

Stack and Heap memory

Coventry University

Overview

1 Stack and Heap

- Stack
- Heap

- Memory model used so far is a simplification.
- Actually two places in memory that variables can go.
 - The stack and the heap.
- Both are just regions of the same physical memory.
 - Are managed differently.

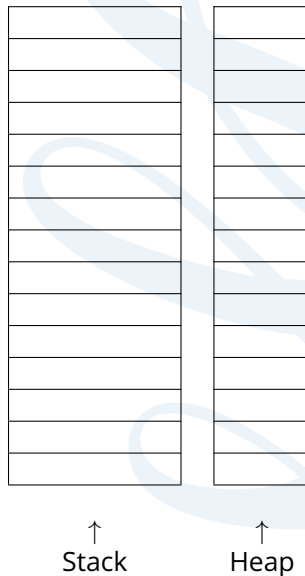


- When program is run, block of memory is allocated.
 - Called the stack.
- Each program has it's own stack.
 - Each instance.
- As variables created and functions called they are put on the stack.
- When variables are destroyed/functions complete they are removed from the stack.
- Has limited size.
 - Recursive functions can fill the stack if not careful.

```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

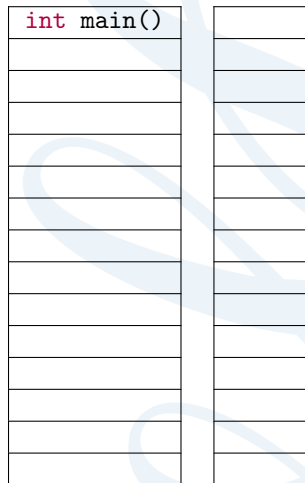
int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```



```
int add( int a, int b)
{
    int result = a+b;
    return result;
}
```

```
int sub( int a, int b )
{
    int result = a-b;
    return result;
}
```

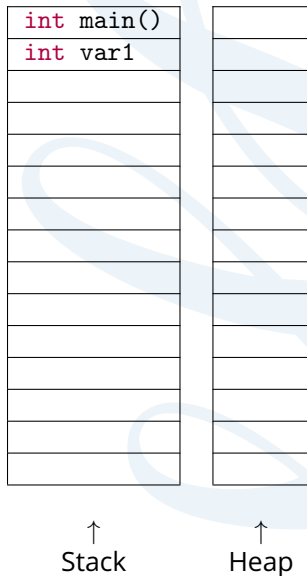
```
⇒ int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

↑
Stack↑
Heap

```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    ⇒ int var1 = 42;
      int var2 = 1;
      add(var1,var2);
      sub(var1,var2);
      return 0;
}
```

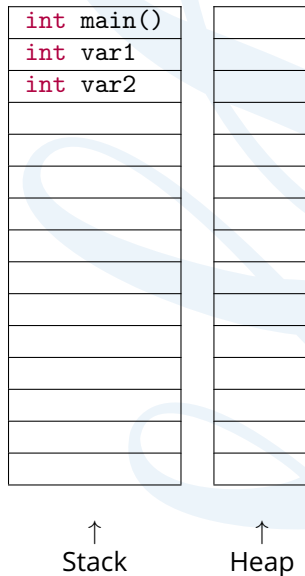


```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

⇒

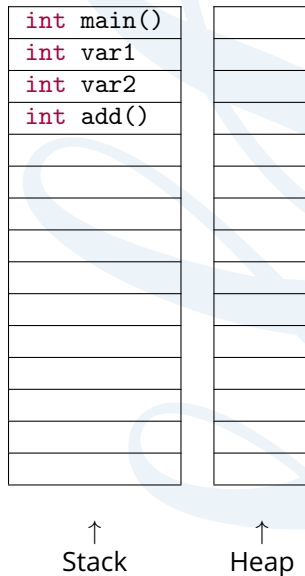



