Starter

1. What is the stored program concept?
   - A program's code/instructions must be resident in main memory for it to be executed. These instructions are fetched and executed one at a time by the processor.
2. In this context, what is a process?
   - An **instance of a program being executed**.

The Call Stack

4.1.1.15 Role of stack frames in subroutine calls

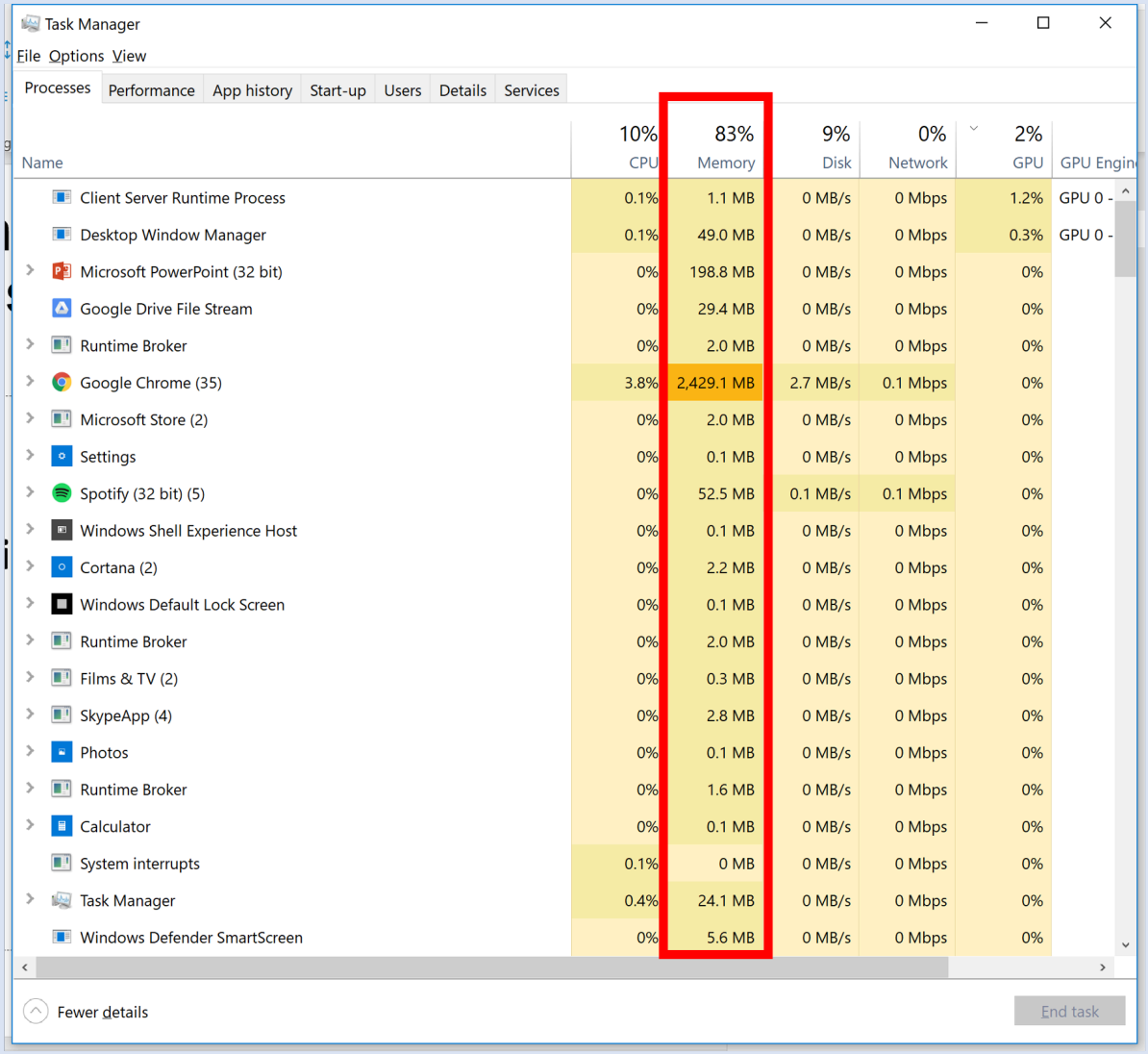
Content	Additional information
Be able to explain how a stack frame is used with subroutine calls to store: • return addresses • parameters • local variables.	

Processes

A **process** is an **instance of a program being executed**.

In a modern OS, many processes exist at once.

Each process is allocated its own portion of main memory by the operating system.



