5.2 RUN-TIME STORAGE MANAGEMENT

Run-Time Storage Management is the process of managing the memory required by a program while the program is executing.

When a program runs, memory is required for:

- Variables
- Constants
- Arrays
- Functions
- Parameters
- Temporary values
- Dynamic objects

The compiler and run-time system decide where these items are stored, when memory is allocated, and when it is released.

1. Run-Time Memory Organization

The memory used by a running program is generally divided into four main areas:

Program memory layout — Stack and Heap | by Nat Retsel | Medium

medium.com

1. Code Area
2. Static / Data Area
3. Stack
4. Heap

2. Code Area

The Code Area contains the machine instructions of the program.

Example:

MOV R1, 10

ADD R1, R2

STORE R1, x

These instructions remain available while the program executes.

Characteristics:

- Contains executable instructions.
- Usually remains fixed during execution.
- May be shared or protected by the operating system.

3. Static / Data Area

The Static / Data Area stores data whose lifetime is generally the entire execution of the program.

Examples:

- Global variables.
- Static variables.
- Constants or initialized data.

Example:

```
int count = 10;


void main()
{
   static int x = 20;
}
```

Here, count and x have storage that persists throughout program execution.

4. Stack

The Stack is used to manage function calls and local data.

Whenever a function is called, a stack frame or activation record is created.

It may contain:

- Parameters.
- Local variables.
- Return address.
- Temporary values.
- Saved registers.
- Control information.

Example:

```
void fun(int a)
{
   int x;
}
```

When fun() is called, a stack frame is created for it.

5. Heap

The Heap is used for dynamic memory allocation.

Memory is allocated when it is required during program execution.

Examples:

- Dynamic objects.
- Dynamically created arrays.
- Linked lists.
- Trees.
- Other dynamic data structures.

Example in C:

```
int *p;
p = (int*)malloc(sizeof(int));
```

Memory is dynamically allocated from the heap.

It can later be released using:

```
free(p);
```

In languages such as Java, dynamically created objects are generally allocated in the heap and reclaimed by garbage collection.

6. Activation Record

An Activation Record is a block of memory associated with a particular execution of a procedure or function.

It is also called a Stack Frame.

Important Components:

1. Parameters: Values passed to the function.

2. Return Address: Address of the instruction to which control should return after the function finishes.
3. Control Link: Points to the activation record of the caller.
4. Access Link: Used to access variables in surrounding scopes, especially in nested procedures.
5. Local Variables: Variables declared inside the function.
6. Temporary Values: Temporary results generated during computation.

7. Storage Allocation Strategies

There are three major storage allocation strategies.

1. Static Allocation

Memory is allocated before program execution.

Working:

Program Start

Memory Allocated

Program Executes

Program Ends

Memory Released

Suitable for:

- Global variables.
- Static variables.
- Data with fixed lifetime.

2. Stack Allocation

Memory is allocated and released according to function calls.

Working:

Function Call

Stack Frame Created

Function Executes

Function Returns

Stack Frame Removed

Suitable for:

- Local variables.
- Parameters.
- Temporary values.

3. Heap Allocation

Memory is allocated dynamically during program execution.

Working:

Request Memory

Heap Allocation

Use Object / Data

Release / Garbage Collection

Suitable for:

- Dynamic data structures.
- Objects.
- Data whose size or lifetime is not known at compile time.

8. Static vs Stack vs Heap Allocation

Feature	**Static**	**Stack**	**Heap**
Allocation time	Before execution	Function call	During execution
Lifetime	Usually entire program	Function execution	Dynamic
Used for	Global/static data	Local variables	Dynamic objects
Management	Compiler/runtime	Stack mechanism	Programmer/GC
Flexibility	Low	Medium	High

9. Storage Allocation for Procedures

Consider the following program:

```
main()
{
   int a;
```

```
    fun();
}

fun()
{
    int b;
}
```

Execution:

main() called

main stack frame created

fun() called

fun stack frame created

fun() returns

fun stack frame removed

main() continues

main() returns

When fun() is called, its stack frame is created. After the function returns, its stack frame is removed, and execution continues in main().

10. Dynamic Storage Management

Dynamic Storage Management deals with memory that is allocated and released during program execution.

There are two common approaches:

1. Manual Memory Management

The programmer explicitly allocates and releases memory.

Example in C:

```
p = malloc(sizeof(int));

free(p);
```

2. Automatic Memory Management

The runtime system automatically identifies memory that is no longer required and releases it.

This is commonly known as Garbage Collection.

11. Garbage Collection

Garbage Collection (GC) is an automatic memory-management technique.

It identifies objects that are no longer reachable or required and reclaims their memory.

Basic Process:

Objects in Heap

Find Reachable Objects

Identify Unused Objects

Reclaim Unused Memory

Memory Becomes Available

Example:

- p → Object A
- Object B → No reference

Object B cannot be reached by the program, so its memory can be reclaimed by the garbage collector.

12. Importance of Run-Time Storage Management

Run-Time Storage Management is important because it:

1. Uses memory efficiently.
2. Supports function calls.
3. Manages local variables and parameters.
4. Supports dynamic data structures.
5. Prevents unnecessary memory usage.
6. Supports recursion.
7. Manages object lifetime.
8. Helps avoid memory-related errors.