```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

⇒

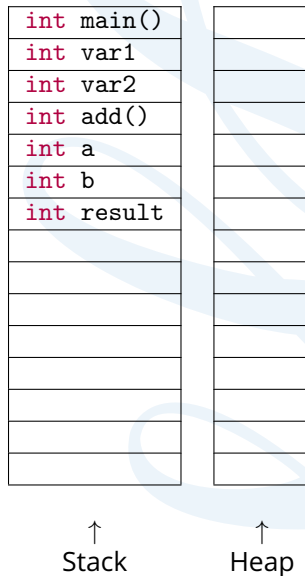


[illegible]

```
⇒ int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```



<code>int</code>	<code>main()</code>	
<code>int</code>	<code>var1</code>	
<code>int</code>	<code>var2</code>	
<code>int</code>	<code>add()</code>	
<code>int</code>	<code>a</code>	
<code>int</code>	<code>b</code>	
<code>int</code>	<code>result</code>	

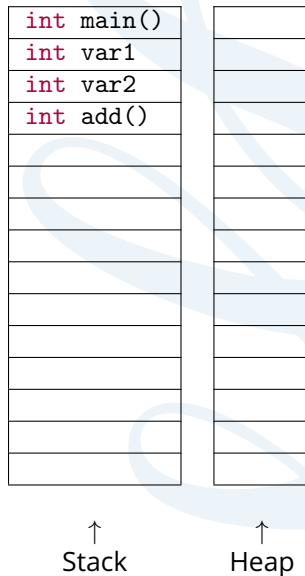
↑ ↑
Stack Heap

```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

⇒

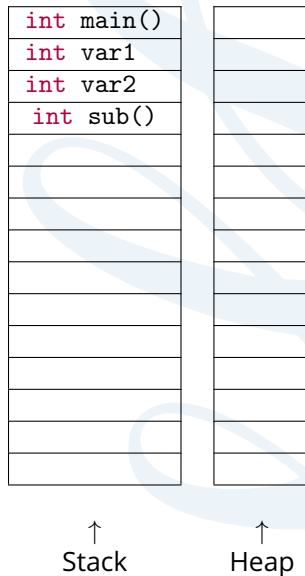


```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

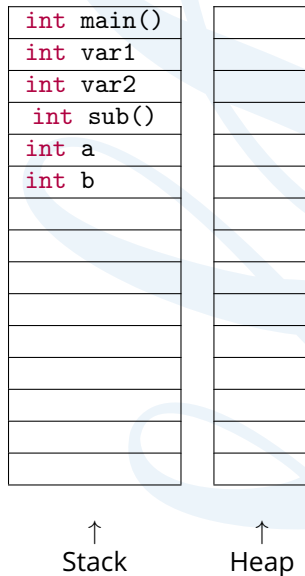
⇒



```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

⇒ int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```



int main()	
int var1	
int var2	
int sub()	
int a	
int b	
int result	

↑ ↑

Stack Heap

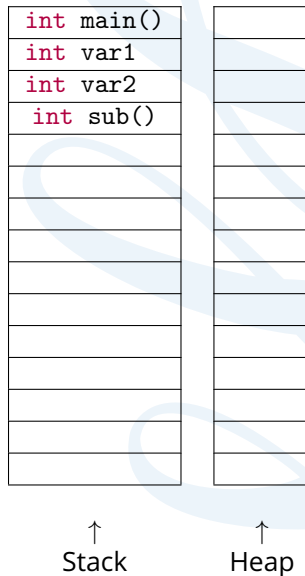
[illegible]

```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

⇒

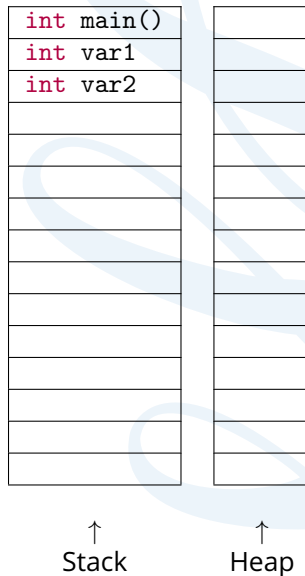


```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```

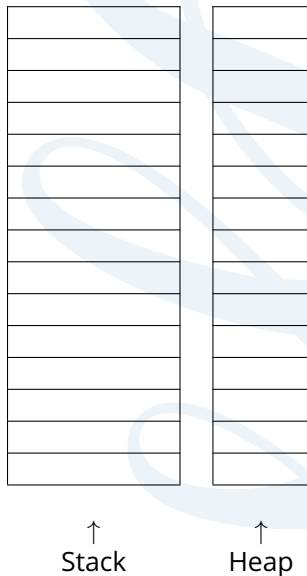
⇒



```
int add( int a, int b)
{
    int result = a+b;
    return result;
}

int sub( int a, int b )
{
    int result = a-b;
    return result;
}

int main()
{
    int var1 = 42;
    int var2 = 1;
    add(var1,var2);
    sub(var1,var2);
    return 0;
}
```