Main memory management

The operating system will keep track of which parts of the main memory are being used by which process.

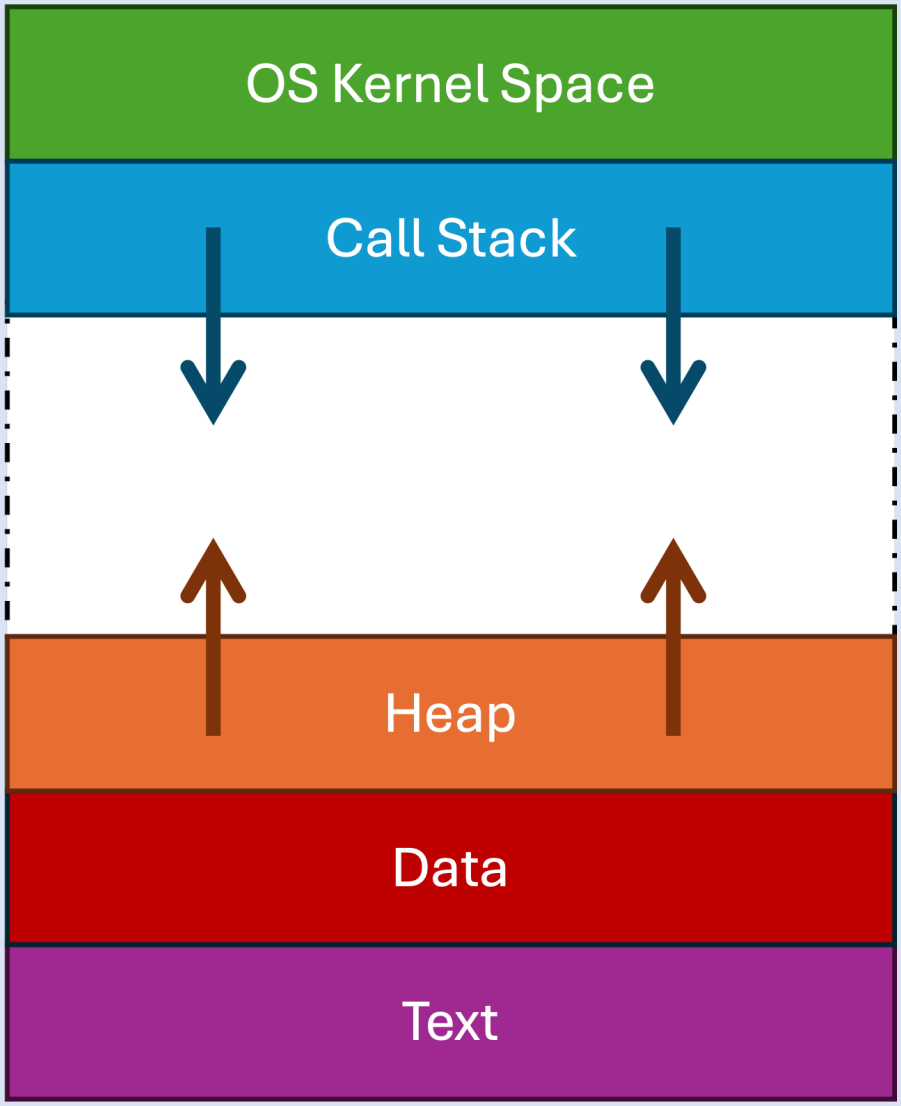
Each process may use groups of locations that are not **contiguous**.

The OS hides this complexity – the process and the user do not need to know where in the main memory the code and data can be found.

A	001	00001010
	001	10010101
	001	00100101
	001	11101010
...	...	...
B	0A6	010101111
	0A7	101010111
	0A8	110111011
	0A9	111111111
...	...	...
A	0F9	101010111
	0FA	00001010
	0FB	10011011
...	...	...
C	1A2	11100000
	1A3	00000000
	1A4	00100101
	1A5	11010110
...	...	...

Main memory management

- This model shows the section of main memory each process sees.
- Global variables are stored in **data**.
- Statically allocated variables, such as subroutine **local variables**, **parameter values** and **return addresses** are stored in the **call stack**.
- Program code is stored in **text**.
- Dynamically allocated **objects**, such as strings, arrays and lists, are stored in the **heap**.

