Programming with C++

LSEG Technology

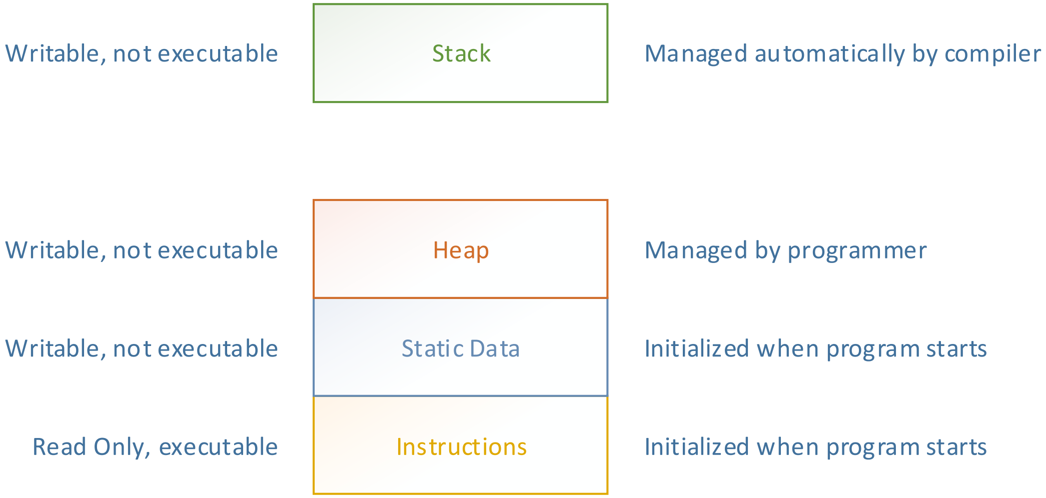
07th November 2022

Agenda

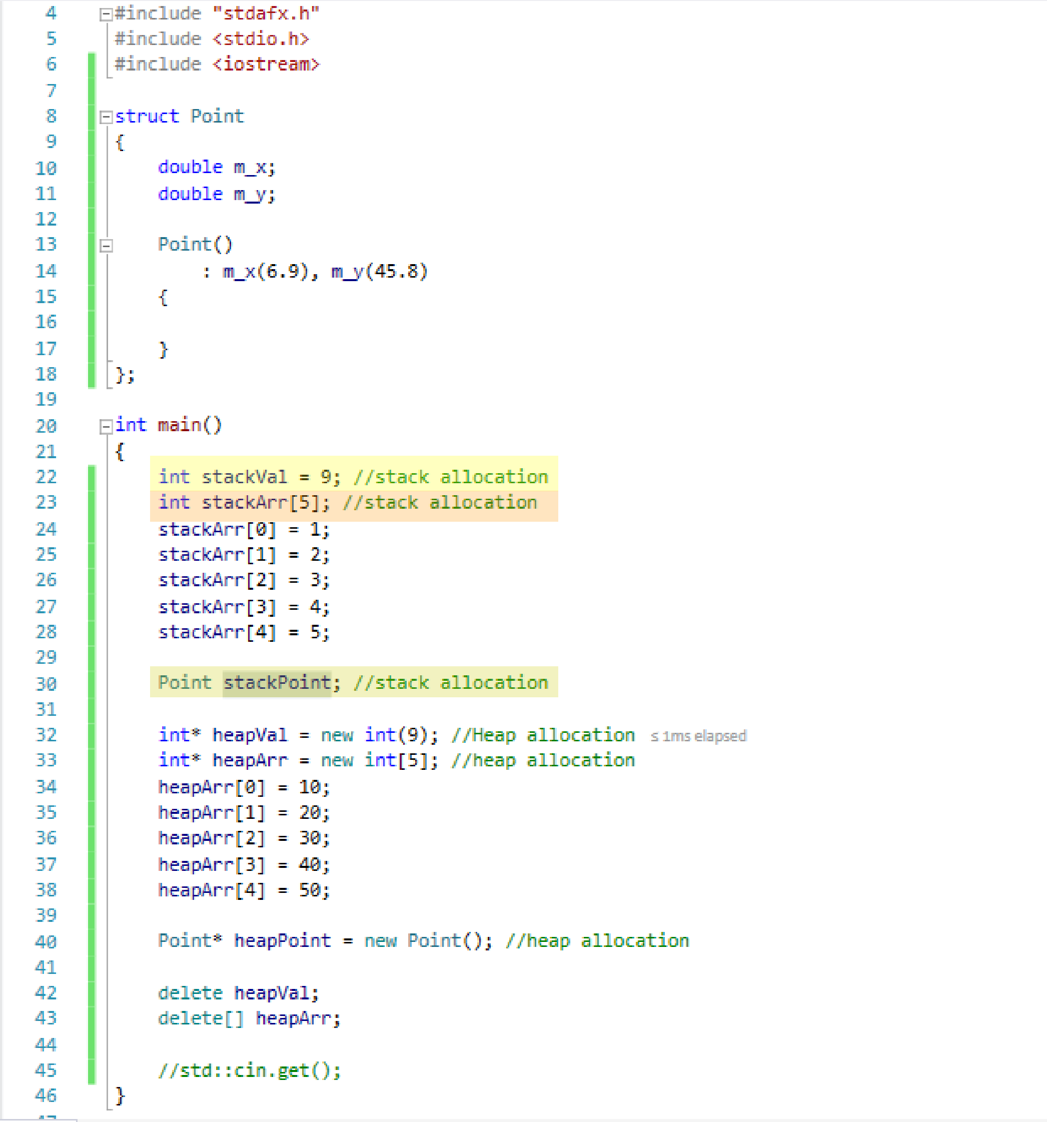
- Stack vs Heap Memory
- How Strings Work
- Dynamic Arrays (Vectors)
- Maps
- Templates
- Smart Pointers
- STL Algorithms
- Boost

