ROBOTICS CORNER

Namespaces

Mohamed Saied

Namespace

```
// A program to demonstrate need of namespace
int main()
{
    int value;
    value = 0;
    double value; // Error here
    value = 0.0;
}
```

Namespace 2

```
// Here we can see that more than one
variables
// are being used without reporting any error.
// That is because they are declared in the
// different namespaces and scopes.
#include <iostream>
using namespace std;

// Variable created inside namespace
namespace first
{
    int val = 500;
}

// Global variable
int val = 100;

int main()
{
    // Local variable
    int val = 200;

    // These variables can be accessed from
    // outside the namespace using the scope
    // operator ::
    cout << first::val << '\n';

    return 0;
}
```

Namespace 3

```cpp
// Creating namespaces
#include <iostream>
using namespace std;
namespace ns1
{
    int value()    { return 5; }
}
namespace ns2
{
    const double x = 100;
    double value() {  return 2*x; }
}

int main()
{
    // Access value function within ns1
    cout << ns1::value() << '\n';

    // Access value function within ns2
    cout << ns2::value() << '\n';

    // Access variable x directly
    cout << ns2::x << '\n';

    return 0;
}
```

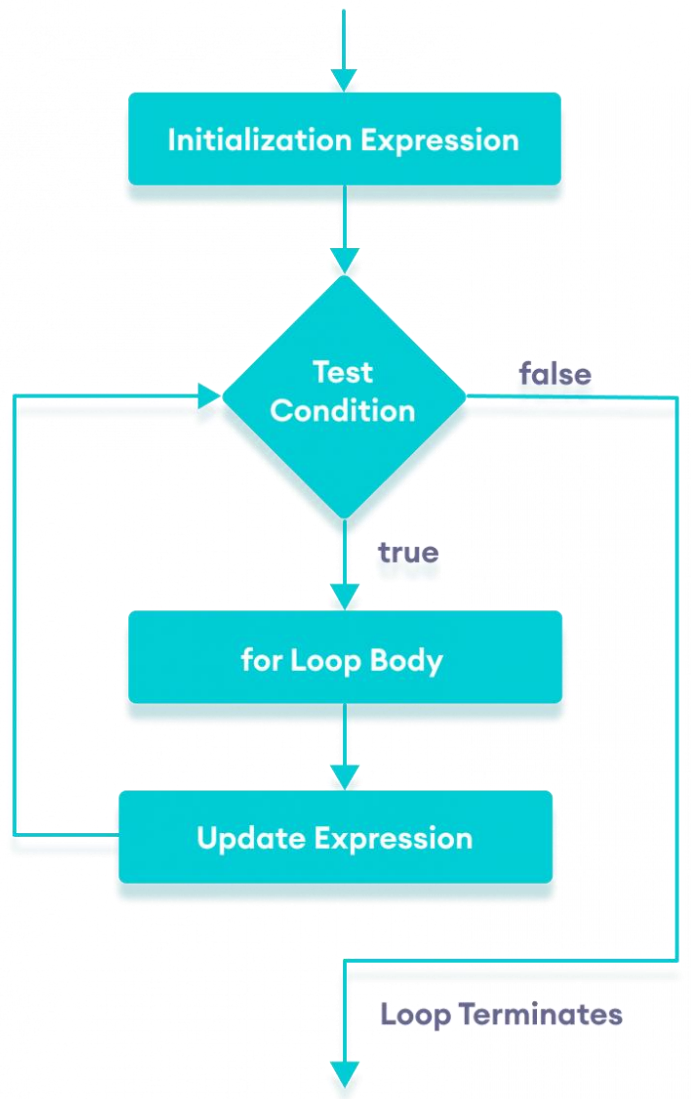
For loops

- for (initialization; condition; update) { // body of-loop }

•initialization - initializes variables and is executed only once

•condition - if true, the body of for loop is executed
if false, the for loop is terminated

•update - updates the value of initialized variables and again checks the condition


Initialization Expression
Test Condition
false
true
for Loop Body
Update Expression
Loop Terminates

Range based for loops c++11

```
// the initializer may be a braced-init-list
for (int n : {0, 1, 2, 3, 4, 5})
    std::cout << n << ' ';

std::cout << '\n';

// Iterating over array
int a[] = {0, 1, 2, 3, 4, 5};
for (int n : a)
    std::cout << n << ' ';

std::cout << '\n';

// Just running a loop for every array
// element
for (int n : a)
    std::cout << "In loop" << ' ';
```

