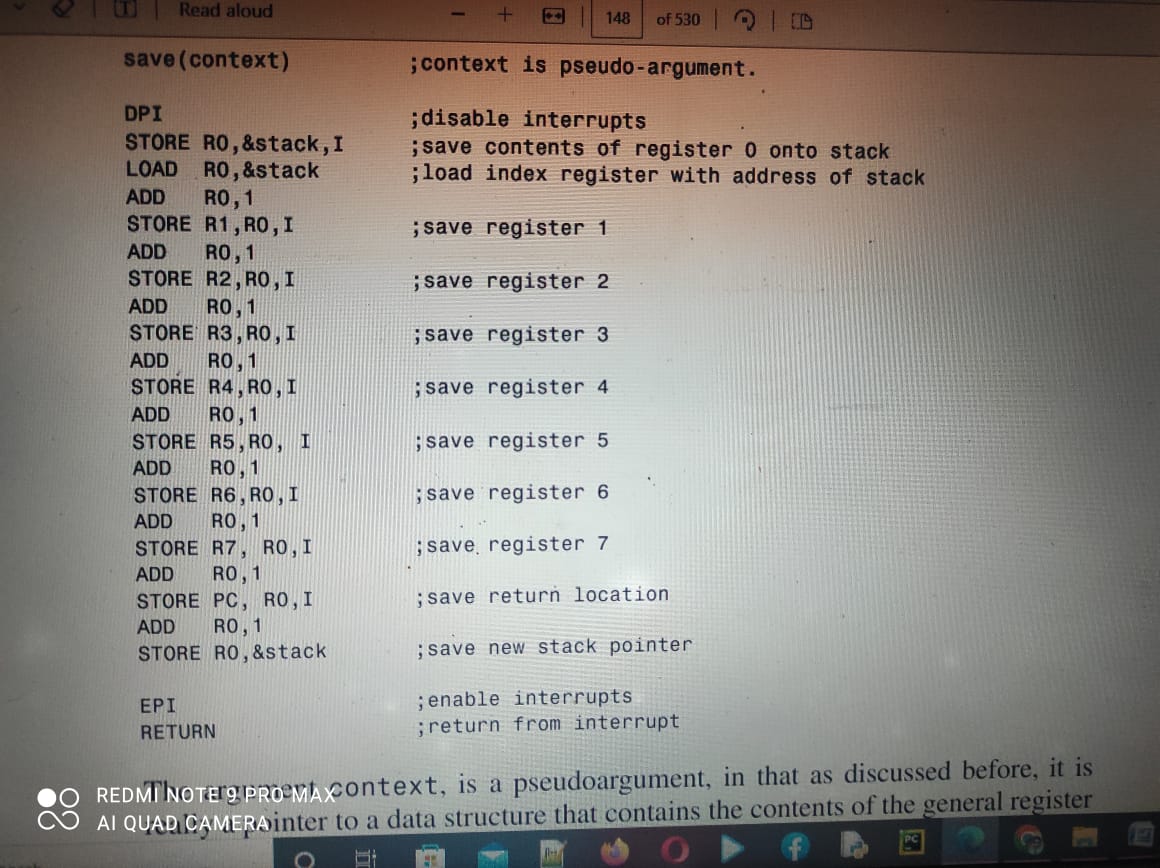
A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

**Process Stack Management:**

In a multitasking system, context for each task needs to be saved and restored in order to switch processes. This can be done by using one or more run-time stacks or the task-control block model. Run-time stacks work best for interruptonly systems and foreground/background systems, whereas the task-control block model works best with full-featured real-time operating systems

Managing the Stack If a run-time stack is to be used to handle the run-time saving and restoring of context, two simple routines – save and restore – are necessary. The save routine is called by an interrupt handler to save the current context of the machine into a stack area. To present disaster, this call should be made immediately after interrupts have been disabled. The restore routine should be called just before interrupts are enabled and before returning from the interrupt handler. Consider, for example, the implementation of the save routine. Assume the global variable stack is to point to the top of the stack and that eight genera registers (R0–R7) are to be saved on a stack. The memory location “PC” corresponds to the interrupt return vector location, and so it contains the program counter value at the time of interruption. It is necessary to save this on the stack to allow stacking of interrupts. The pseudocode for a 2-address architecture works by saving a pointer to the top of the stack and uses an “indirect: mode represented by “,I” to sequentially store the other registers.

