Introducción a C++

Diego Useche - dh.useche@uniandes.edu.co

Metodos Computacionales II

Physics Department, Universidad de los Andes, Bogotá

Pointers: pointer address

- &score, indicates the address in memory of variable score.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int score = 5;
    cout << &score << endl;

    return 0;
}
```

Pointers: pointer address

- &score, indicates the address in memory of variable score.

```
#include <iostream>
using namespace std;

int main()
{
    int score = 5;
    cout << &score << endl;

    return 0;
}
```

Your Output

```
0x7ffffa2568bc
```

prints hexadecimal address of variable in memory

Pointers: creating a pointer variable

- a pointer is variable that saves the address in memory of another variable
- int* scorePtr, indicates that scorePtr is a pointer of an int

```
#include <iostream>
using namespace std;

int main()
{
    int score = 5;
    int* scorePtr;
    scorePtr = &score;

    cout << scorePtr << endl;

    return 0;
}
```

Your Output

```
0x7ffffa2568bc
```

Pass by reference vs pass by value

By value

```cpp
#include <iostream>
using namespace std;

void myFunc(int x) {
    x = 100;
}

int main() {
    int var = 20;
    myFunc(var);
    cout << var;
}
```

By reference

```cpp
#include <iostream>
using namespace std;

void myFunc(int* x) {
    *x = 100;
}

int main() {
    int var = 20;
    myFunc(&var);
    cout << var;
}
```

Create a pointer to a Class

We build an instance of the class

```cpp
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    cout << mytriangle.a << endl;
    cout << mytriangle.perimeter();
    return 0;
}
```

Create a pointer to a Class

Your Output

```
2
9
```

```
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    cout << mytriangle.a << endl;
    cout << mytriangle.perimeter();
    return 0;
}
```

Create a pointer to a Class

Create a pointer to a Class, and access its members and functions

```
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    Triangle* ptrMytriangle = &mytriangle;
    cout << ptrMytriangle.a << endl;
    cout << ptrMytriangle.perimeter();
    return 0;
}
```

Create a pointer to a Class

Your Output

```
/usercode/file0.cpp: In function 'int main()':
/usercode/file0.cpp:30:24: error: request for member
'a' in 'ptrMytriangle', which is of pointer type
'Triangle*' (maybe you meant to use '->' ?)
   30 |   cout << ptrMytriangle.a << endl;
      |                         ^
/usercode/file0.cpp:31:24: error: request for member
'perimeter' in 'ptrMytriangle', which is of pointer
type 'Triangle*' (maybe you meant to use '->' ?)
   31 |   cout << ptrMytriangle.perimeter();
      |                         ^~~~~~~~~
```

```cpp
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    Triangle* ptrMytriangle = &mytriangle;
    cout << ptrMytriangle.a << endl;
    cout << ptrMytriangle.perimeter();
    return 0;
}
```

Create a pointer to a Class

```
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    Triangle* ptrMytriangle = &mytriangle;
    cout << ptrMytriangle->a << endl;
    cout << ptrMytriangle->perimeter();
    return 0;
}
```

Use " -> " to access the attributes and functions of a pointer of a Class.

Create a pointer to a Class

```cpp
class Triangle
{
    public:

        float a;
        float b;
        float c;

        Triangle(int a_param, int b_param, int c_param)
        {
            a = a_param;
            b = b_param;
            c = c_param;
        }

        float perimeter()
        {
            float perim = a + b + c;
            return perim;
        }

};

int main() {
    Triangle mytriangle(2, 3, 4);
    Triangle* ptrMytriangle = &mytriangle;
    cout << ptrMytriangle->a << endl;
    cout << ptrMytriangle->perimeter();
    return 0;
}
```

Your Output