Memory Management

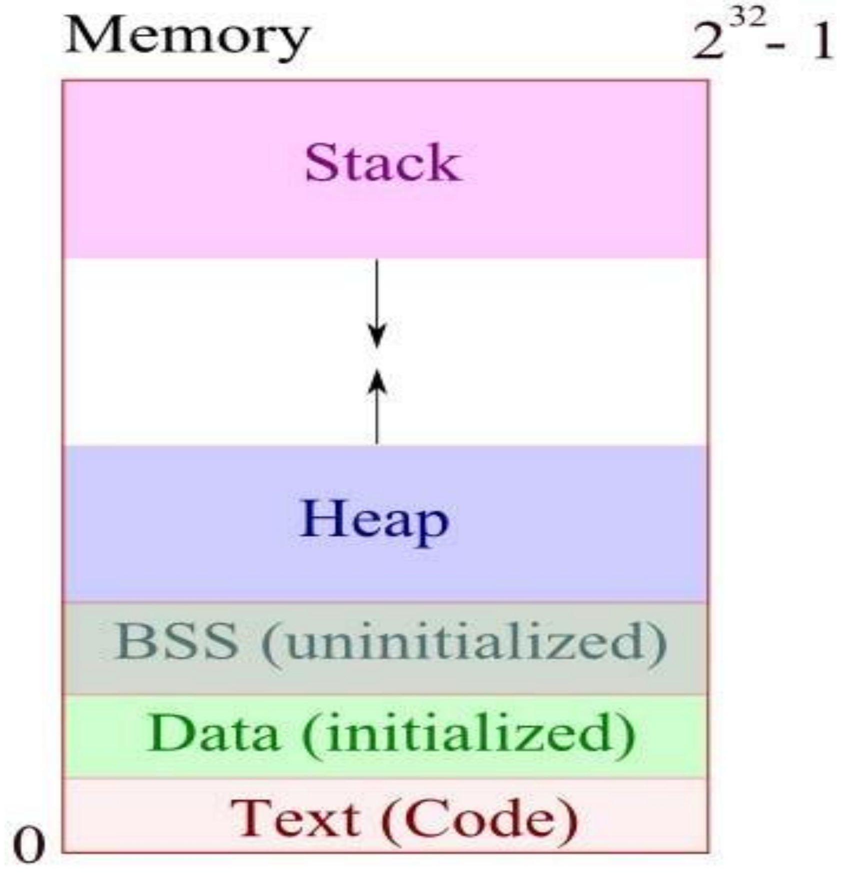
Outline

- The Course consists of the following topics:
  - **Memory Layout**
  - Stack
  - Call Stack
  - Data Segment
  - Heap
  - Rodata segment

Outline

- The Course consists of the following topics:
  - **Memory Layout**
  - Stack
  - Call Stack
  - Data Segment
  - Heap
  - Rodata segment

Memory Layout



Memory Layout diagram courtesy of bogotobogo.com

Outline

- The Course consists of the following topics:
  - Memory Layout
  - **Stack**
  - Call Stack
  - Data Segment
  - Heap
  - Rodata segment

Outline

- The Course consists of the following topics:
  - Memory Layout
  - **Stack**
  - Call Stack
  - Data Segment
  - Heap
  - Rodata segment

Stack

- Stack contains local variables from functions and related book-keeping data. LIFO structure.
  - Function variables are pushed onto stack when called.
  - Functions variables are popped off stack when return.

Outline

- The Course consists of the following topics:
  - Memory Layout
  - Stack
  - **Call Stack**
  - Data Segment
  - Heap
  - Rodata segment

Outline

- The Course consists of the following topics:
  - Memory Layout
  - Stack
  - **Call Stack**
  - Data Segment
  - Heap
  - Rodata segment

Call Stack

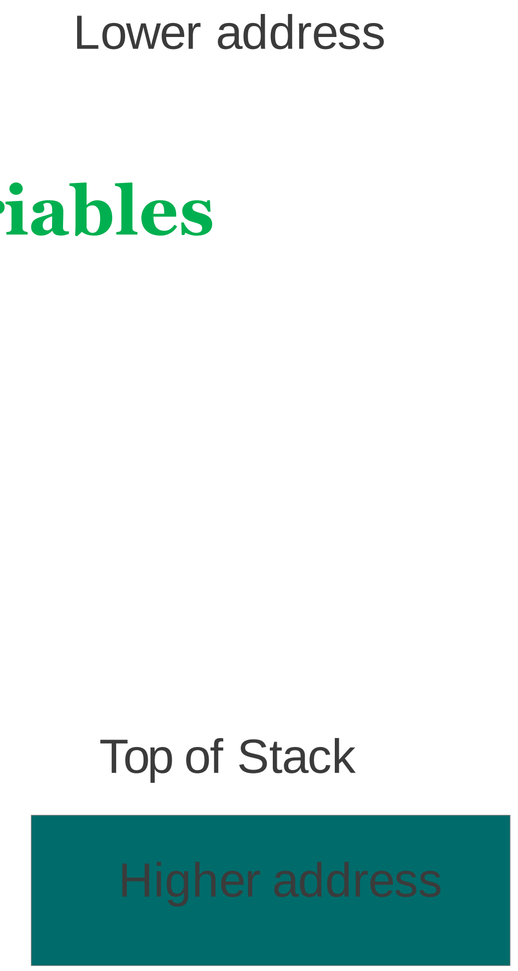
Example: DrawSquare called from main()

```
void DrawSquare(int i){
      int start, end, .... //other local variables
      DrawLine(start, end);
 }
 void DrawLine(int start, int end){
      //local variables
      ...
 }
```

Call Stack

Example:

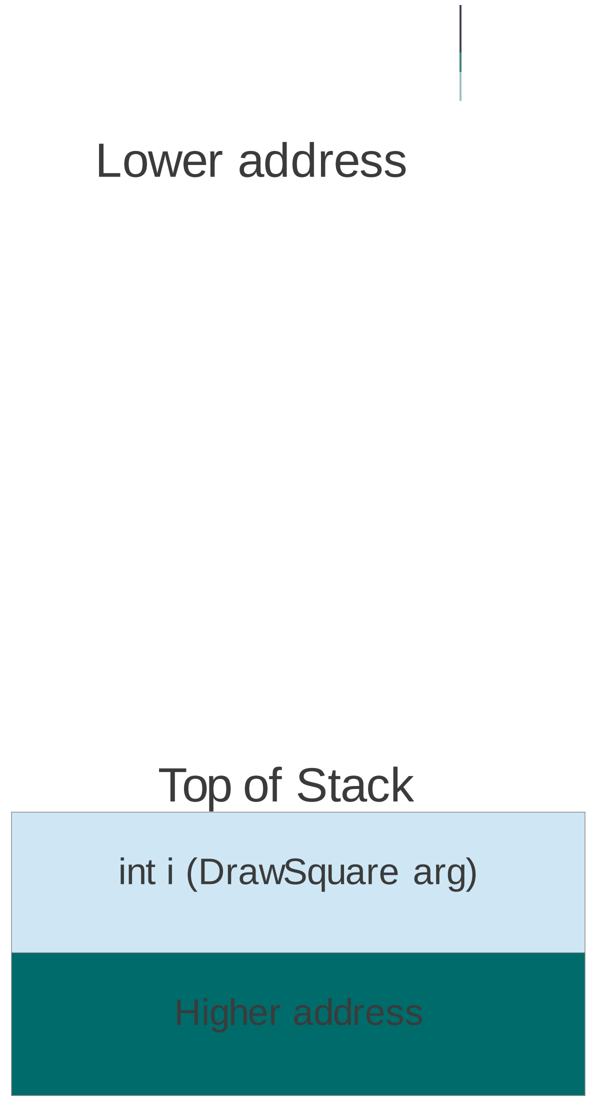
```
void DrawSquare(int i){
    int start, end, .... //other local variables
    DrawLine(start, end);
 }
 void DrawLine(int start, int end){
    //local variables
    ...
 }
```



Call Stack

Example: DrawSquare is called in main

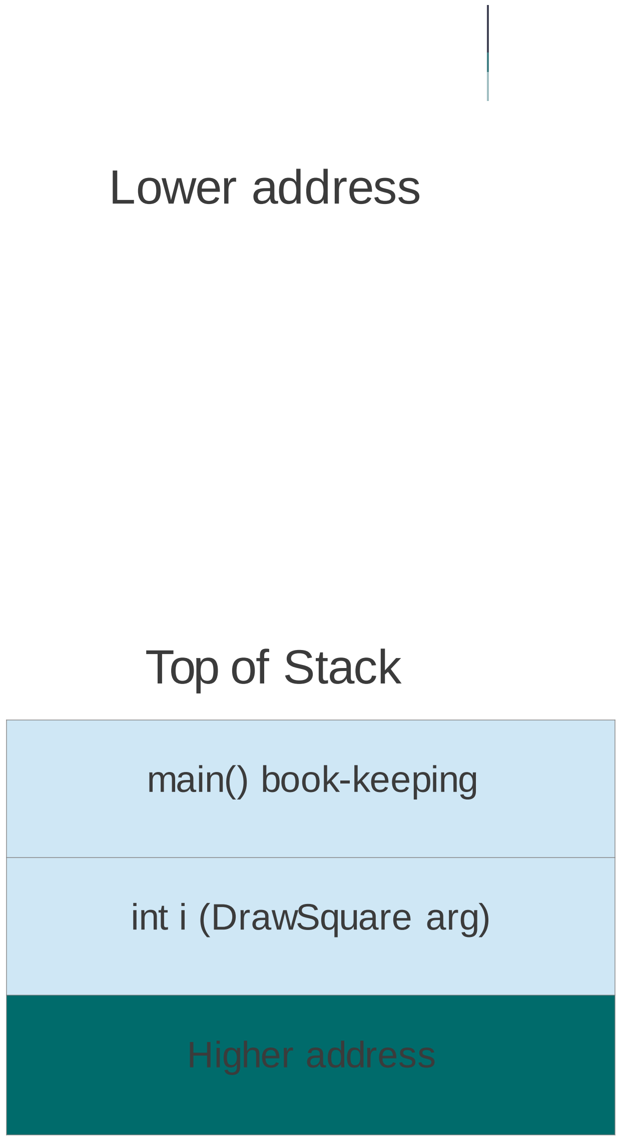
```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,  int end){
    //local variables
    ...
    }
```



Call Stack

Example:

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,  int end){
   //local variables
        ...
   }
```

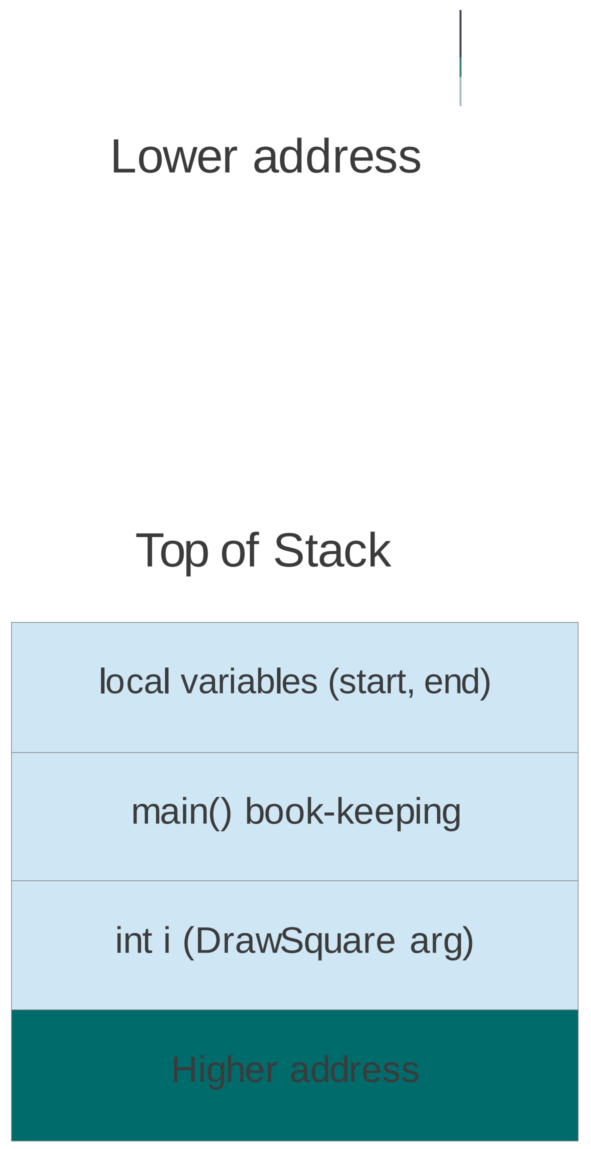


Call Stack

Example:

```
 void DrawSquare(int i){

   int start, end, ...

   DrawLine(start, end);
}
void DrawLine(int start,  int end)
{
        //local variables
        ...

 }
```

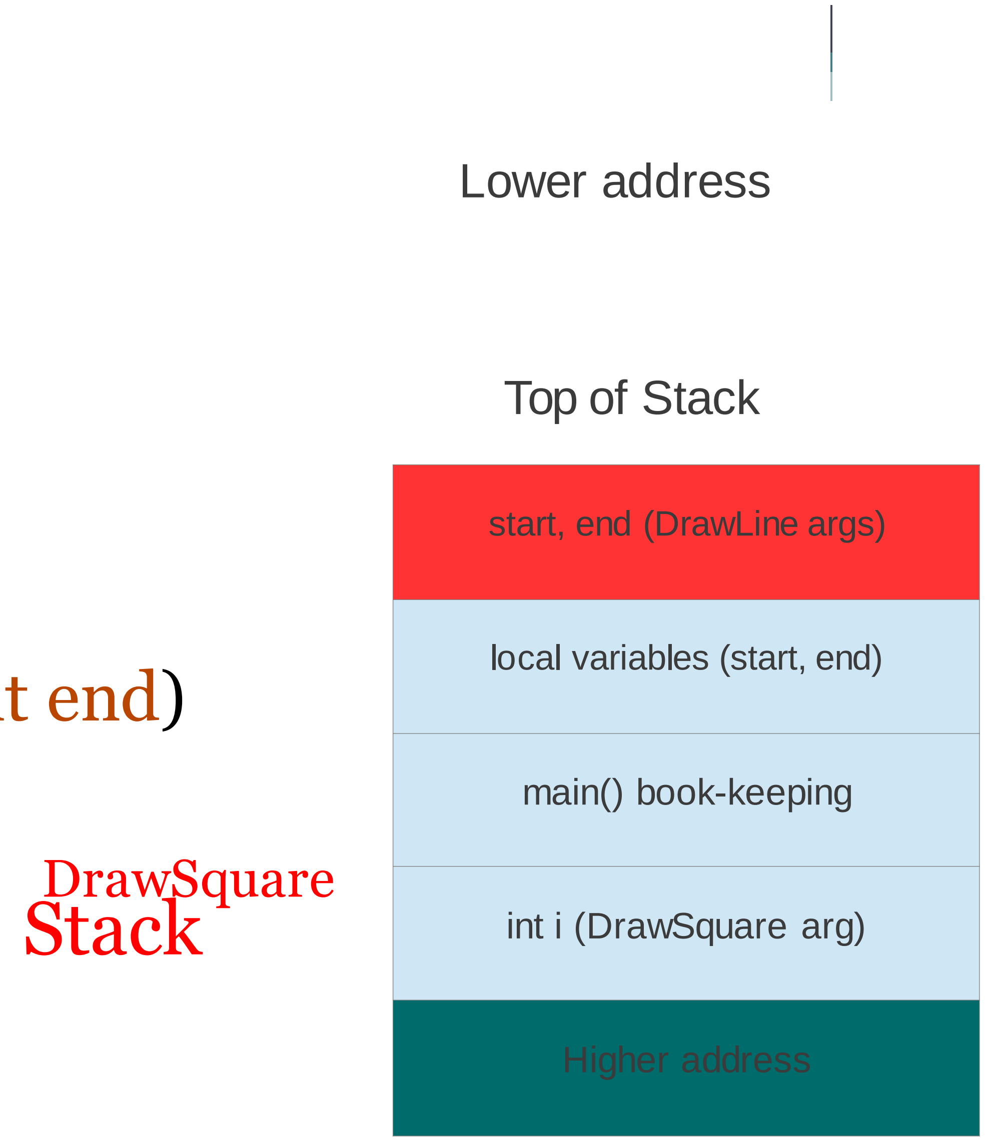
DrawSquare
Stack Frame



Call Stack

Example:

