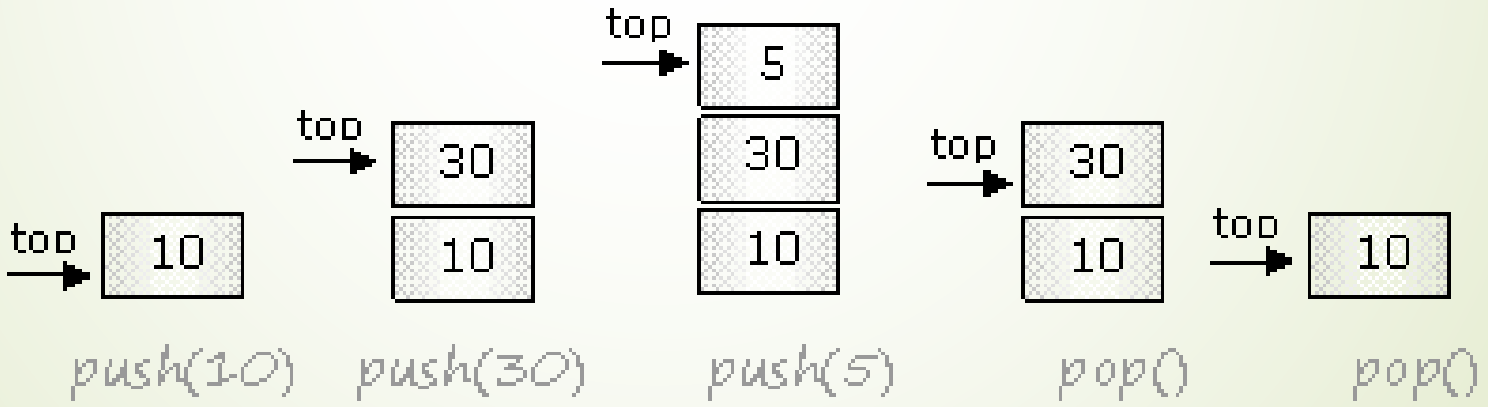
STACK

- If we consider a stack of plates in a cafeteria, we can perform limited operations on the stack of plates like
  - adding a new plate at the top.
  - removing a plate from the top.
  - Here adding a new plate on the top of plates stack is called **push operation.**
  - Removing a plate from the top of the stack is called **pop operation.**

  - Limitation with the stack is that, it works on LIFO principle that is last added plate is first removed (Last In First Out).

STACK

- In stack new element can be added only at the top and removed only from the top that is works on LIFO (Last in first out) principle.
- Adding a new element at the top of the stack is called push and removing an element from the top of the stack is called pop.
- A pointer (arrow) is at the top which keeps the track of the top means it shows that the top of the stack is here.