Stack vs Heap Memory

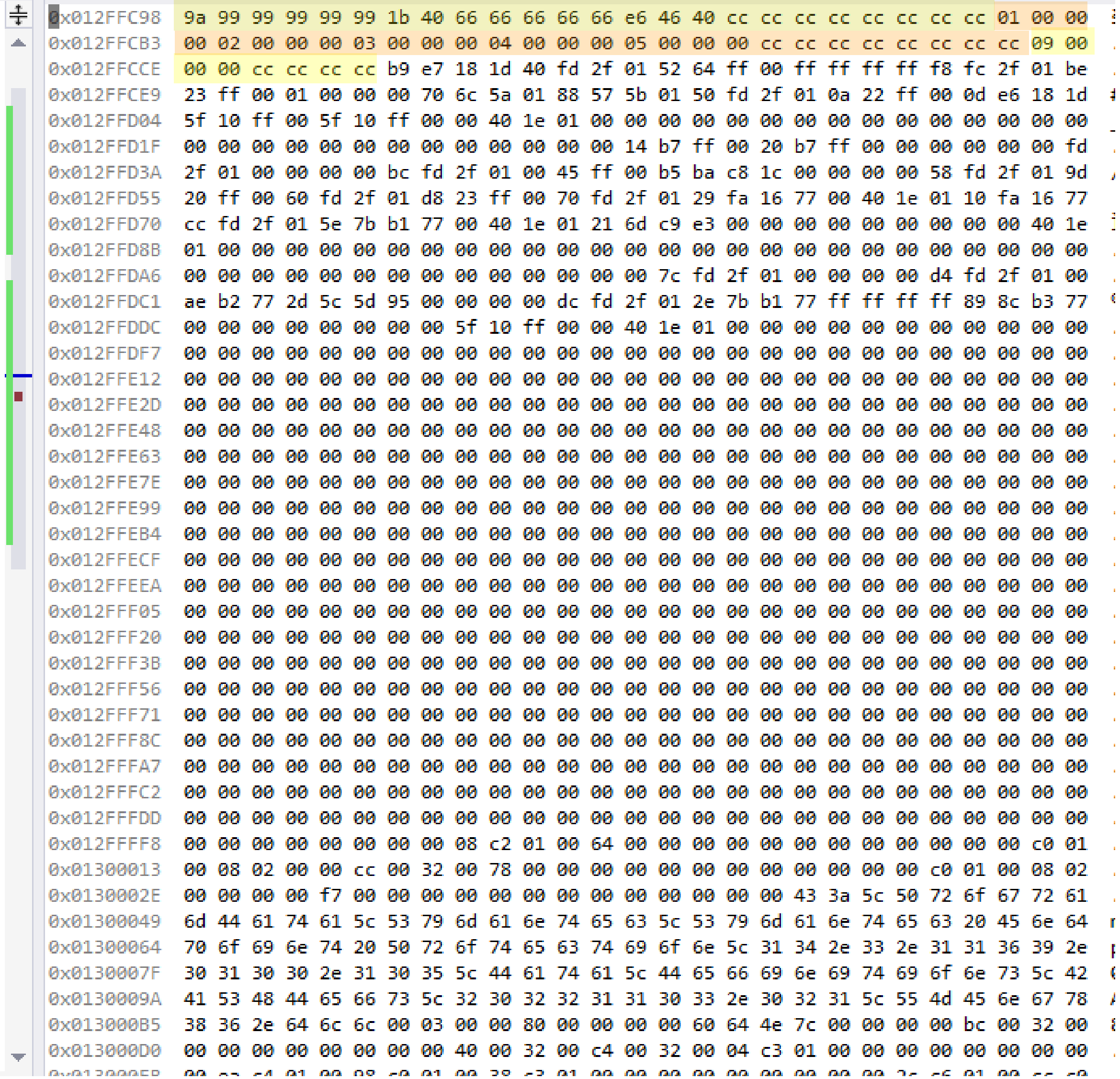
- Stack and heap what are they?
- Different memory areas
- Difference is how they allocate memory