```
  void DrawSquare(int i){

    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,  int end)
{        //local variables
         ...
}
```

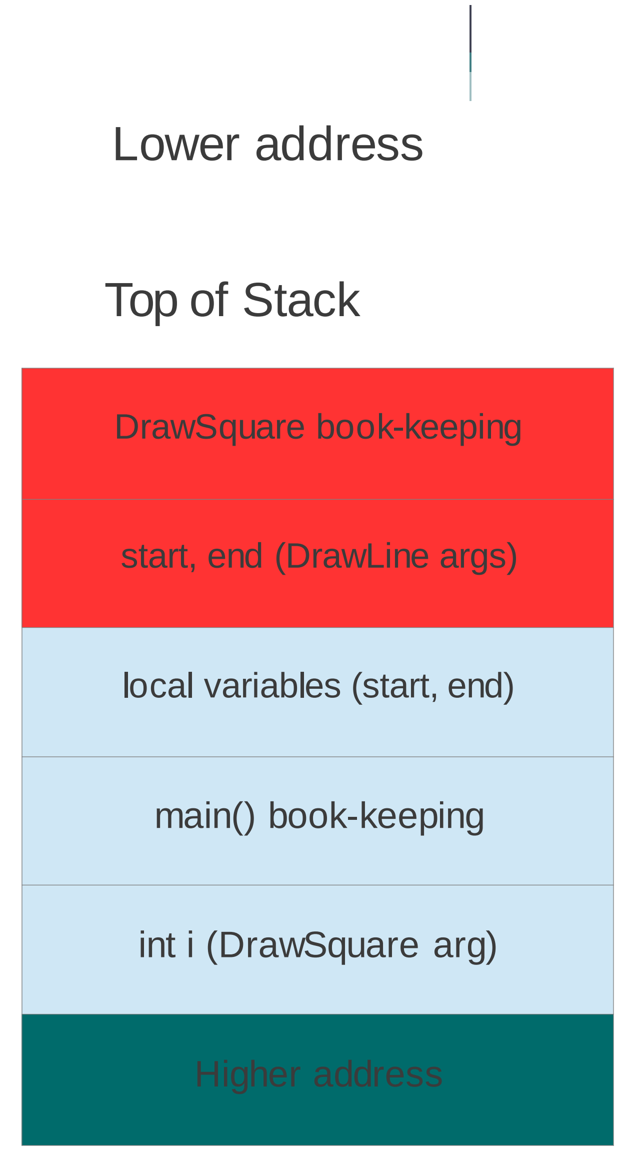


Call Stack

Example:

```
  void DrawSquare(int i){
   int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,  int end)
{
        //local
        variables
        ...
    }
```

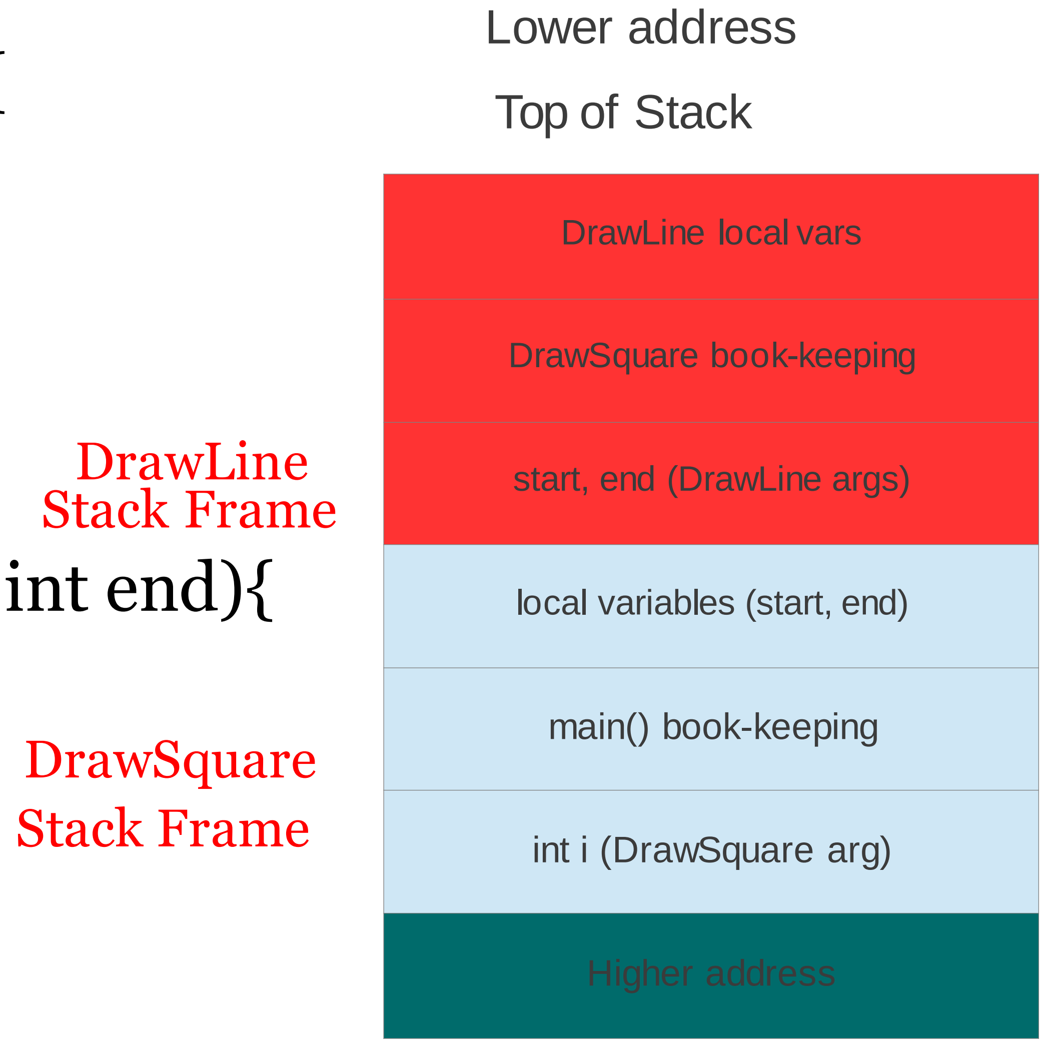
DrawSquare Stack Frame



Call Stack

Example:

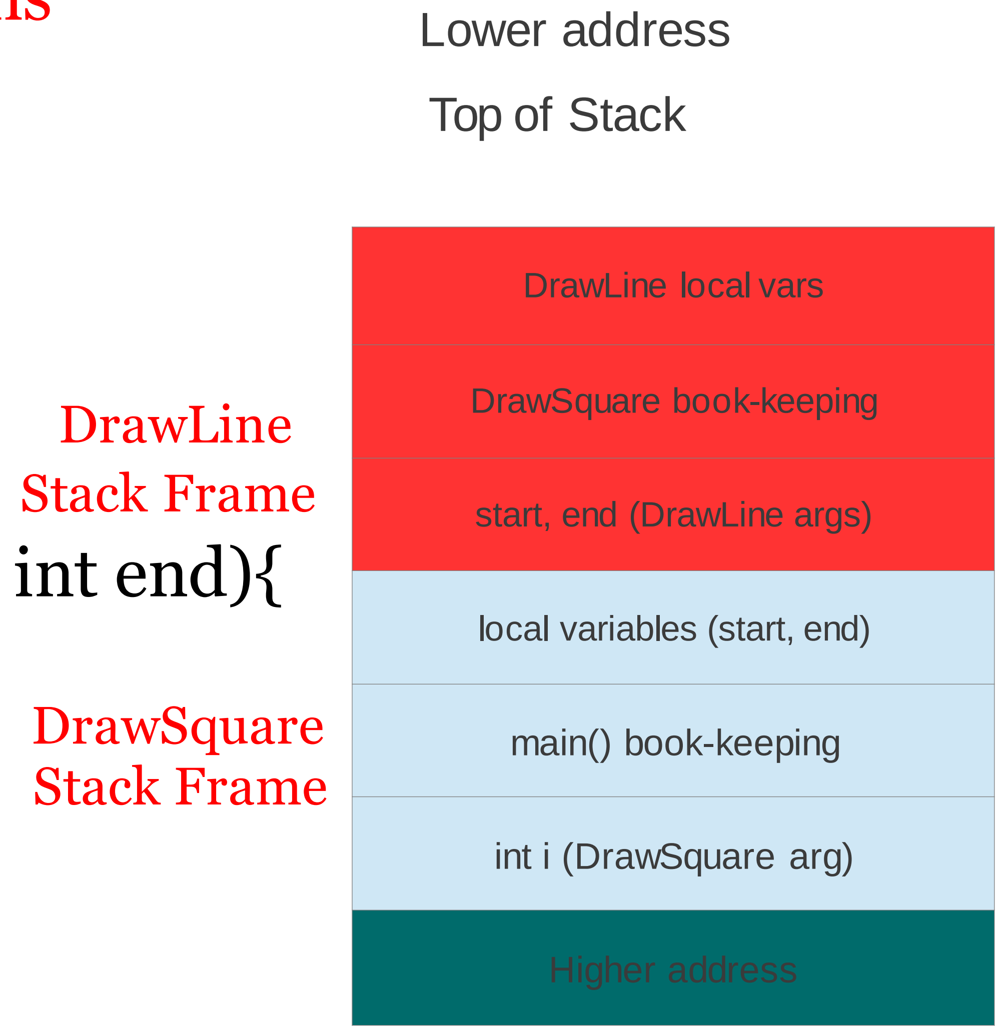
```
void DrawSquare(int i){
  int start, end, ...
  DrawLine(start, end);
}