```
2
9
```

Stack vs Heap Memory

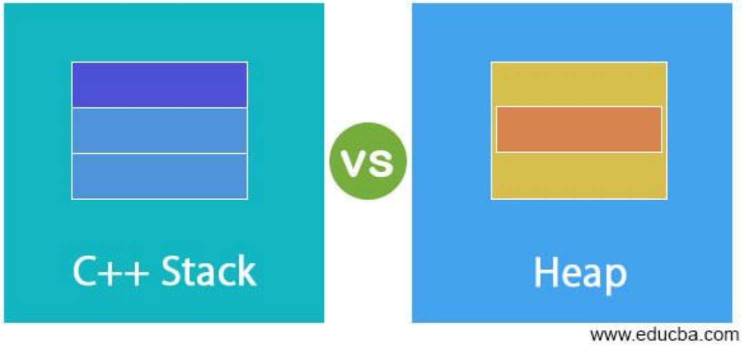
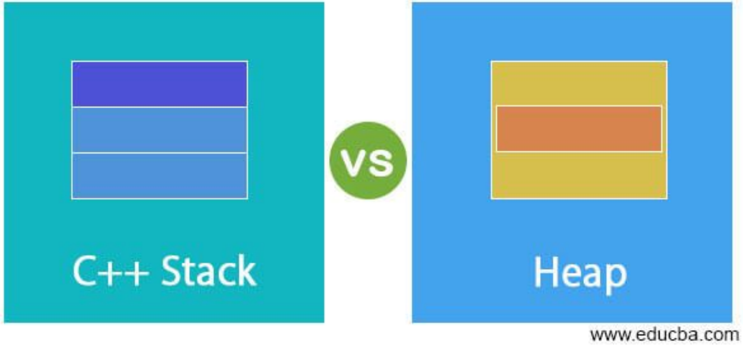


Image taken from https://www.educba.com/c-stack-vs-heap/

Stack vs Heap Memory



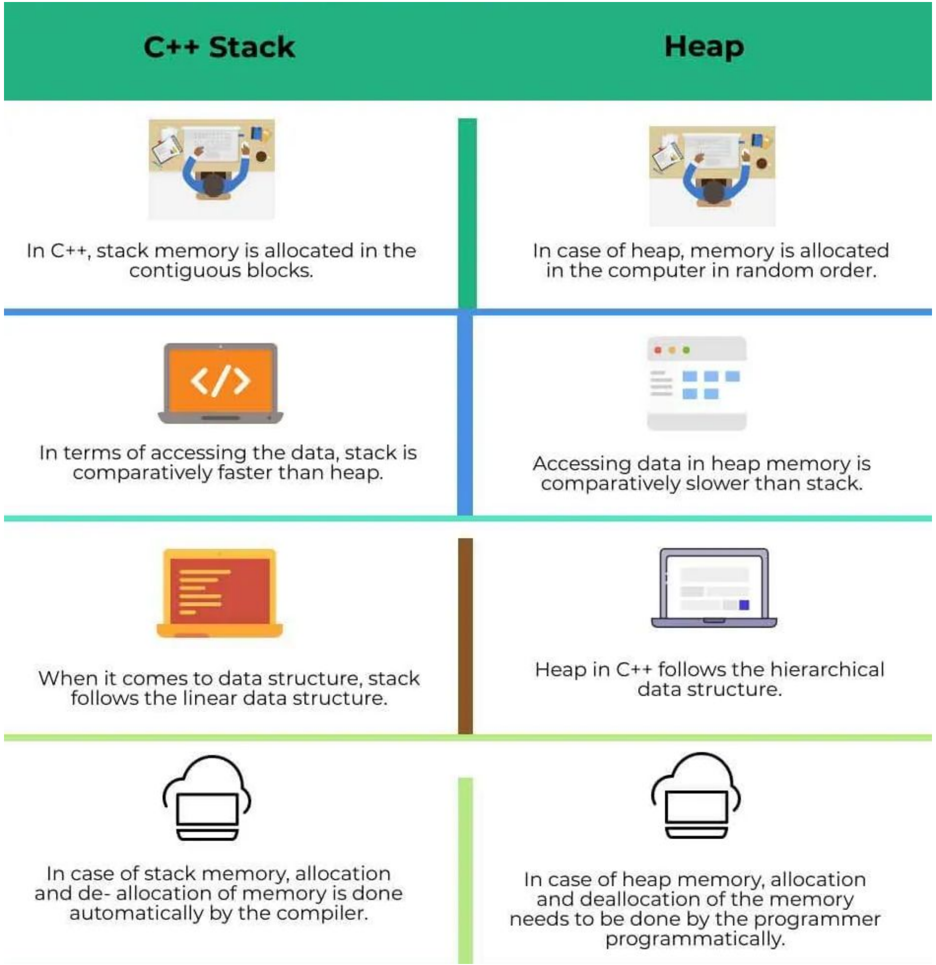
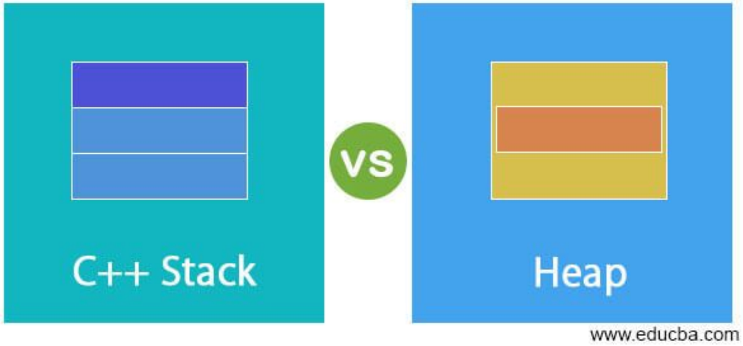
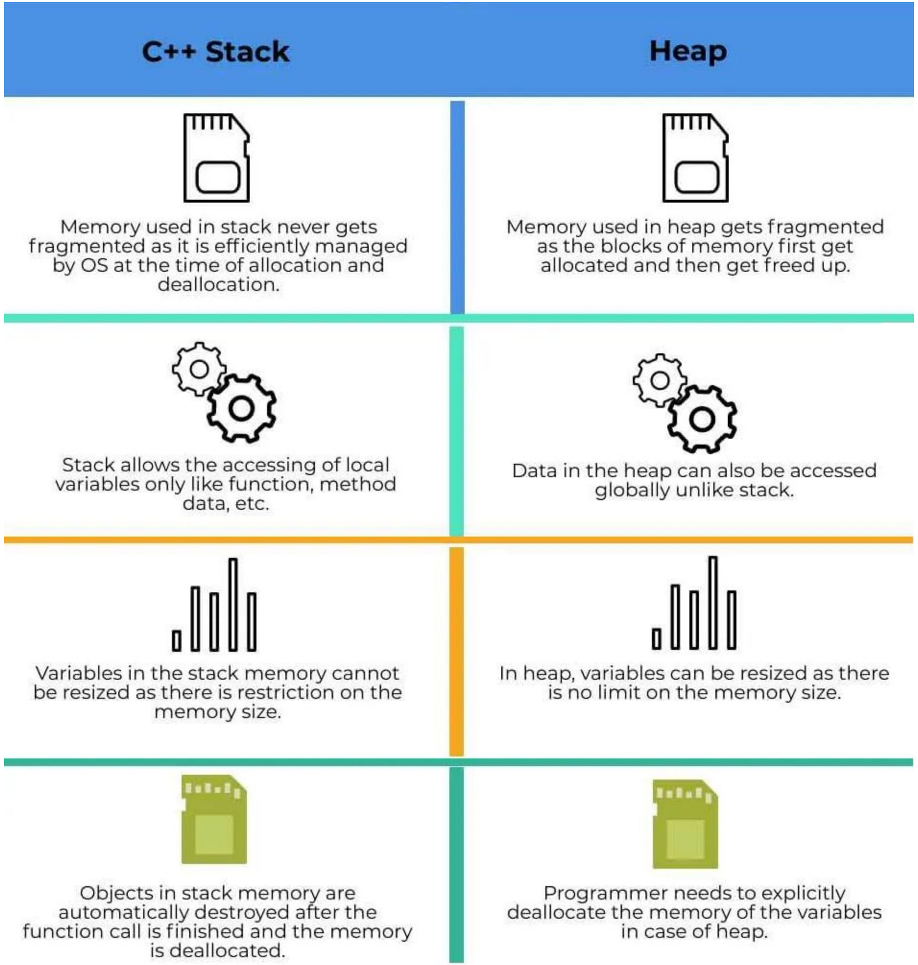


Image taken from https://www.educba.com/c-stack-vs-heap/

Stack vs Heap Memory





C++ Stack	Heap
Memory used in stack never gets fragmented as it is efficiently managed by OS at the time of allocation and deallocation.	Memory used in heap gets fragmented as the blocks of memory first get allocated and then get freed up.
Stack allows the accessing of local variables only like function, method data, etc.	Data in the heap can also be accessed globally unlike stack.
Variables in the stack memory cannot be resized as there is restriction on the memory size.	In heap, variables can be resized as there is no limit on the memory size.
Objects in stack memory are automatically destroyed after the function call is finished and the memory is deallocated.	Programmer needs to explicitly deallocate the memory of the variables in case of heap.

Image taken from https://www.educba.com/c-stack-vs-heap/

Create a variable in heap