Stack vs Heap Memory

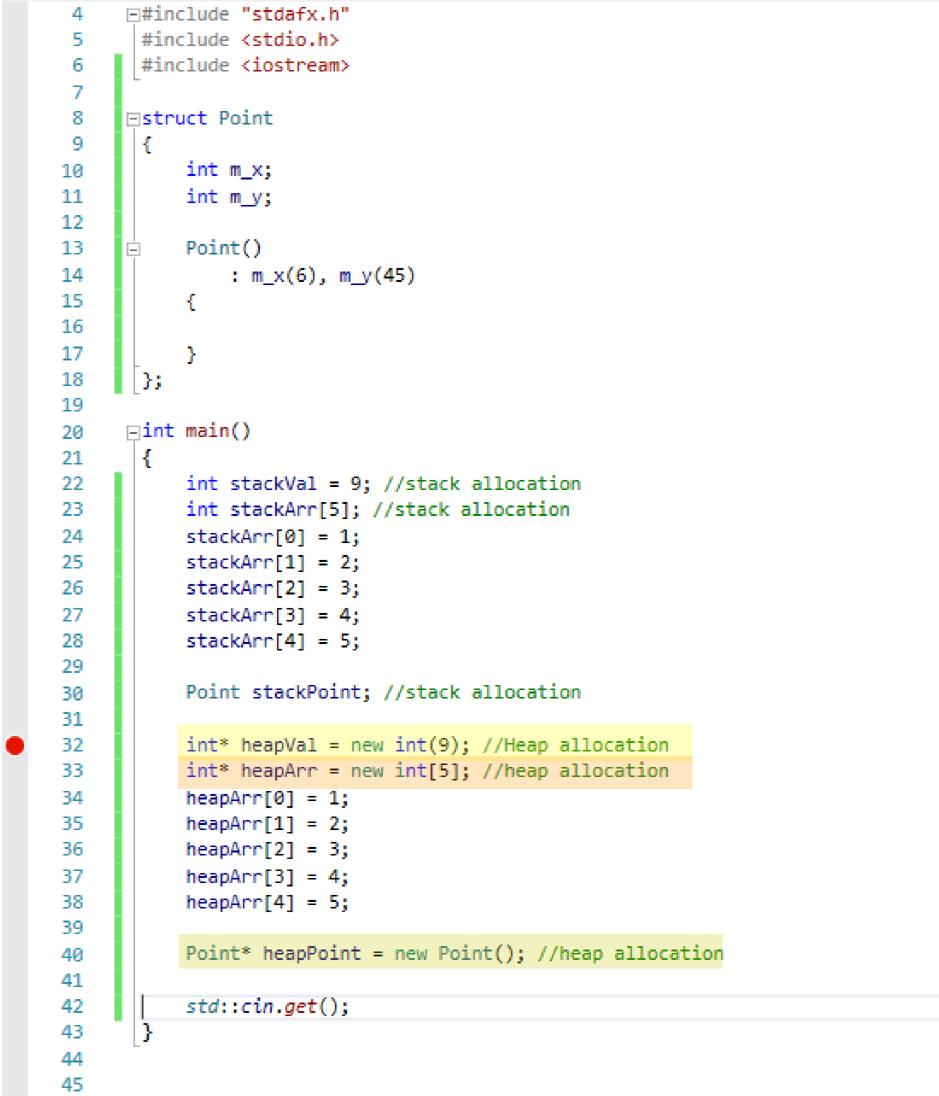


```
#include "stdafx.h"
#include <stdio.h>
#include <iostream>

struct Point
{
    double m_x;
    double m_y;

    Point()
        : m_x(6.9), m_y(45.8)
    {

    }
};

int main()
{
    int stackVal = 9; //stack allocation
    int stackArr[5]; //stack allocation
    stackArr[0] = 1;
    stackArr[1] = 2;
    stackArr[2] = 3;
    stackArr[3] = 4;
    stackArr[4] = 5;

    Point stackPoint; //stack allocation

    int* heapVal = new int(9); //Heap allocation  ≤ 1ms elapsed
    int* heapArr = new int[5]; //heap allocation
    heapArr[0] = 10;
    heapArr[1] = 20;
    heapArr[2] = 30;
    heapArr[3] = 40;
    heapArr[4] = 50;

    Point* heapPoint = new Point(); //heap allocation

    delete heapVal;
    delete[] heapArr;

    //std::cin.get();
}
```

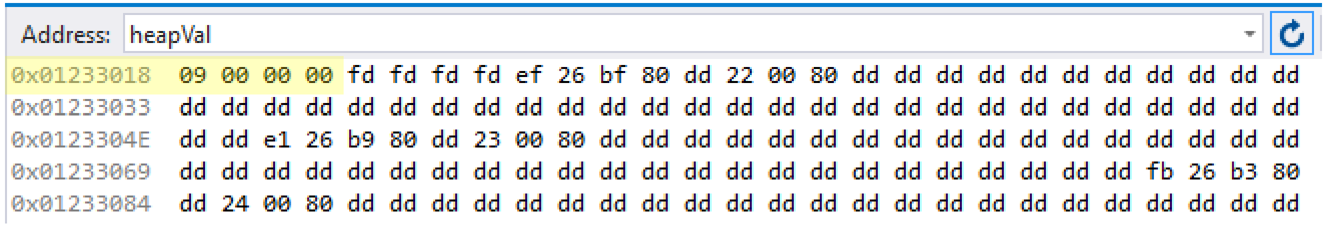


```
0x012FFC98  9a 99 99 99 99 99 1b 40 66 66 66 66 66 e6 46 40 cc cc cc cc cc cc cc cc 01 00 00
0x012FFCB3  00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 cc cc cc cc cc cc cc cc 09 00
0x012FFCCE  00 00 cc cc cc cc b9 e7 18 1d 40 fd 2f 01 52 64 ff 00 ff ff ff ff f8 fc 2f 01 be
0x012FFCE9  23 ff 00 01 00 00 00 70 6c 5a 01 88 57 5b 01 50 fd 2f 01 0a 22 ff 00 0d e6 18 1d
0x012FFD04  5f 10 ff 00 5f 10 ff 00 00 40 1e 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFD1F  00 00 00 00 00 00 00 00 00 00 00 00 00 14 b7 ff 00 20 b7 ff 00 00 00 00 00 00 fd
0x012FFD3A  2f 01 00 00 00 00 bc fd 2f 01 00 45 ff 00 b5 ba c8 1c 00 00 00 00 58 fd 2f 01 9d
0x012FFD55  20 ff 00 60 fd 2f 01 d8 23 ff 00 70 fd 2f 01 29 fa 16 77 00 40 1e 01 10 fa 16 77
0x012FFD70  cc fd 2f 01 5e 7b b1 77 00 40 1e 01 21 6d c9 e3 00 00 00 00 00 00 00 00 00 40 1e
0x012FFD8B  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFDA6  00 00 00 00 00 00 00 00 00 00 00 00 00 7c fd 2f 01 00 00 00 00 d4 fd 2f 01 00
0x012FFDC1  ae b2 77 2d 5c 5d 95 00 00 00 00 dc fd 2f 01 2e 7b b1 77 ff ff ff ff 89 8c b3 77
0x012FFDDC  00 00 00 00 00 00 00 00 5f 10 ff 00 00 40 1e 01 00 00 00 00 00 00 00 00 00 00 00
0x012FFDF7  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE12  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE2D  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE48  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE63  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE7E  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFE99  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFEB4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFECF  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFEEA  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF05  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF20  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF3B  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF56  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF71  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFF8C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFFA7  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFFC2  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFFDD  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x012FFFF8  00 00 00 00 00 00 00 00 08 c2 01 00 64 00 00 00 00 00 00 00 00 00 00 00 00 c0 01
0x01300013  00 08 02 00 00 cc 00 32 00 78 00 00 00 00 00 00 00 00 00 00 00 00 c0 01 00 08 02
0x0130002E  00 00 00 00 f7 00 00 00 00 00 00 00 00 00 00 00 00 00 43 3a 5c 50 72 6f 67 72 61
0x01300049  6d 44 61 74 61 5c 53 79 6d 61 6e 74 65 63 5c 53 79 6d 61 6e 74 65 63 20 45 6e 64
0x01300064  70 6f 69 6e 74 20 50 72 6f 74 65 63 74 69 6f 6e 5c 31 34 2e 33 2e 31 31 36 39 2e
0x0130007F  30 31 30 30 2e 31 30 35 5c 44 61 74 61 5c 44 65 66 69 6e 69 74 69 6f 6e 73 5c 42
0x0130009A  41 53 48 44 65 66 73 5c 32 30 32 32 31 31 30 33 2e 30 32 31 5c 55 4d 45 6e 67 78
0x013000B5  38 36 2e 64 6c 6c 00 03 00 00 80 00 00 00 00 60 64 4e 7c 00 00 00 00 bc 00 32 00
0x013000D0  00 00 00 00 00 00 00 00 40 00 32 00 c4 00 32 00 04 c3 01 00 00 00 00 00 00 00 00
```

