RUN-TIME ENVIRONMENT (Part-I)

STORAGE ORGANISATION

CONTENTS :

- **RUN-TIME ENVIRONMENT**
- **LOGICAL ADDRESS SPACE**
- **STORAGE ORGANISATION**
  - STATIC STORAGE ALLOCATION
  - DYNAMIC STORAGE ALLOCATION

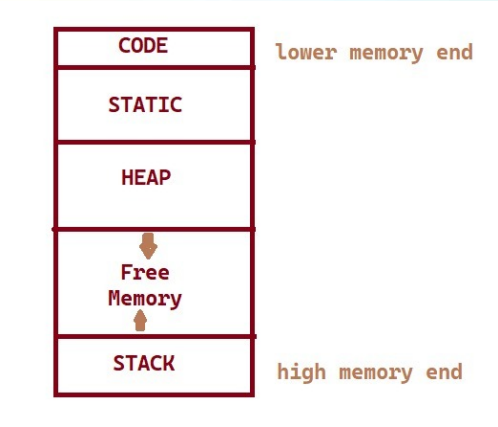
RUN-TIME ENVIRONMENT

- To perform those implementations, the compiler creates and manages a **run-time environment** in which target programs are executed.
- **This environment deals with a variety of issues such as –**
  - Layout and storage allocation for objects of the source program
  - The mechanisms used by the target program to access variables
  - Linkages between procedures
  - Mechanisms for parameter passing
  - Interface to the operating system, i/o devices, and other programs

STORAGE ORGANISATION

- The target program runs in its own logical address space in which each program value has a location
- The management and organization of this logical address space is shared between the compiler, operating system, and target machine.
- The logical address is mapped into the physical address by the operating system.

STORAGE ORGANISATION



STORAGE ORGANISATION

- We assume the run-time storage comes in blocks of bytes.
- 1 byte = 8 bits
- Four bytes form a machine word
- Multibyte objects are stored in consecutive bytes and give the address of the first byte.
- An elementary datatype, such as char, int, etc can be stored in an integral number of bytes.
- Storage for an aggregate type, such as an array or structure, must be large enough to hold all its components.

LOGICAL ADDRESS SPACE

- The size of generated target code is fixed at compile time, so the compiler can place the executable target code in the **code** region (*statically determined*).
- The size of program objects such as global constants, and the data generated by the compiler, such as information to support garbage collection are known at compile time. So, they can be place in the **Static** region (*statically determined*).
- The **Heap** and **stack** regions are at the opposite ends of remaining address space. (*Dynamically determined*). Their size can change as the program executes.

LOGICAL ADDRESS SPACE

- The stack is used to store data structures called activation records that get generated during procedure calls
- The activation record is used to store information about the status of the machine, such as the program counter and machine registers.
- The stack grows towards lower addresses, the heap towards higher.
- Many programming languages allow the allocation and deallocation of data under program control.
- Ex: In C, functions such as malloc and free are used to obtain and give back arbitrary chunks of storage.
- The heap is used to manage this kind of long-lived data.

STATIC Vs DYNAMIC STORAGE ALLOCATION

- Storage allocation decision is static if it can be made by the compiler by only looking at the text of the program, not at what the program does when it executes.
- Decision is dynamic if it can be decided only when the program is running
- Many compilers use a combination of the two strategies for dynamic storage allocation:
  - **Stack storage**: Names local to a procedure
  - **Heap storage**: Data that may outlive the call to the procedure (*It is area of virtual memory that allows objects or other data elements to obtain storage when created and return storage when they are invalidated*)

  To support heap management, ***"garbage collection"*** enables the run-time system to detect useless data elements and reuse their storage, even if the space is not returned explicitly.