CO650 Advanced Programming
Stack, Heap, Memory Management & Destructors

Memory Management

- The Stack
- Memory
- Destructors
- Pointers
- Auto Pointers
- Memory Error Handler

Performance
- Inline functions
- Register variables

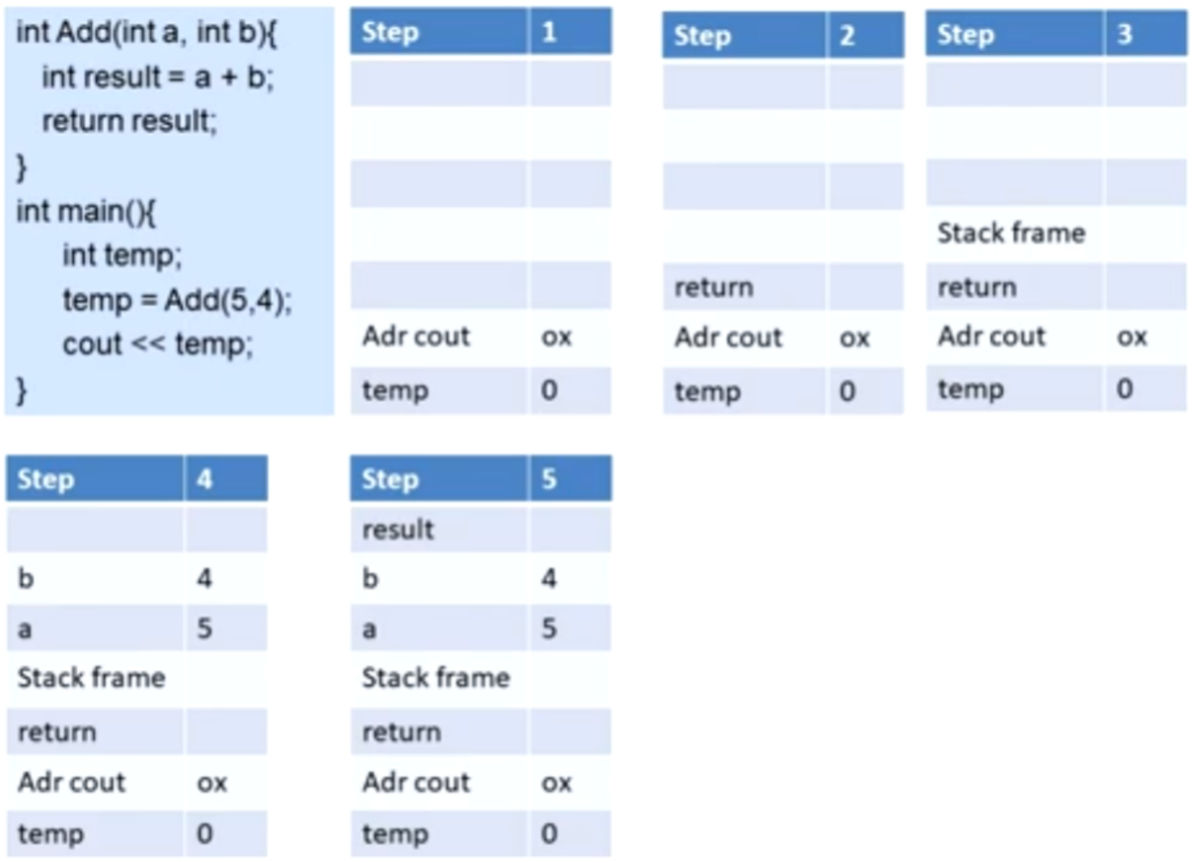
Memory

A program utilizes four types of memory:

1. The code area: compiled application
2. The global area: global variables
3. The heap: dynamically allocated objects
4. The stack: parameters and local variables \

The Stack

- A data structure that restricts access. Last In First Out (LIFO).
    a. Add a new item
    b. Look at the last item added
    c. Remove last item
- Statically declared variables are allocated memory from within the stack
- This memory is automatically freed when the variable goes out of scope



```
int Add(int a, int b){
  int result = a + b;
  return result;
}
int main(){
    int temp;
    temp = Add(5,4);
    cout << temp;
}
```

