Stack & Heap

Program memory

- When you run a program (say, Microsoft Word, or Google Chrome, etc), it occupies some amount of memory on your computer.
- This is the memory taken up by that program process.
- Now, in that program, there will typically be code that allocates memory in different ways.
- This memory comes from either the stack or the heap… think of these as different areas of memory in the computer.
- We will talk more on this soon.

- The processes that are running on your computer take up memory from the RAM available on your computer. Eg, your computer may have 8GB or 16GB or 32 GB RAM (or more).

- Your programs can take up more memory than the RAM that is available.
- This can be done because of a concept called *virtual memory*.
- In this, data from RAM is written to the disk by the operating system.
    - Note: This is transparent to you as the developer of the program, or the user of the computer. It is handled by the operating system.

Stack memory

- Stack is a part of the computer memory from where objects created inside a function are stored.
- For example, if a function declares, say, two variables of type integer and one variables of type double, these variables are allocated in the stack region of memory.
- When the function exits, the variables on the function stack are released because the stack memory is deleted.
  - What does it mean to say this memory is deleted?
  - It just means that area of memory is now available again.

```
void FuncA()
{
        int a, b;
}
```

← These are allocated in stack memory… you typically say, these are allocated on the function stack.

Stack memory

- Objects declared on the stack do NOT need to be deleted, when the function exits, these are automatically deleted because that memory is released (marked as available again).

- Program has a limited amount of stack space. This can be increased or decreased using compiler options, but still is not as big as the memory available in *heap*.

Heap memory

- Heap is a region of computer memory where you can allocate memory for your objects.
- This is typically much bigger memory than stack memory.

- This is allocated using calls such as new or malloc, etc (depending on the language).

- Heap memory usage can cause fragmentation of memory.
- This is because a program will ask for different sized chunks of memory at different times, and may release these at different times as well. What is released is then available again. This results in fragmentation, as expected.
- Stack memory is allocated in a contiguous fashion, mainly because what is allocated on the stack needs to be known at compile time.