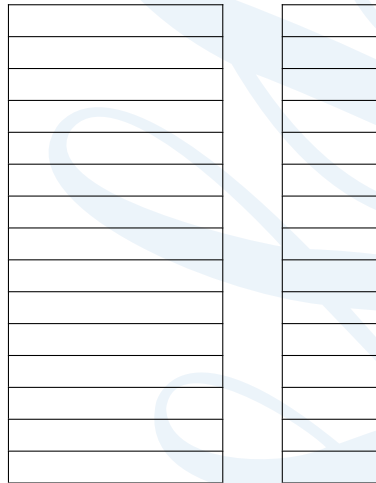


- Shared memory between all running programs.
- Very big in comparison to the stack.
- Dangerous, must remember to deallocate our memory.
 - Memory leaks.

```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    int *pointer2;
    pointer2 = new int[3];

    delete [] pointer1;
    return 0;
}
```

↑
Stack↑
Heap

[illegible]

↑
Heap

The diagram illustrates memory allocation for two different types of variables:

- Stack:** A vertical column of 16 rectangular boxes representing memory slots. The top two boxes are filled with the code `int main()` and `int variable`, respectively. Below these are 14 empty boxes. An upward-pointing arrow labeled "Stack" is positioned at the bottom left.
- Heap:** A vertical column of 16 rectangular boxes, identical in size to the stack. It contains 14 empty boxes. An upward-pointing arrow labeled "Heap" is positioned at the bottom right.

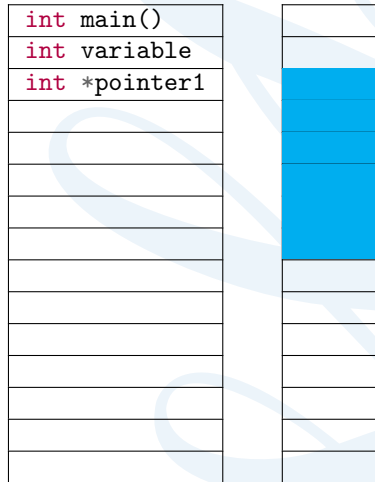
A large, light blue watermark logo is visible across the center of the image.

[illegible]

```
int main()
{
    int variable = 42;
    int *pointer1;
    ⇒ pointer1 = new int[6];

    int *pointer2;
    pointer2 = new int[3];

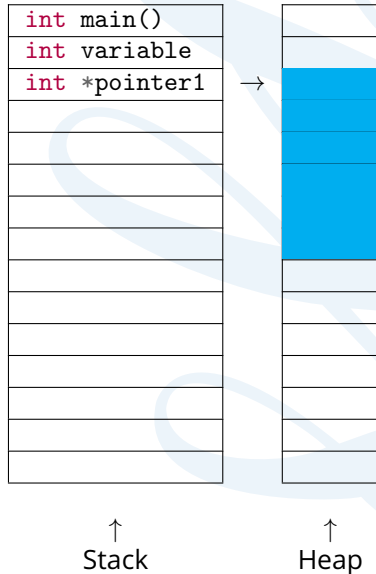
    delete [] pointer1;
    return 0;
}
```