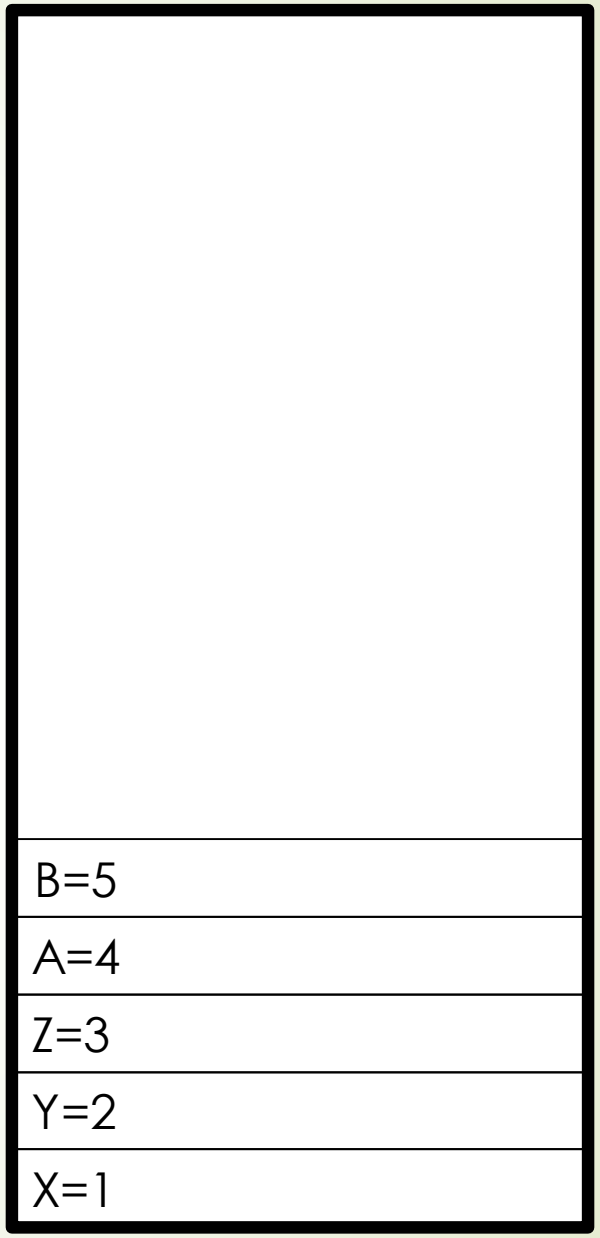
X=1

Y=2

Z=3

A-4

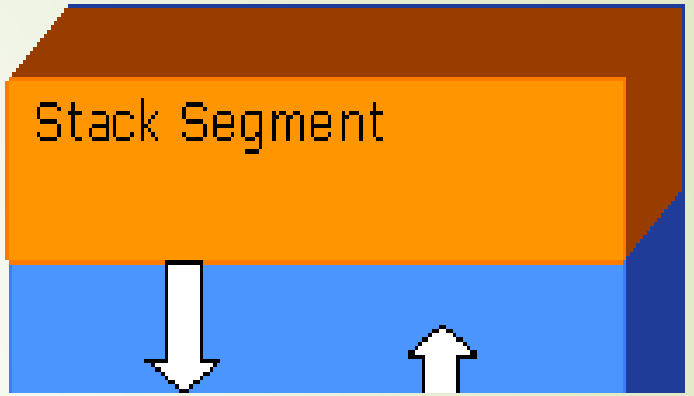
B-5



STACK

STACK

- Stack segment or call stack is located in the higher part of memory and stack grows downwards as the new segments adds.
- You can see a portion of memory named heap. Heap grows upwards whenever it fills in with data.
- If both heap and stack meets at a point then there would be no free memory and the state is called **stack overflow.**
- We have already seen that when main() is called, a block is assigned in memory for main as we did in class. That block of memory is basically a stack. (The memory has already fixed a big block for functions to store variables and parameters and that portion of memory is stack)



STACK

- Every Function on stack is assigned a different block to store their local variables and parameters (if any). And that block is called **Stack frame.**
- Local variables of functions, parameters and addresses of calling functions are pushed on the stack segment.
- Stack grows as the number of functions called is increased.