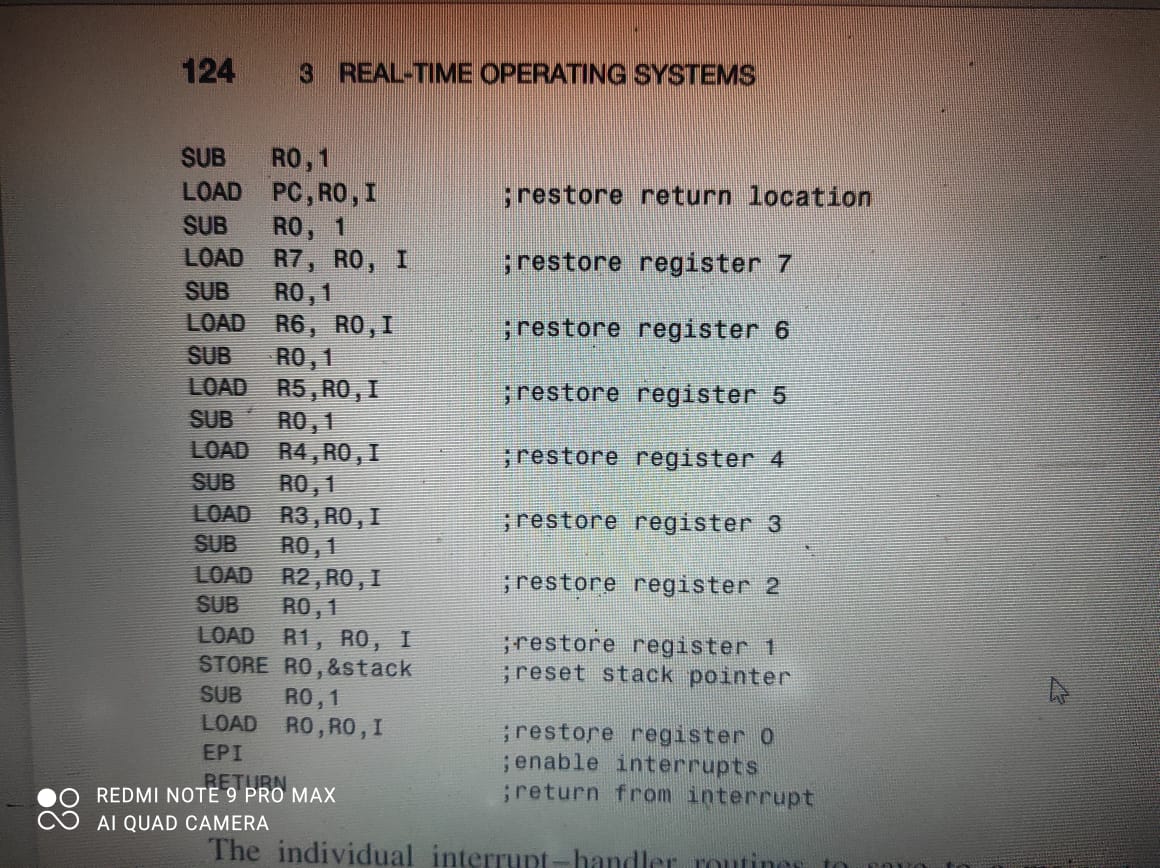


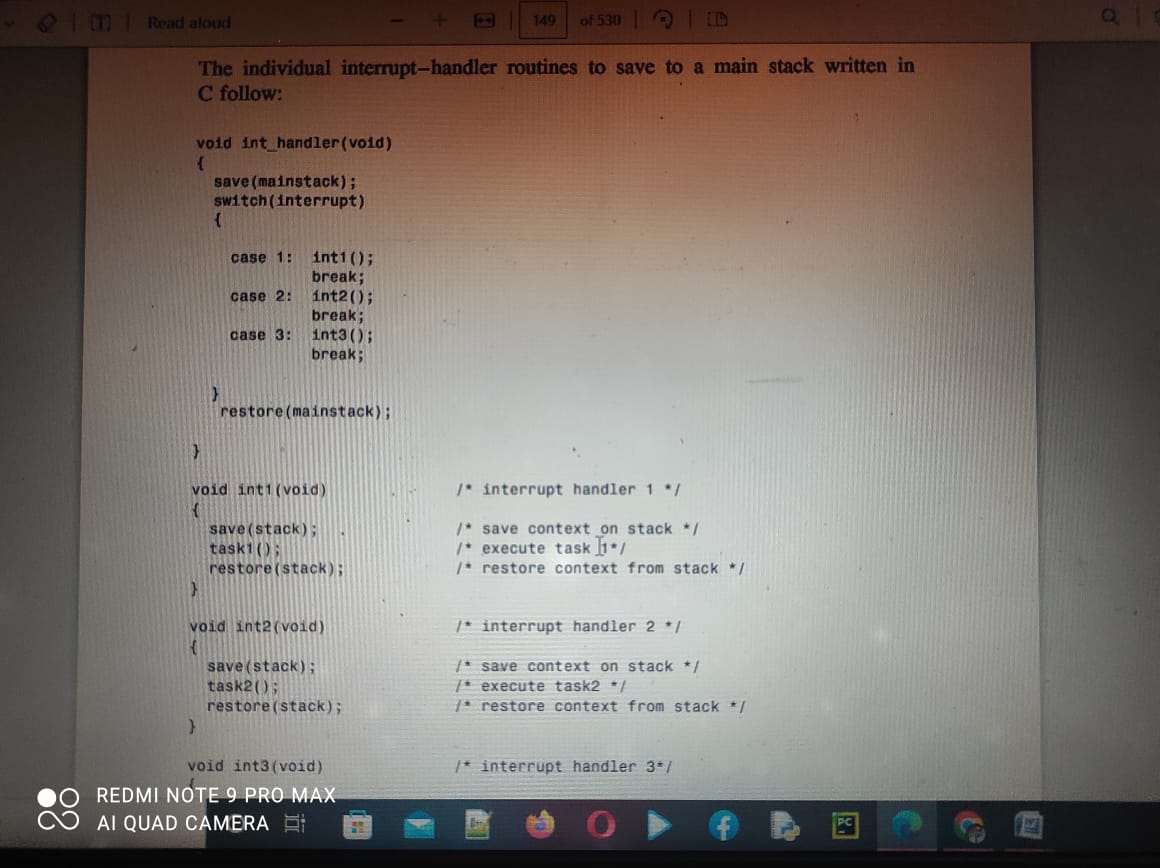
The argument context, is a pseudoargument, in that as discussed before, it is really a pointer to a data structure that contains the contents of the general register set and PC at a given instant in time. A high-order-language-callable program, written in assemble language, would be needed to save the PC, general registers, and any other context to the context data structure. Another assembly program is needed to restore the registers. Fortunately, most high-order language compilers provide these in a run-time library. Further, while stack is shown here as a global variable, in practice it is probably better to make it an argument of the save and restore routines so that multiple run-time stacks can be maintained. Throughout this chapter, however, the representation of save and restore have been kept simple for clarity of discussion. Next consider the restore routine, written in 2-address code, which restores context in precisely the reverse way as the save routine using an index register.