↑
Stack↑
Heap

```
⇒ int main()
   {
     int variable = 42;
     int *pointer1;
     pointer1 = new int[6];

     int *pointer2;
     pointer2 = new int[3];

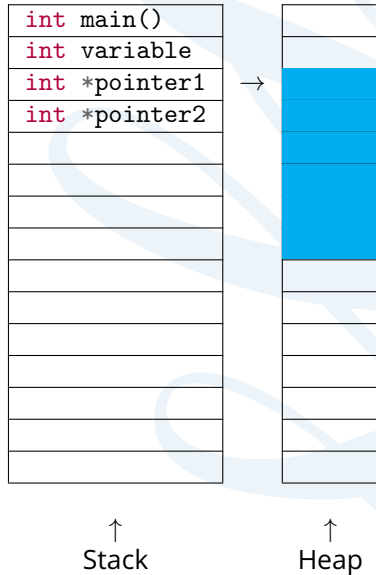
     delete [] pointer1;
     return 0;
   }
```



```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    ⇒ int *pointer2;
    pointer2 = new int[3];

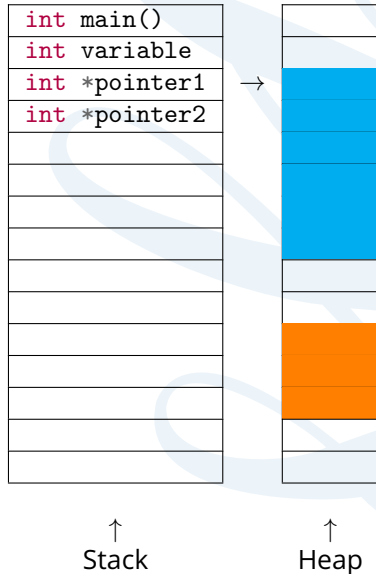
    delete [] pointer1;
    return 0;
}
```



```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    int *pointer2;
    ⇒ pointer2 = new int[3];

    delete [] pointer1;
    return 0;
}
```

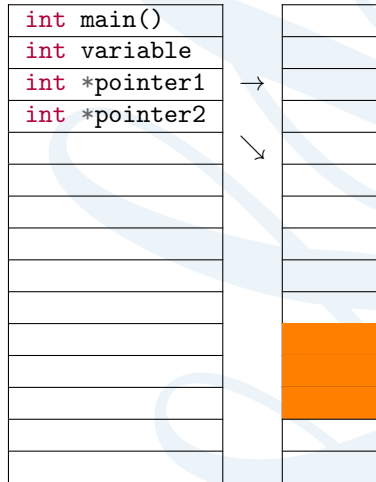


The diagram illustrates the memory stack and heap. The stack is on the left, growing downwards (indicated by an upward arrow labeled 'Stack'). It contains variables: `int main()`, `int variable`, `int *pointer1`, and `int *pointer2`. The heap is on the right, growing upwards (indicated by an upward arrow labeled 'Heap'). It is represented by a blue block. Arrows show the stack growing down and the heap growing up towards each other.

```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    int *pointer2;
    pointer2 = new int[3];

    ⇒ delete [] pointer1;
    return 0;
}
```

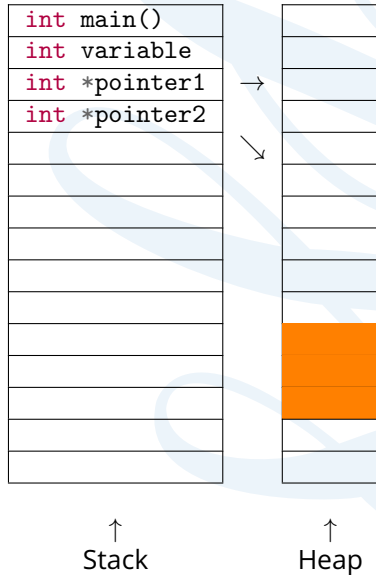
↑
Stack↑
Heap

```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    int *pointer2;
    pointer2 = new int[3];

    delete [] pointer1;
    return 0;
}
```

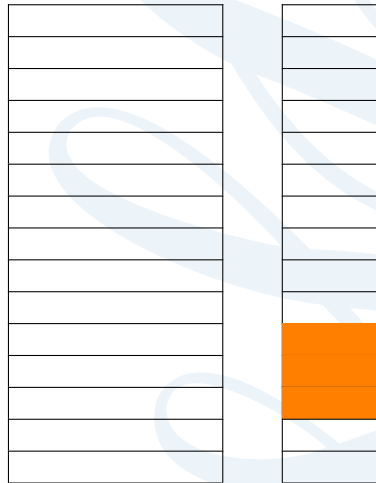
⇒




```
int main()
{
    int variable = 42;
    int *pointer1;
    pointer1 = new int[6];

    int *pointer2;
    pointer2 = new int[3];

    delete [] pointer1;
    return 0;
}
```

↑
Stack↑
Heap

Differences

Stack

- Fast - processors typically have special instructions for dealing with stacks quickly.
- Contiguous - everything in one block, easier to know where to put next variable/function.
- Small - limited size.
 - Trying too variables will fill stack and cause "stack overflow".

Heap

- Huge - relative to the stack.
- Dangerous - must remember to deallocate otherwise have memory leaks.

The End