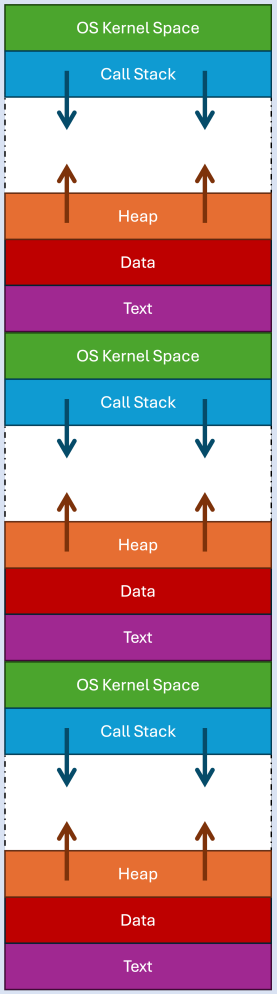


Main memory management

Process A

Process B

Process C



The call stack

- Stores all information about **active** subroutine calls
- Two main purposes
  - To keep track of where in the program to return to when a subroutine call terminates
  - To store the values of local variables and parameters specific to that subroutine call
- This information is stored in containers called **stack frames**

Stack frames

- A **stack frame** is a portion of memory that stores data related to an **active subroutine** (a subroutine that has been called, but has not finished executing)

Stack Frame

1. Parameter values
2. Local variables
3. Return memory address

Call stack example

C#

```
1   class Program {
2     static void Add1(int toAdd) {
3       int result = 0;
4       Console.WriteLine("Adding");
5       result = toAdd + 1;
6       Console.WriteLine("Value is " + result);
7     }
8     static void First(int firstInput) {
9       Console.WriteLine("Adding 1 to " + firstInput);
10      Add1(firstInput);
11    }
12    static void Main(string[] args) {
13      Console.WriteLine("Main");
14      First(10);
15    }
16  }
```

```
Main
Adding 1 to 10
Adding
Value is 11
```

What does this program output?

Call stack example

C#

```
1   class Program {
2     static void Add1(int toAdd) {
3       int result = 0;
4       Console.WriteLine("Adding");
5       result = toAdd + 1;
6       Console.WriteLine("Value is " + result);
7     }
8     static void First(int firstInput) {
9       Console.WriteLine("Adding 1 to " + firstInput);
10      Add1(firstInput);
11    }
12    static void Main(string[] args) {
13      Console.WriteLine("Main");
14      First(10);
15    }
16  }
```

```
Main
Adding 1 to 10
Adding
Value is 11
```

Call Stack

How the call stack changes in size

- The call stack stores data about all active subroutine calls
- It is made up of stack frames (one for each subroutine call)
- Each time a subroutine is called, a new stack frame is created, and the call stack **increases in size**
- Each time a subroutine terminates, the associated stack frame is removed, and the call stack **decreases in size**

Practice

```
1   class Program {
2     static void Add1(int toAdd) {
3         int result = 0;
4         Console.WriteLine("Adding");
5         result = toAdd + 1;
6         Console.WriteLine(result);
7     }
8
9     static void Subtract(int a, int b) {
10        int result = 0;
11        result = a - b;
12        Console.WriteLine(result);
13    }
14
15    static void Main(string[] args) {
16        int myNum1 = 5;
17        int myNum2 = 10;
18
19        Add1(myNum2);
20        Subtract(myNum1, myNum2);
21    }
22  }
```

C#

What does the call stack look like

1. just after line 18?
2. just after line 6?
3. just after line 20?

Ada Quiz: L117 – Call Stack

Lists

Objects

- There are two forms of memory allocation
- **Static**
  - Used for values of a fixed size that we can allocated space for before running the code
  - `int` and `float` are both stored statically, because they both always require exactly 4 bytes of memory
    - They are **primitive data types**
- **Dynamic**
  - Used for values for which we do not know how much memory needs to be allocated
  - Arrays, strings, lists are examples
  - These are called **objects**
  - To store an object, with no fixed size, we need to use a fixed-size **object reference** to refer to it

Lists in C# Demo

Lists exercise

You have **10 minutes**.

1. Declare and create a new list of strings in `Main`
2. Populate it with four items: `"Eggs"`, `"Milk"`, `"Flour"`, `"Butter"`
3. Use a `for` loop to print out each item in the list on a new line
4. Create a procedure called `PrintList`, which takes a string list reference parameter:
   ```
   static void PrintList(List<string> shoppingList)
   ```
5. The procedure should use a `for` loop to print out the contents of the list, in the following format
   ```
   [Eggs, Milk, Flour, Butter]
   ```
6. Test your program using different lists

Worksheet

W109 – C# Lists