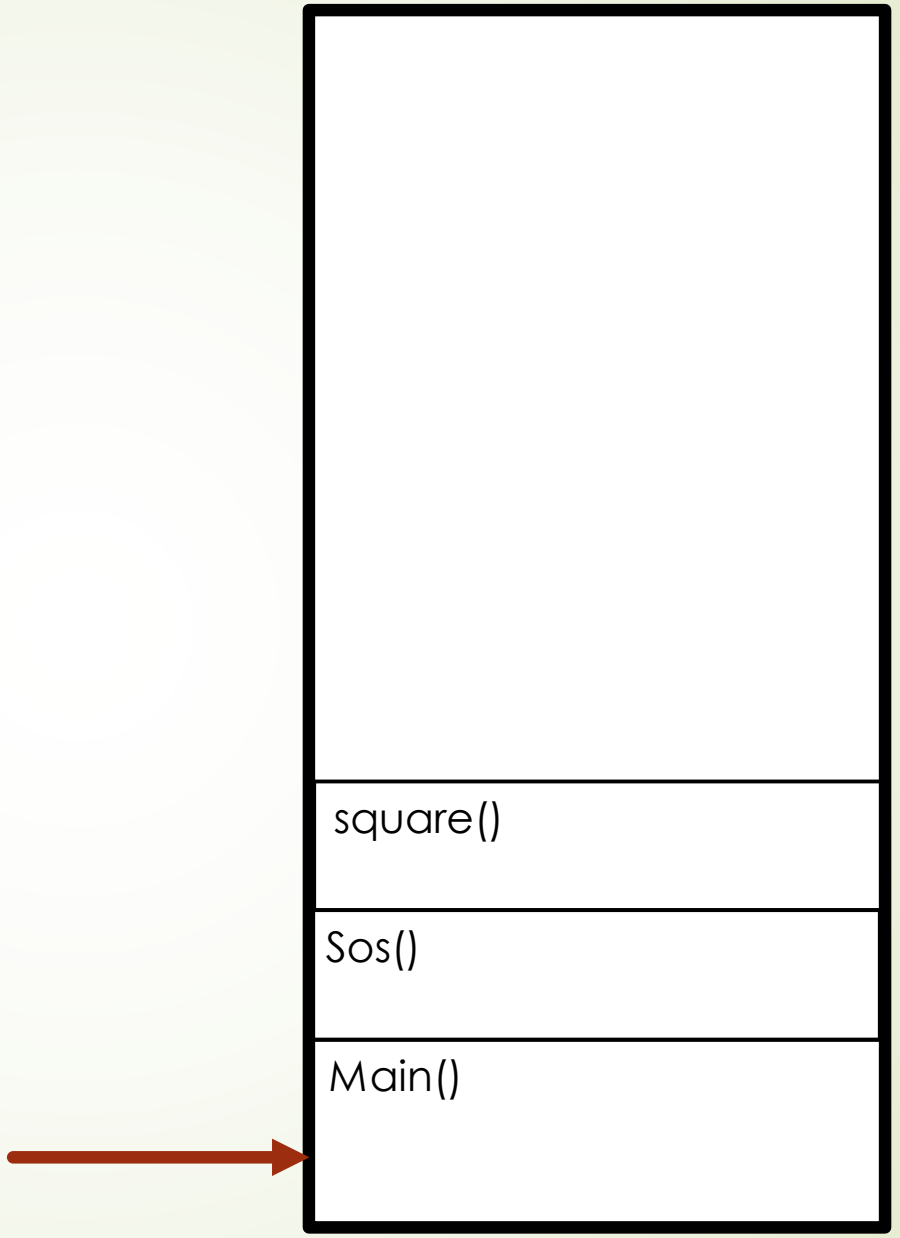
What happens when Function is called

- All the local variables and parameters are local to function.
- **Steps are Follows**
- return address is saved means address of the instruction next to the function call is pushed onto the stack e.g after function call in main which is the next statement.
- All function arguments are placed on the stack.
- The operations inside the function are executed.

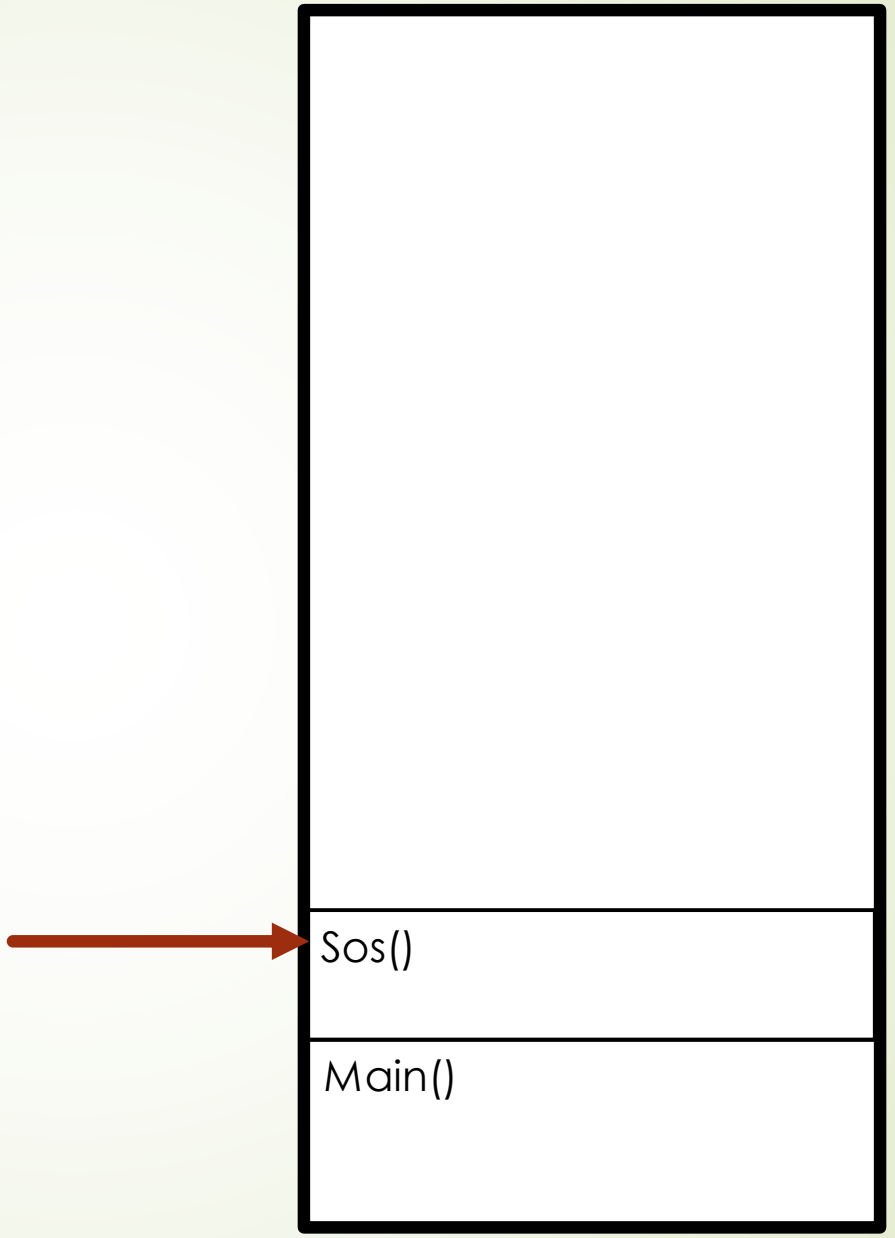
What happens when a function is terminated?