Stack vs Heap Memory

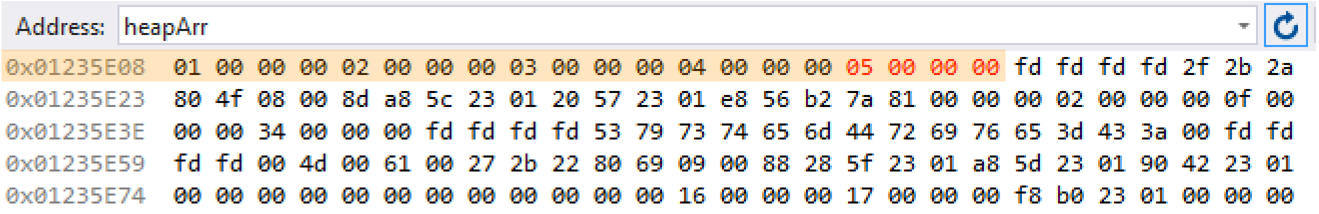


```cpp
#include "stdafx.h"
#include <stdio.h>
#include <iostream>

struct Point
{
    int m_x;
    int m_y;

    Point()
        : m_x(6), m_y(45)
    {

    }
};

int main()
{
    int stackVal = 9; //stack allocation
    int stackArr[5]; //stack allocation
    stackArr[0] = 1;
    stackArr[1] = 2;
    stackArr[2] = 3;
    stackArr[3] = 4;
    stackArr[4] = 5;

    Point stackPoint; //stack allocation

    int* heapVal = new int(9); //Heap allocation
    int* heapArr = new int[5]; //heap allocation
    heapArr[0] = 1;
    heapArr[1] = 2;
    heapArr[2] = 3;
    heapArr[3] = 4;
    heapArr[4] = 5;

    Point* heapPoint = new Point(); //heap allocation

    std::cin.get();
}
```



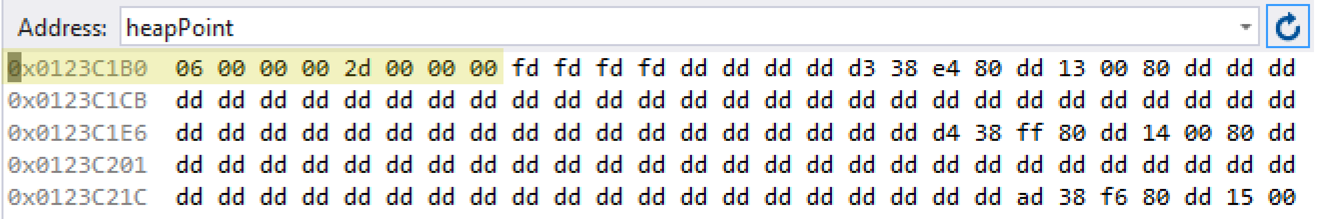
Address: heapVal

```
0x01233018  09 00 00 00 fd fd fd fd ef 26 bf 80 dd 22 00 80 dd dd dd dd dd dd dd dd dd dd dd
0x01233033  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd
0x0123304E  dd dd e1 26 b9 80 dd 23 00 80 dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd
0x01233069  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd fb 26 b3 80
0x01233084  dd 24 00 80 dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd
```



Address: heapArr

```
0x01235E08  01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 fd fd fd fd 2f 2b 2a
0x01235E23  80 4f 08 00 8d a8 5c 23 01 20 57 23 01 e8 56 b2 7a 81 00 00 00 02 00 00 00 0f 00
0x01235E3E  00 00 34 00 00 00 fd fd fd fd 53 79 73 74 65 6d 44 72 69 76 65 3d 43 3a 00 fd fd
0x01235E59  fd fd 00 4d 00 61 00 27 2b 22 80 69 09 00 88 28 5f 23 01 a8 5d 23 01 90 42 23 01
0x01235E74  00 00 00 00 00 00 00 00 00 00 00 00 16 00 00 00 17 00 00 00 f8 b0 23 01 00 00 00
```