void DrawLine(int start,  int end){

//local variables

    ...
}
```



Call Stack

Example: DrawLine returns

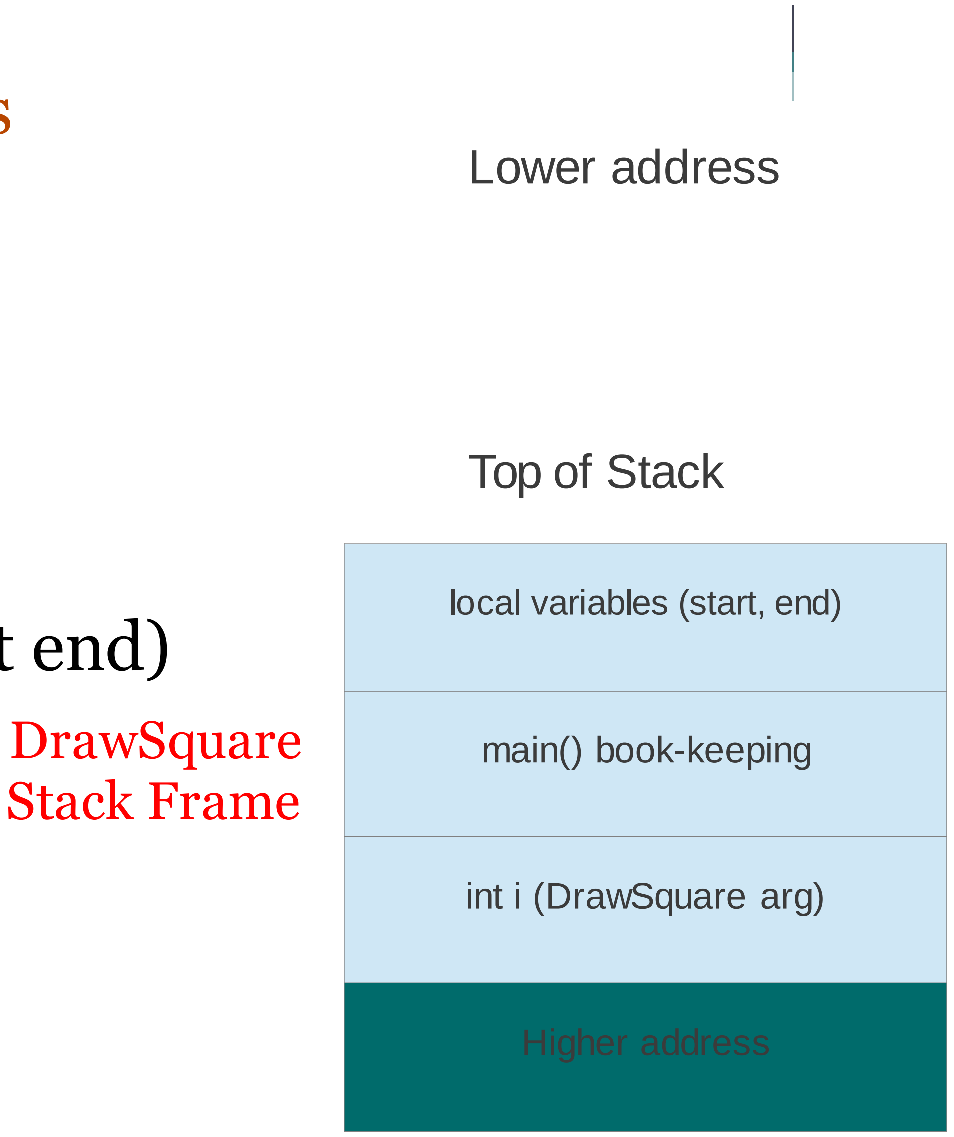
```
void DrawSquare(int i){
int start, end, ...
DrawLine(start, end);
}