- Everything after the stack from stack frame is popped off. This destroys all local variables and arguments.
- The return value is popped off the stack and is assigned as the value of the function.
- The address of the next statement to execute is popped off the stack, and the CPU resumes execution from that statement.

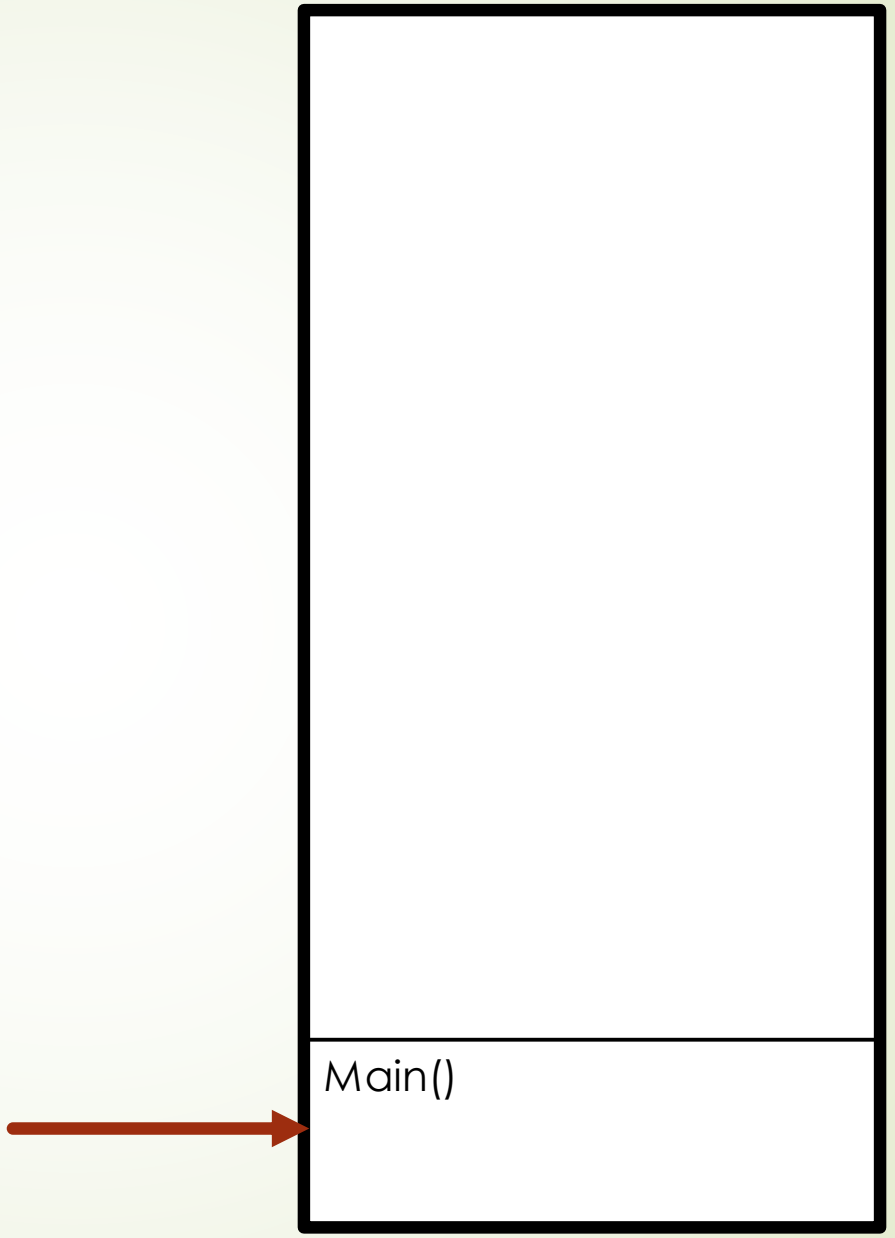
```
#include <iostream>
Int square(int x)
      { return x*x; }
Int sos(int a, int b)
{ int z=square(a+b);
Return z; }
Int main()
{
Int  total, x=4, y =2;
Total = sos(x,y);
Cout<< total;
}
```