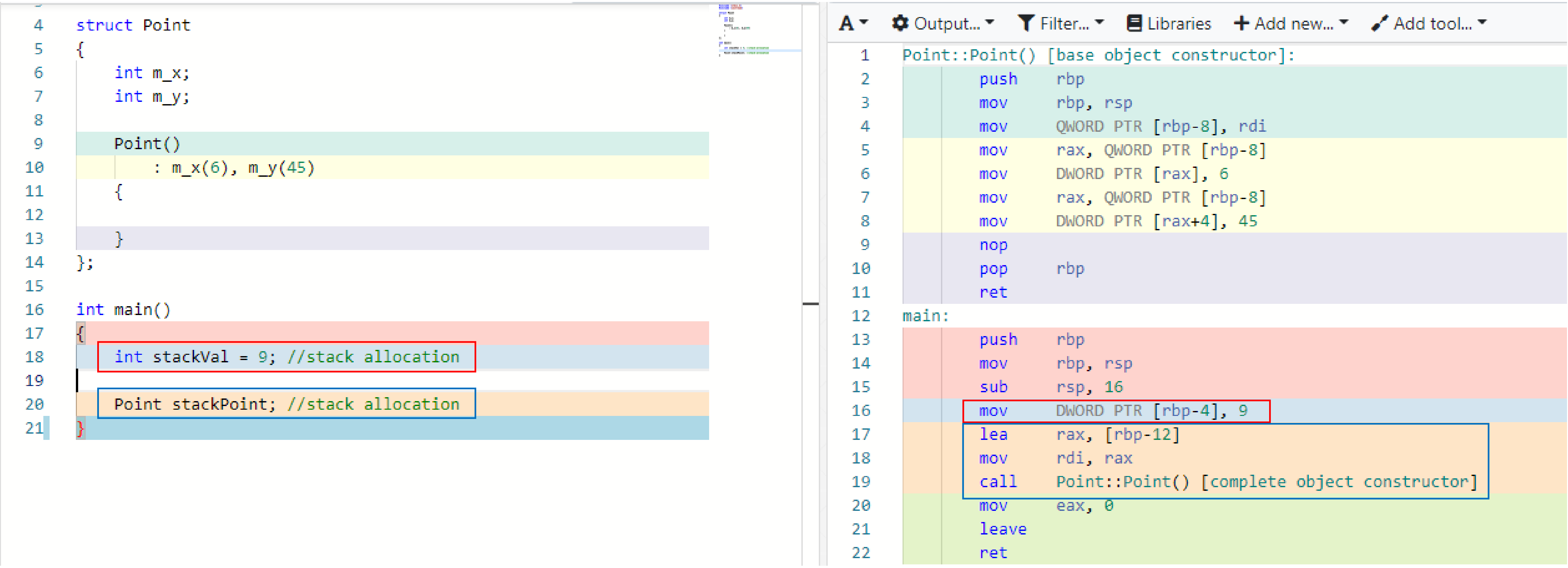


Address: heapPoint

```
0x0123C1B0  06 00 00 00 2d 00 00 00 fd fd fd fd dd dd dd dd d3 38 e4 80 dd 13 00 80 dd dd dd
0x0123C1CB  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd
0x0123C1E6  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd d4 38 ff 80 dd 14 00 80 dd
0x0123C201  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd
0x0123C21C  dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd dd ad 38 f6 80 dd 15 00
```