```cpp
#include <iostream>
using namespace std;


int main() {
    int* ptrScore = new int(5);
    cout << *ptrScore << endl;
    cout << ptrScore << endl;
    delete ptrScore;
    return 0;
}
```

use new to create the novel variable in heap

deletions are not handled automatically in heap, pointer variables in heap must always be deleted.

Create a variable in heap

```cpp
#include <iostream>
using namespace std;


int main() {
    int* ptrScore = new int(5);
    cout << *ptrScore << endl;
    cout << ptrScore << endl;
    delete ptrScore;
    return 0;
}
```

use new to create the novel variable in heap

deletions are not handled automatically in heap, pointer values must always be deleted.

Your Output

```
5
0x192ae70
```

Example

Returns

- The function is declared with a `void` return type, so there is no value to return. Modify the values in memory so that a contains their sum and b contains their absoluted difference.
- $a' = a + b$
- $b' = |a - b|$

Input Format

Input will contain two integers, a and b, separated by a newline.

Sample Input

```
4
5
```

Sample Output

```
9
1
```

Problem taken from https://www.hackerrank.com/

References

https://www.cs.mtsu.edu/~xyang/2170/datatypes.html

https://www.sololearn.com/

https://www.hackerrank.com/

https://data-flair.training/blogs/c-tutorial/

https://www.astateofdata.com/python-programming/can-python-be-compiled/

https://slideplayer.com/slide/15836241/

https://www.youtube.com/watch?v=BdnpFbODLc0

https://www.educba.com/c-stack-vs-heap/