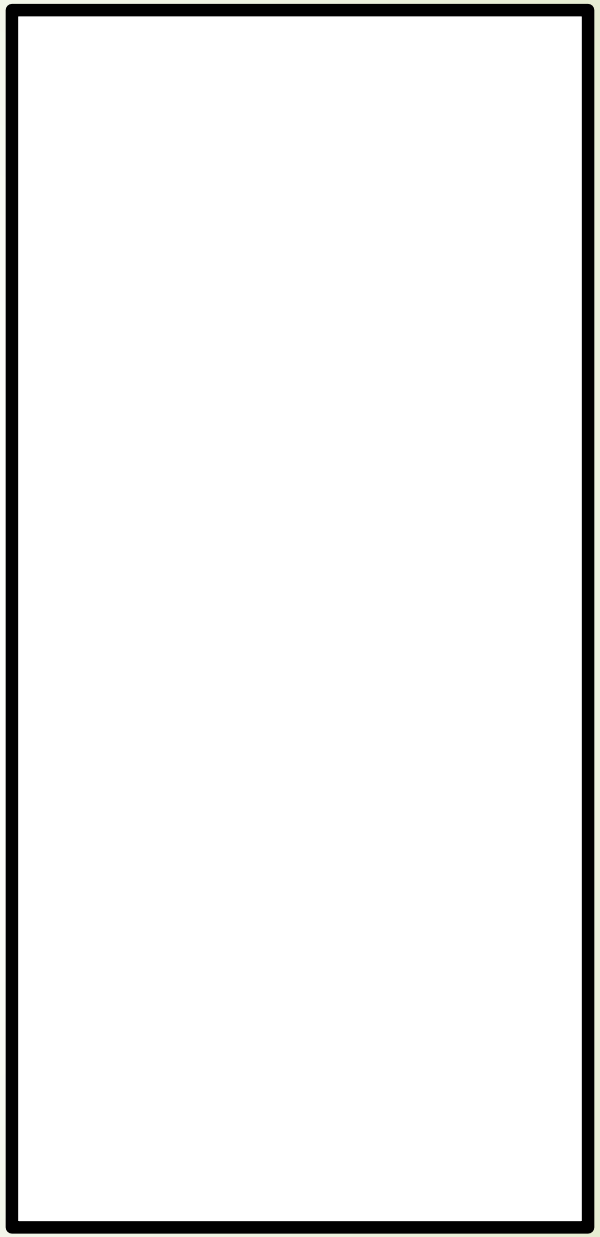
```
#include <iostream>
Int square(int x)
      { return x*x; }
Int sos(int a, int b)
{ intz=square(a+b);
Return z; }
Int main()
{
Int  total, x=4, y =2;
Total = sos(x,y);
Cout<< total;
}
```



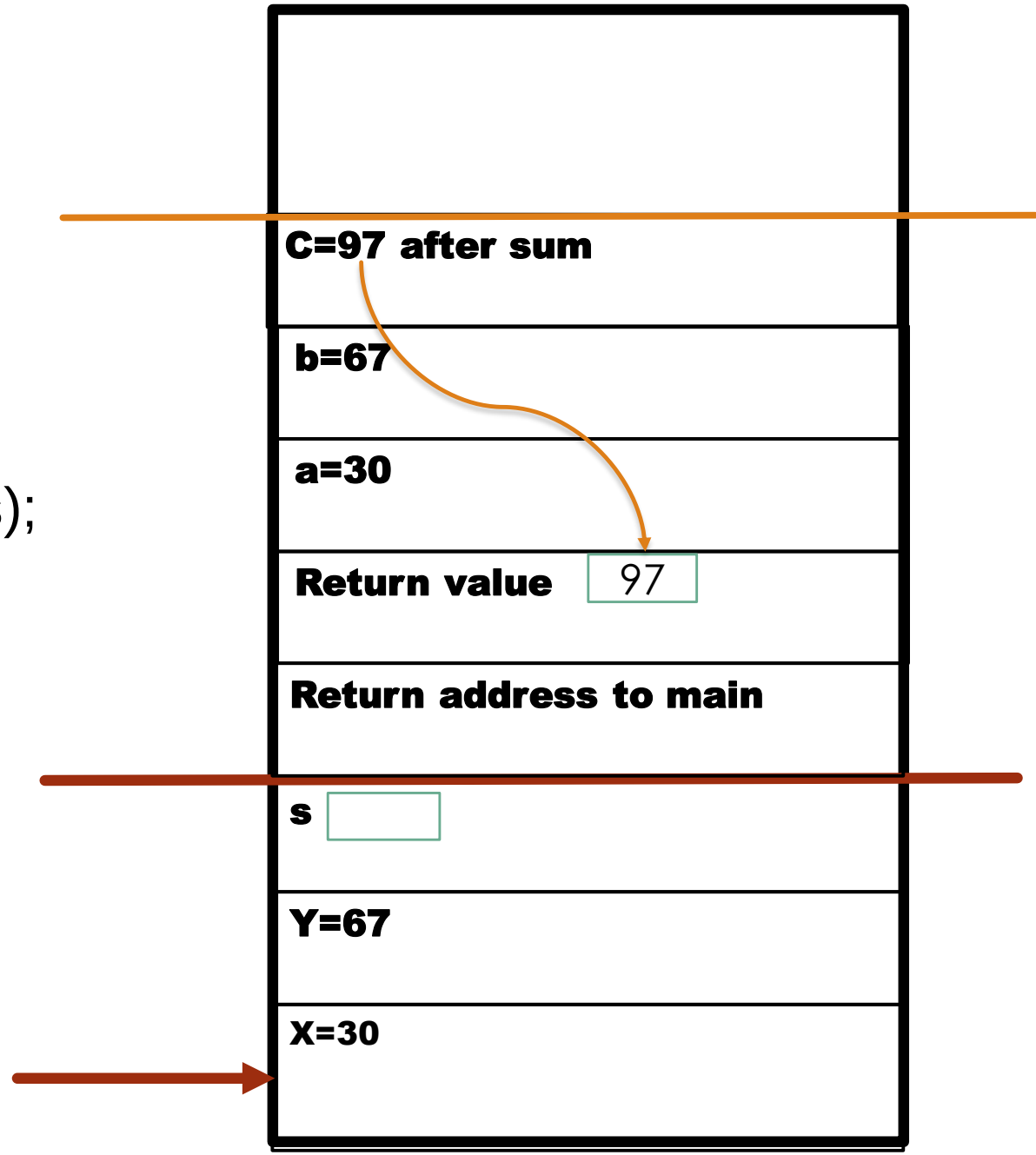
```
#include <iostream>
Int square(int x)
      { return x*x; }
Int sos(int a, int b)
{ intz=square(a+b);
Return z; }
Int main()
{
Int  total, x=4, y =2;
Total = sos(x,y);
Cout<< total;
}
```