LSEG

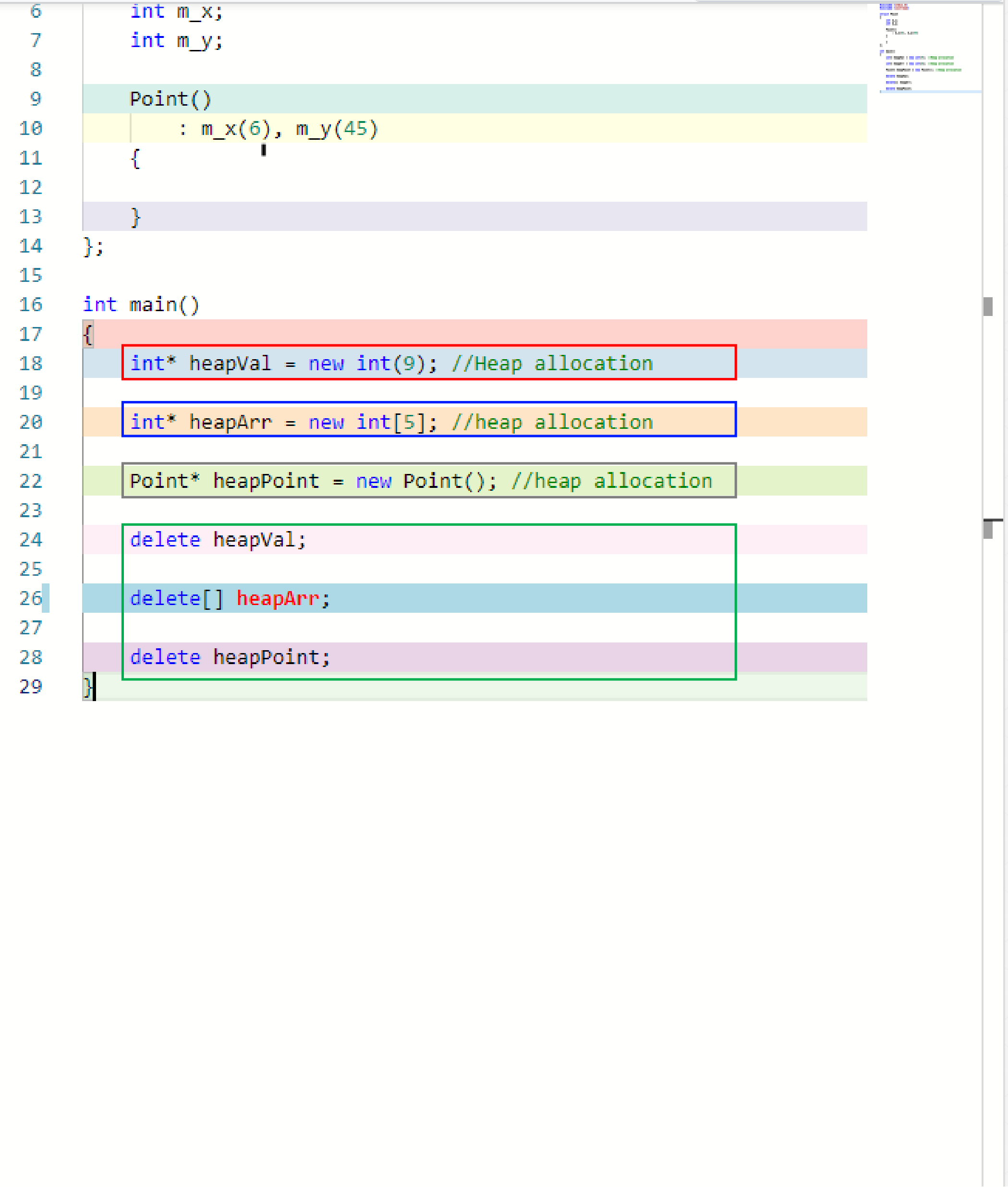
Stack vs Heap Memory



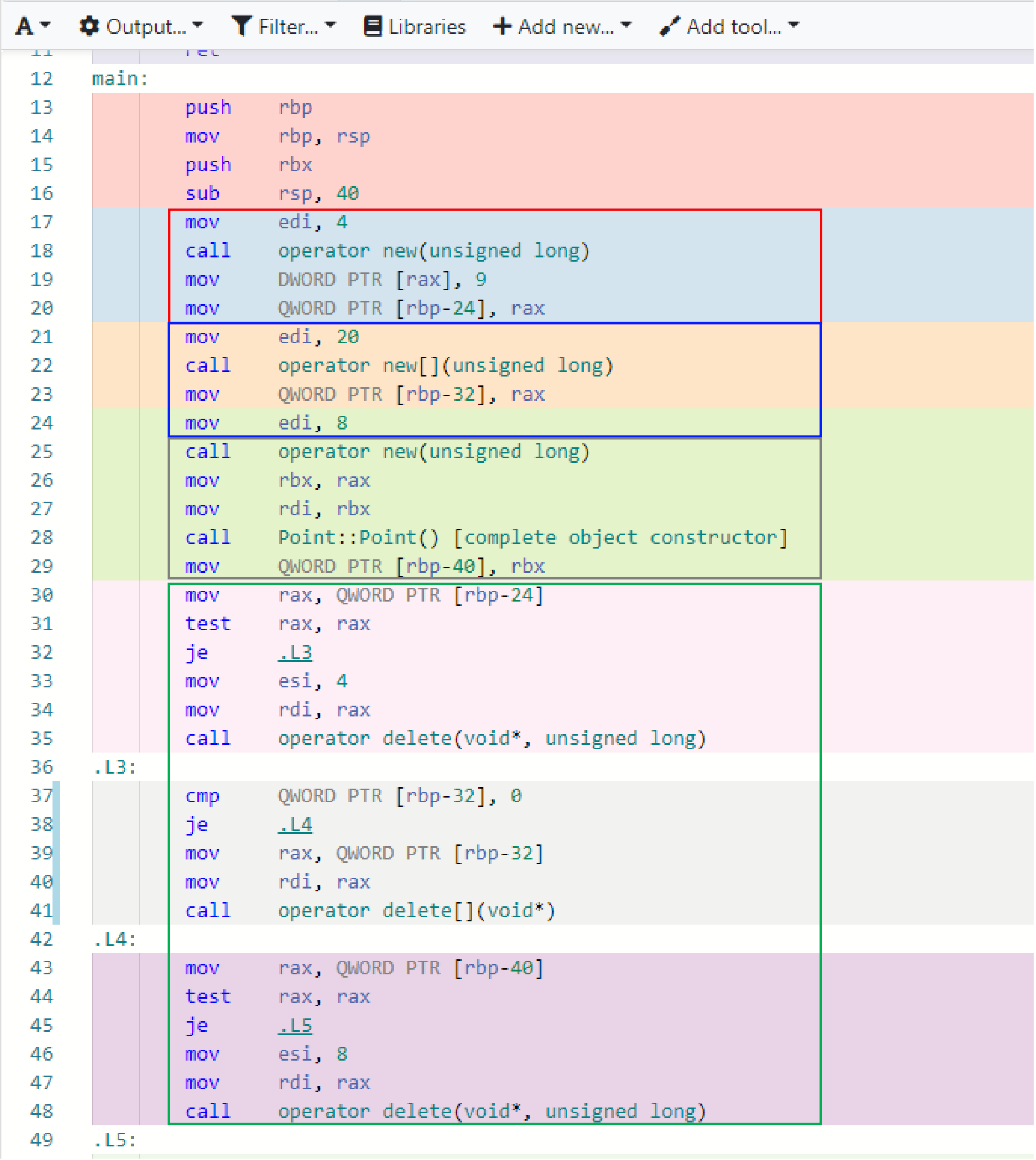
```cpp
struct Point
{
    int m_x;
    int m_y;

    Point()
        : m_x(6), m_y(45)
    {

    }
};

int main()
{
    int stackVal = 9; //stack allocation

    Point stackPoint; //stack allocation
}
```

```asm
Point::Point() [base object constructor]:
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     DWORD PTR [rax], 6
        mov     rax, QWORD PTR [rbp-8]
        mov     DWORD PTR [rax+4], 45
        nop
        pop     rbp
        ret
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 9
        lea     rax, [rbp-12]
        mov     rdi, rax
        call    Point::Point() [complete object constructor]
        mov     eax, 0
        leave
        ret
```