Step	1
Adr cout	ox
temp	0

Step	2
return	
Adr cout	ox
temp	0

Step	3
Stack frame	
return	
Adr cout	ox
temp	0

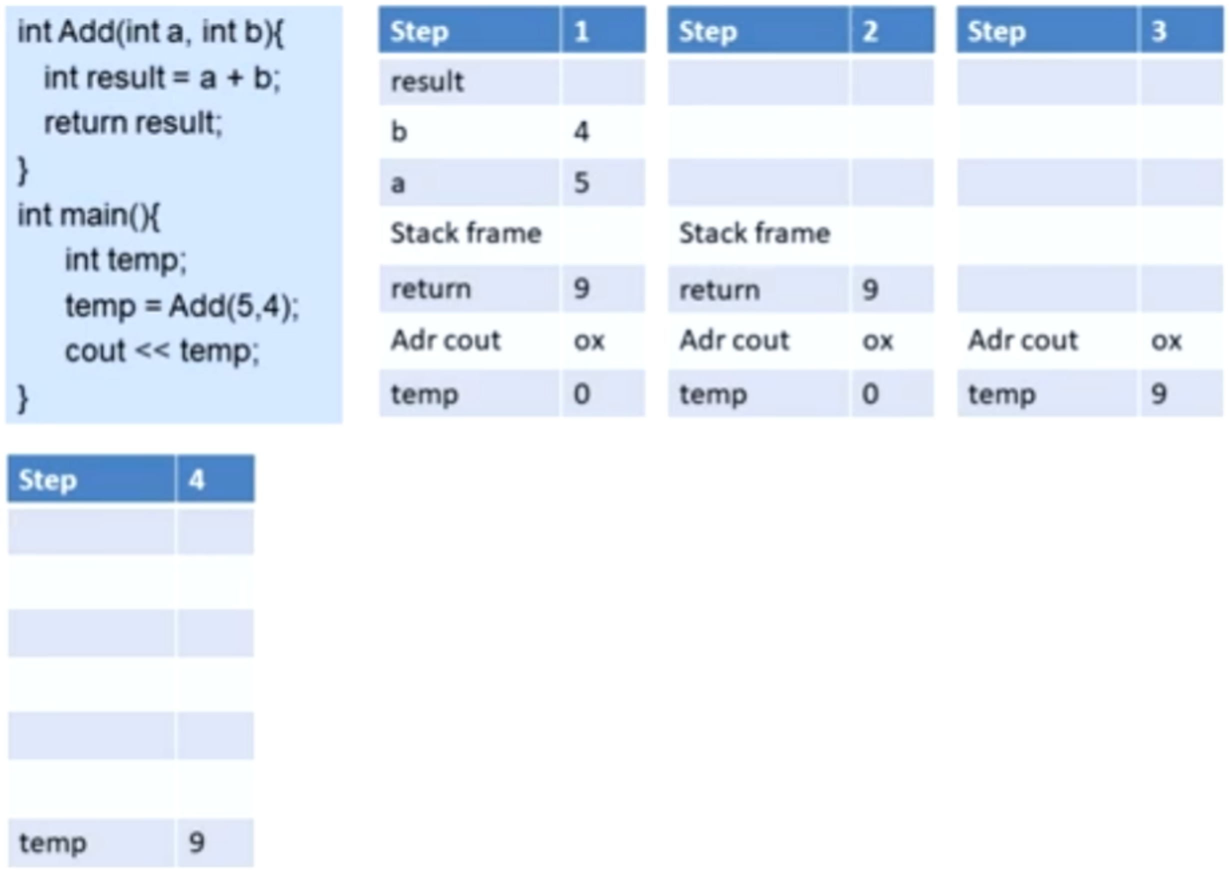
Step	4
b	4
a	5
Stack frame	
return	
Adr cout	ox
temp	0

Step	5
result	
b	4
a	5
Stack frame	
return	
Adr cout	ox
temp	0

Example of adding items to the stack in order.

When the function terminates, the following steps happen

1. The function's return value is copied into the placeholder that was put on the stack for this purpose.
2. Everything after the stack frame pointer is popped off. This destroys all local variables and arguments.
3. The return value is popped off the stack and is assigned as the value of the function. If the value of the function isn't assigned to anything, no assignment takes place, and the value is lost.
4. The address of the next instruction to execute is popped off the stack, and the CPU resumes execution at that instruction.



```
int Add(int a, int b){
   int result = a + b;
   return result;
}
int main(){
     int temp;
     temp = Add(5,4);
     cout << temp;
}
```

Step	1
result	
b	4
a	5
Stack frame	
return	9
Adr cout	ox
temp	0

Step	2
Stack frame	
return	9
Adr cout	ox
temp	0

Step	3
Adr cout	ox
temp	9

Step	4
temp	9

Process of removing stacks from previous example.

Stack Overflow

- The stack has a limited size, and consequently can only hold a limited amount of information. If the program tries to put too much information on the stack, stack overflow will result. Stack overflow happens when all the memory in the stack has been allocated - in that case, further allocations begin overflowing into other sections of memory.
- Stack overflow is generally the result of allocating too many variables on the stack, and/or making too many nested function calls (where function A calls function B, B calls C, C calls D, etc…) Overflowing the stack generally causes the program to crash

Advantages of Stack

1. Stack automatically manages the memory and garbage collection. Memory allocated on the stack stays in scope as long as it is on the stack. It is destroyed when it is popped off the stack
2. All memory allocated on the stack is known at compile time. Consequently, this memory can be accessed directly through a variable.

3. Because the stack is relatively small, it is generally not a good idea to do anything that eats up lots of stack space. This includes allocating large arrays, structures, and classes, as well as heavy recursion.

The Heap

- Dynamically variables are stored within the heap.
- Objects allocated on the heap can persist beyond the function in which they were instantiated. As long as a pointer is maintained.
- The heap is much larger than the stack and is typically used for storing objects and data structures.
- The dynamic memory is accessed through pointers or references.

.

The “New” Operator

The new operator allocates memory on the heap and returns the address of a chunk of memory.

```
type *pointerName = new type;
int *pAge = new int;
```

The allocated memory can be initialized at the same time that it is allocated using a constructor.

```
int *pAge = new int(21)
```