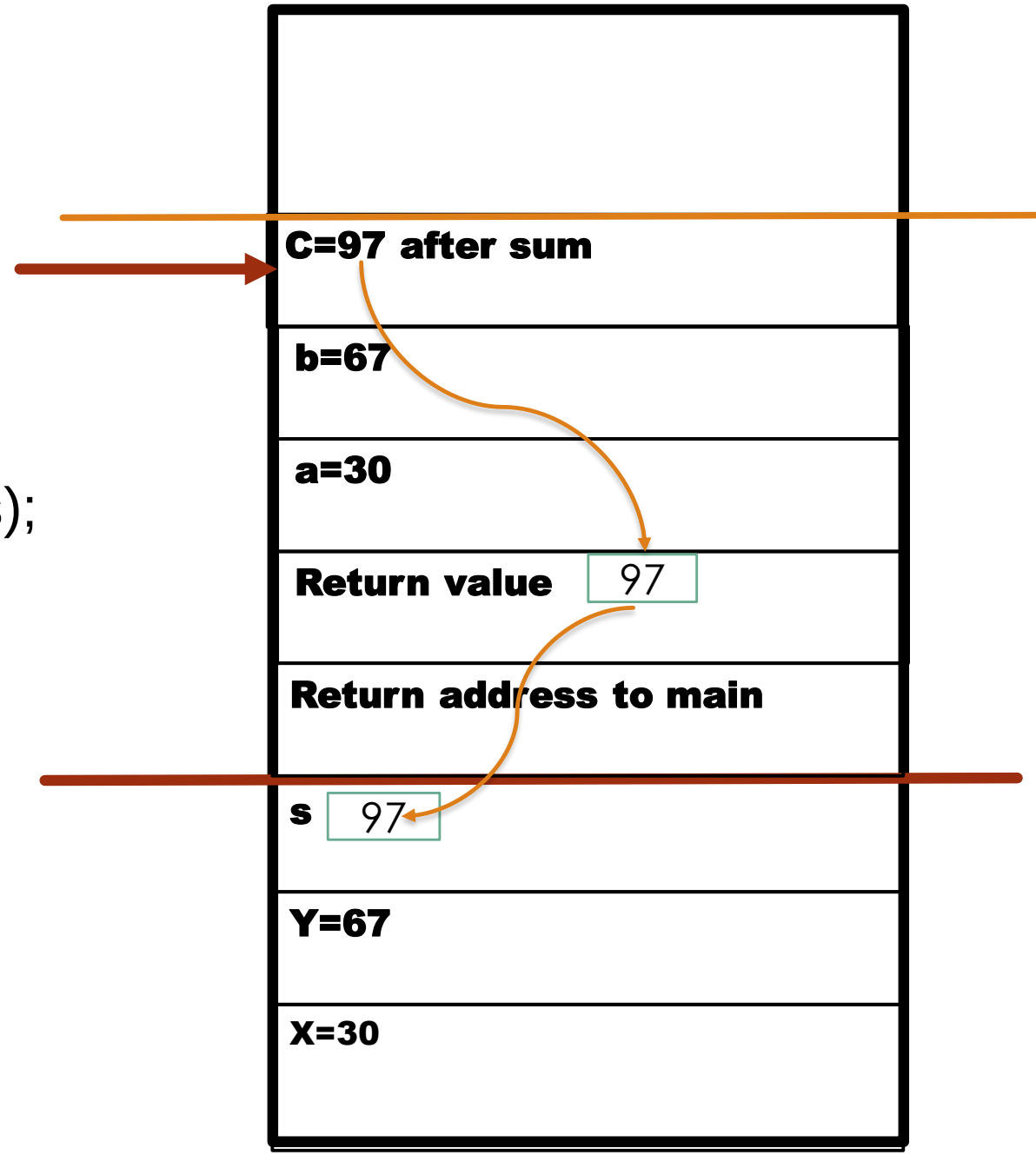
```
#include <iostream>
Int square(int x)
      { return x*x; }
Int sos(int a, int b)
{ intz=square(a+b);
Return z; }
Int main()
{
Int  total, x=4, y =2;
Total = sos(x,y);
Cout<<total;
}
```