Stack vs Heap Memory



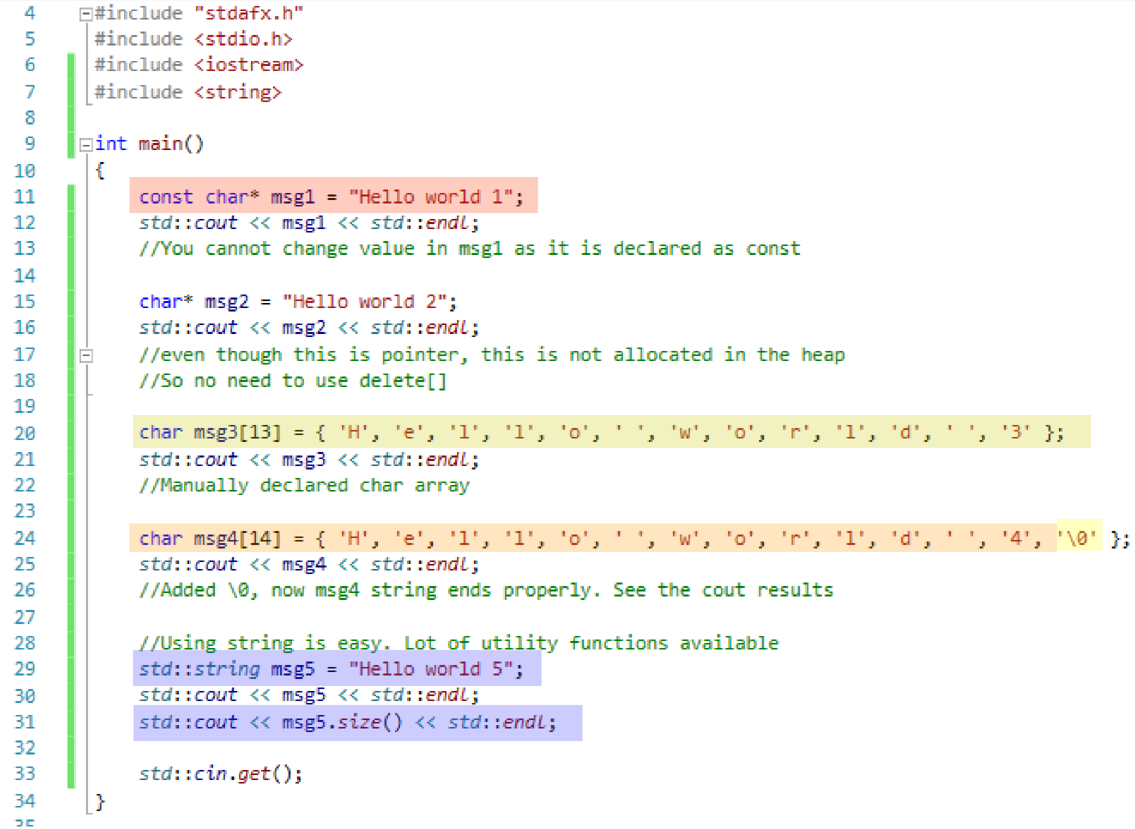
```cpp
    int m_x;
    int m_y;

    Point()
        : m_x(6), m_y(45)
    {

    }
};

int main()
{
    int* heapVal = new int(9); //Heap allocation

    int* heapArr = new int[5]; //heap allocation

    Point* heapPoint = new Point(); //heap allocation

    delete heapVal;

    delete[] heapArr;

    delete heapPoint;
}
```



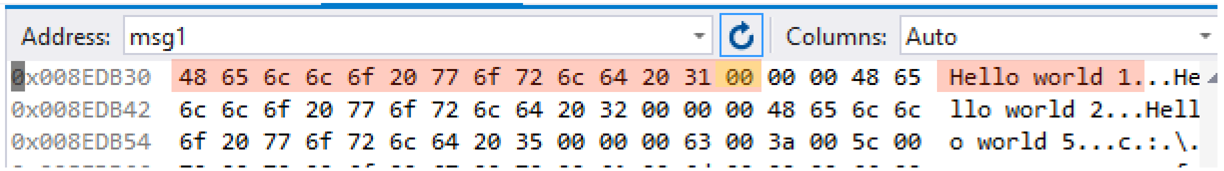
```asm
        ret
main:
        push    rbp
        mov     rbp, rsp
        push    rbx
        sub     rsp, 40
        mov     edi, 4
        call    operator new(unsigned long)
        mov     DWORD PTR [rax], 9
        mov     QWORD PTR [rbp-24], rax
        mov     edi, 20
        call    operator new[](unsigned long)
        mov     QWORD PTR [rbp-32], rax
        mov     edi, 8
        call    operator new(unsigned long)
        mov     rbx, rax
        mov     rdi, rbx
        call    Point::Point() [complete object constructor]
        mov     QWORD PTR [rbp-40], rbx
        mov     rax, QWORD PTR [rbp-24]
        test    rax, rax
        je      .L3
        mov     esi, 4
        mov     rdi, rax
        call    operator delete(void*, unsigned long)
.L3:
        cmp     QWORD PTR [rbp-32], 0
        je      .L4
        mov     rax, QWORD PTR [rbp-32]
        mov     rdi, rax
        call    operator delete[](void*)
.L4:
        mov     rax, QWORD PTR [rbp-40]
        test    rax, rax
        je      .L5
        mov     esi, 8
        mov     rdi, rax
        call    operator delete(void*, unsigned long)
.L5:
```