restore(context): ;context is a pseudo-argument

DPI ; disable interrupts

LOAD R0,&stack





save(stack); /\* save context on stack \*/

task3(); /\* execute task 3 \*/

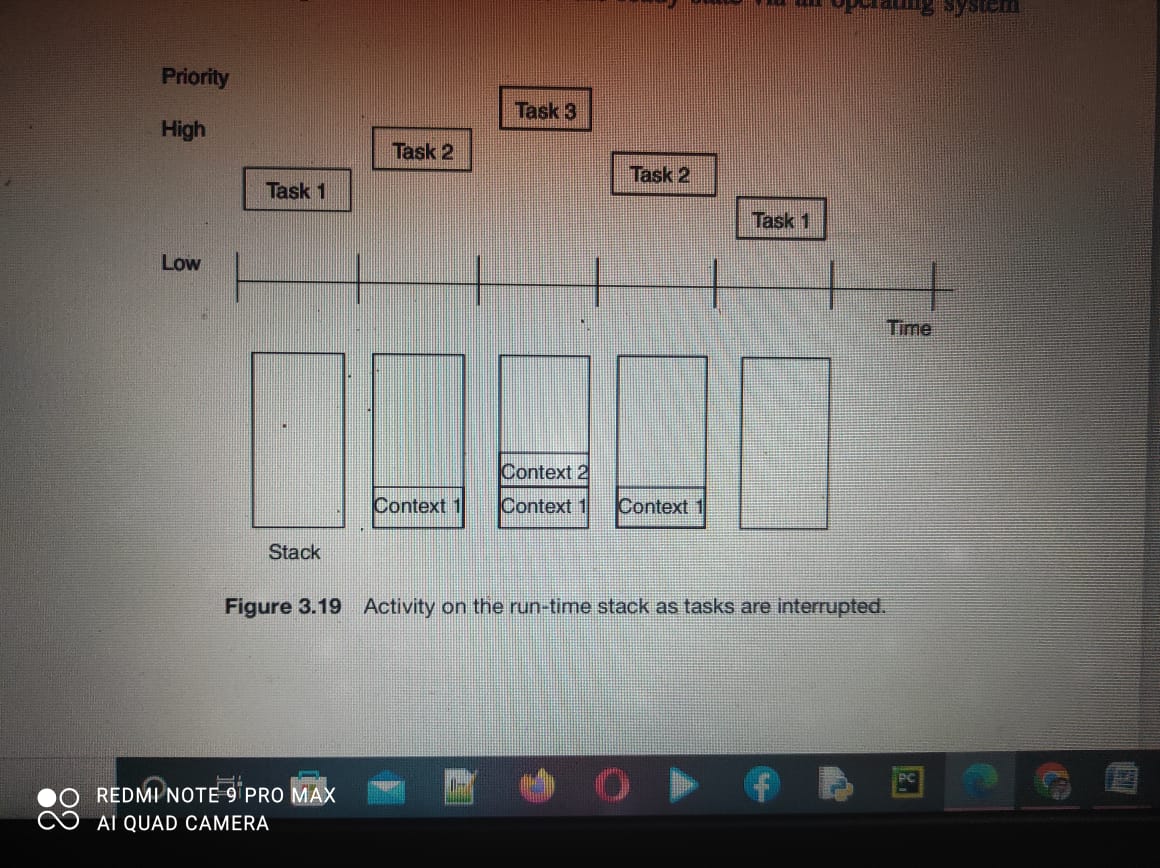
restore(stack); /\* restore context from stack \*/

}

For example, suppose three processes, task1, task2, and task3, are running in an interrupt-only system where a signal interrupt based on three prioritized interrupts is generated. Suppose task1 is running when it is interrupted by task2. Later task2 is interrupted by task3. The run-time stack evolves as in Figure 3.19. Certain machine architectures allow block save and block restore instructions to store and load n general registers in n consecutive memory locations. These instructions greatly simplify the implementation of the save and restore routines. Be aware that such macroinstructions may be designed to be interruptible (to reduce context switch time), so that if interrupts have not already been disabled, they should be.

**Task-Control Block Model:**

If the task-control block model is used, then a list of task-control blocks is kept. This list can be either fixed or dynamic. In the fixed case, n task-control blocks are allocated at system generation time, all in the dormant state. As tasks are created, the task-control block enters the ready state. Prioritization or time slicing will then move the task to the execute state. If a task is to be deleted, its task-control block is simply placed in the dormant state. In the case of a fixed number of task-control blocks, no real-time memory management is necessary. In the dynamic case, task-control blocks are added to a linked list or some other dynamic data structure as tasks are created. Again, the tasks are in the suspended state upon creation and enter the ready state via an operating system.



call or event. The tasks enter the execute state owing to priority or time slicing. When a task is deleted, its task-control block is removed from the linked list, and its heap memory allocation is returned to the unoccupied or available status. In this scheme, real-time memory management consists of managing the heap needed to supply the task-control blocks; however, other data structures such as a list or sequence can be used.