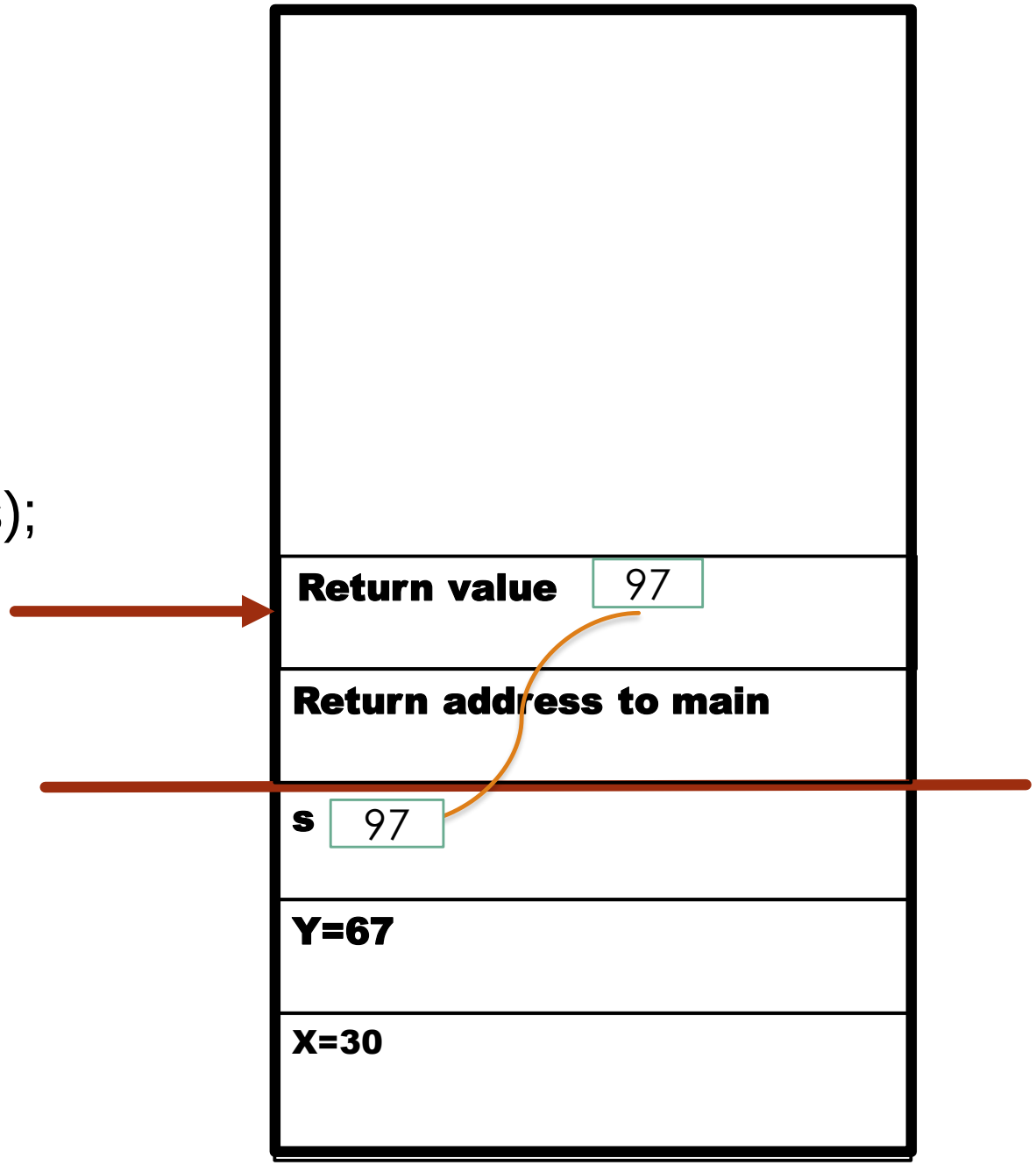
```
#include <iostream>
int sum(int,int);
int main()
{
 int x,y,s;
 x=30,y=67;
 s=sum(x,y);
 printf("Sum of two numbers %d",s);
 return 0;
}
int sum(int a,int b)
{
 int c;
 c=a+b;
 return c;
}
```