void DrawLine(int start,  int end){

    //local variables
        ...
}
```



Call Stack

Example: DrawLine returns

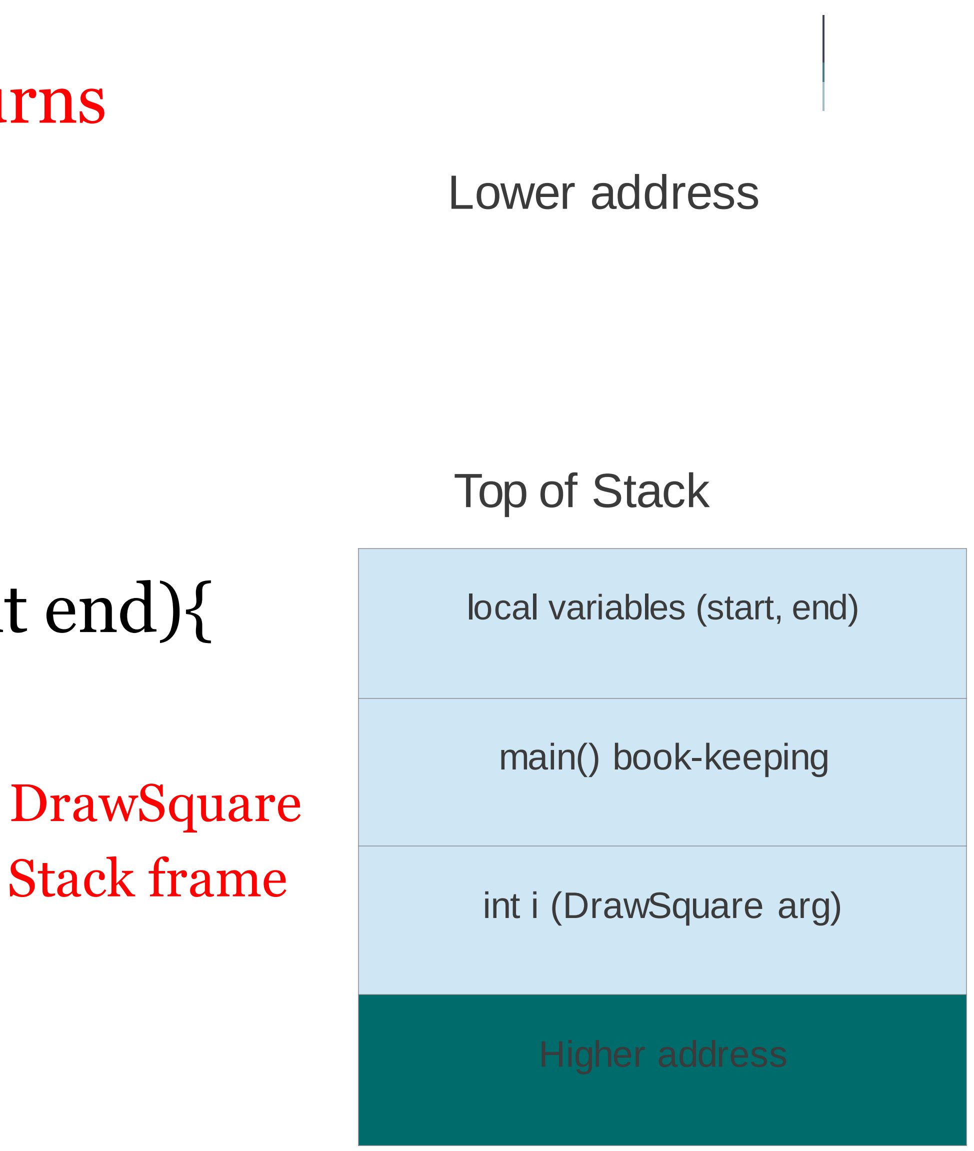
```
  void DrawSquare(int i){
   int start, end, ...
   DrawLine(start, end);
}
void DrawLine(int start,  int end)
{

        //local variables
        ...
}
```



Call Stack

Example: DrawSquare returns

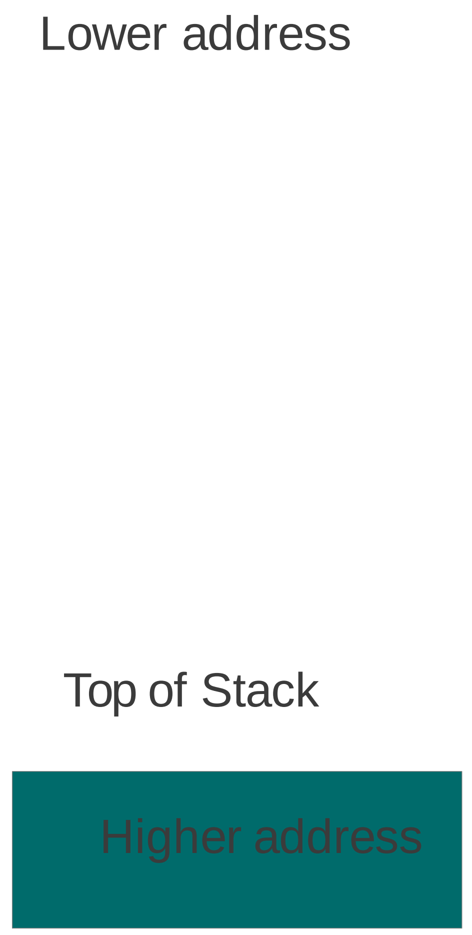
```
  void DrawSquare(int i){
   int start, end, ...
   DrawLine(start, end);
}
void DrawLine(int start,  int end){
         //local variables
         ...

}
```



Call Stack

Example: DrawSquare returns

```
void DrawSquare(int i){  int start, end,
…
    DrawLine(start, end);
}
void DrawLine(int start,  int end){
    //local variables
    …
    }
```



Reference

- A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

Reference Vs Pointers

- References are often confused with pointers but three major differences between references and pointers are –
- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be initialized when it is created. Pointers can be initialized at any time.

Pointers

- A variable that holds the address of another variable