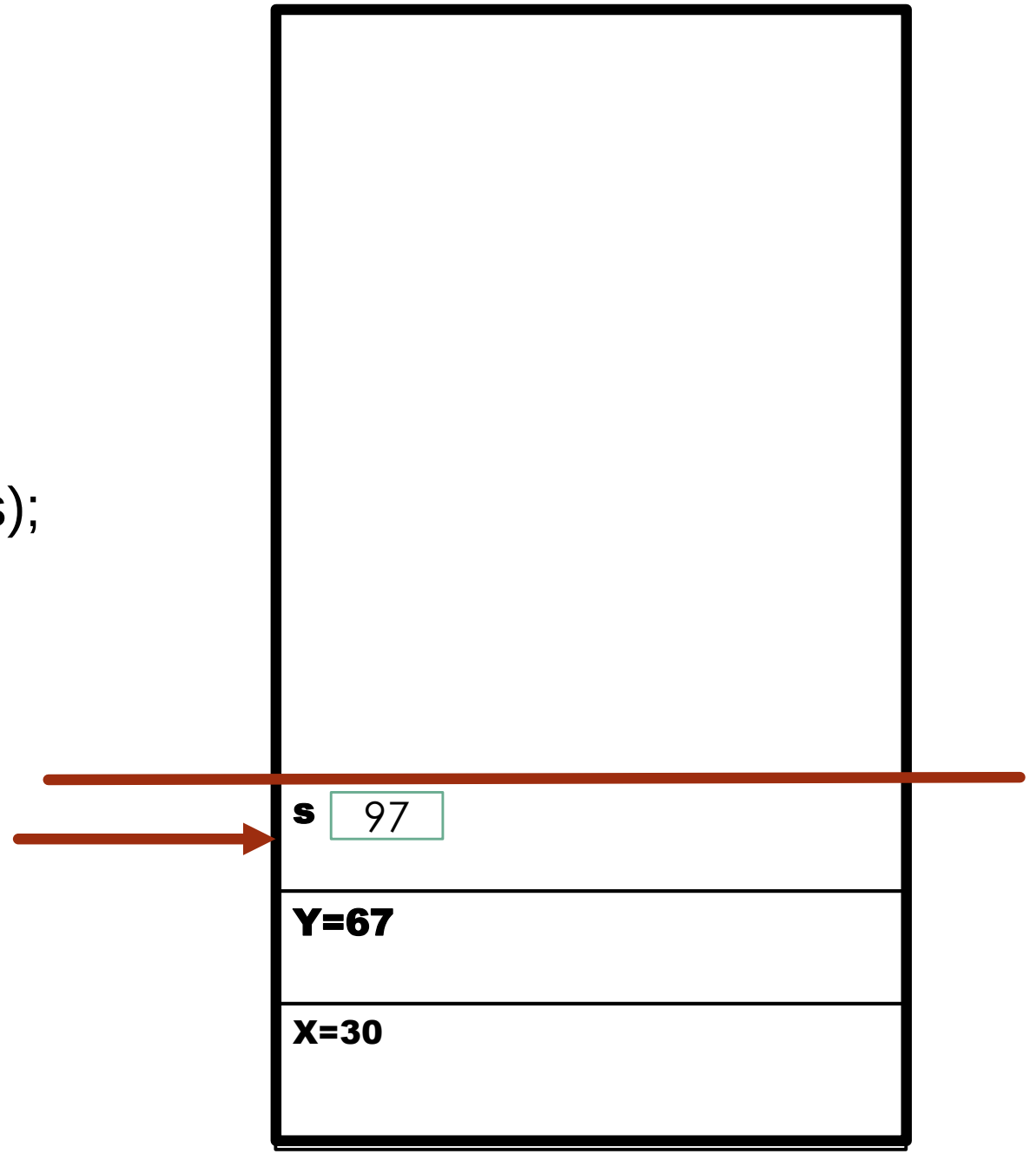
```
#include <iostream>
int sum(int,int);
int main()
{
 int x,y,s;
 x=30,y=67;
 s=sum(x,y);
 printf("Sum of two numbers %d",s);
 return 0;
}
int sum(int a,int b)
{
 int c;
 c=a+b;
 return c;
}
```



```
#include <iostream>
int sum(int,int);
int main()
{
 int x,y,s;
 x=30,y=67;
 s=sum(x,y);
 printf("Sum of two numbers %d",s);
 return 0;
}
int sum(int a,int b)
{
 int c;
 c=a+b;
 return c;
}
```



```
#include <iostream>
int sum(int,int);
int main()
{
 int x,y,s;
 x=30,y=67;
 s=sum(x,y);
 printf("Sum of two numbers %d",s);
 return 0;
}
int sum(int a,int b)
{
 int c;
 c=a+b;
 return c;
}
```



```
#include <iostream>
 int sum(int,int);
int main()
{
 int x,y,s;
 x=30,y=67;
 s=sum(x,y);
 printf("Sum of two numbers %d",s);
 return 0;
}
int sum(int a,int b)
{
 int c;
 c=a+b;
 return c;
}
```