How Strings Work

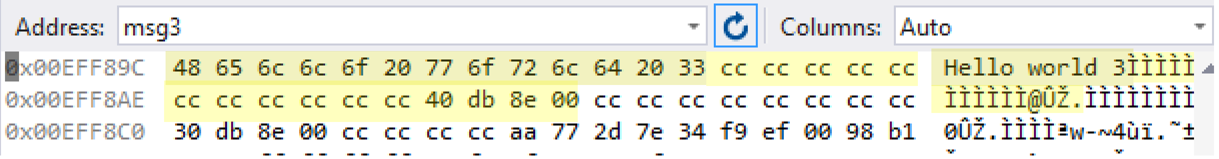
- Array of characters
- Instantiation of the basic_string class template that uses “char”
- Handles bytes independently of the encoding used



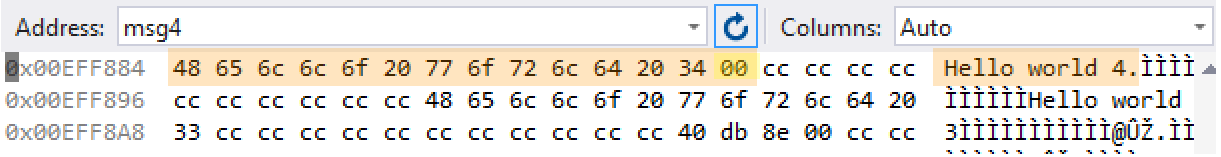
```cpp
#include "stdafx.h"
#include <stdio.h>
#include <iostream>
#include <string>

int main()
{
    const char* msg1 = "Hello world 1";
    std::cout << msg1 << std::endl;
    //You cannot change value in msg1 as it is declared as const

    char* msg2 = "Hello world 2";
    std::cout << msg2 << std::endl;
    //even though this is pointer, this is not allocated in the heap
    //So no need to use delete[]

    char msg3[13] = { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', ' ', '3' };
    std::cout << msg3 << std::endl;
    //Manually declared char array

    char msg4[14] = { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', ' ', '4', '\0' };
    std::cout << msg4 << std::endl;
    //Added \0, now msg4 string ends properly. See the cout results

    //Using string is easy. Lot of utility functions available
    std::string msg5 = "Hello world 5";
    std::cout << msg5 << std::endl;
    std::cout << msg5.size() << std::endl;

    std::cin.get();
}
```



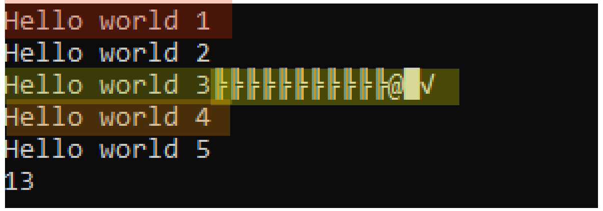
```
0x008EDB30  48 65 6c 6c 6f 20 77 6f 72 6c 64 20 31 00 00 00 48 65  Hello world 1...He
0x008EDB42  6c 6c 6f 20 77 6f 72 6c 64 20 32 00 00 00 48 65 6c 6c  llo world 2...Hell
0x008EDB54  6f 20 77 6f 72 6c 64 20 35 00 00 00 63 00 3a 00 5c 00  o world 5...c.:.\.
```



```
0x00EFF89C  48 65 6c 6c 6f 20 77 6f 72 6c 64 20 33 cc cc cc cc cc  Hello world 3ÌÌÌÌÌ
0x00EFF8AE  cc cc cc cc cc cc 40 db 8e 00 cc cc cc cc cc cc cc cc  ÌÌÌÌÌÌ@ÛŽ.ÌÌÌÌÌÌÌÌ
0x00EFF8C0  30 db 8e 00 cc cc cc cc aa 77 2d 7e 34 f9 ef 00 98 b1  0ÛŽ.ÌÌÌÌªw-~4ùï.˜±
```

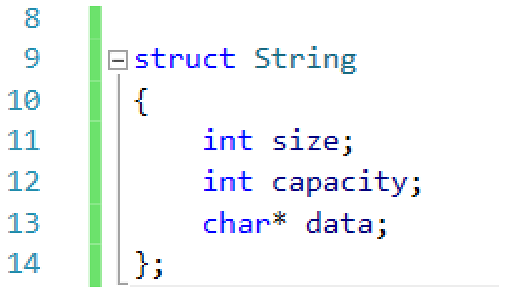


```
0x00EFF884  48 65 6c 6c 6f 20 77 6f 72 6c 64 20 34 00 cc cc cc cc  Hello world 4.ÌÌÌÌ
0x00EFF896  cc cc cc cc cc cc 48 65 6c 6c 6f 20 77 6f 72 6c 64 20  ÌÌÌÌÌÌHello world 
0x00EFF8A8  33 cc cc cc cc cc cc cc cc cc cc cc 40 db 8e 00 cc cc  3ÌÌÌÌÌÌÌÌÌÌÌ@ÛŽ.ÌÌ
```



```
Hello world 1
Hello world 2
Hello world 3╠╠╠╠╠╠╠╠╠╠╠@█√
Hello world 4
Hello world 5
13
```

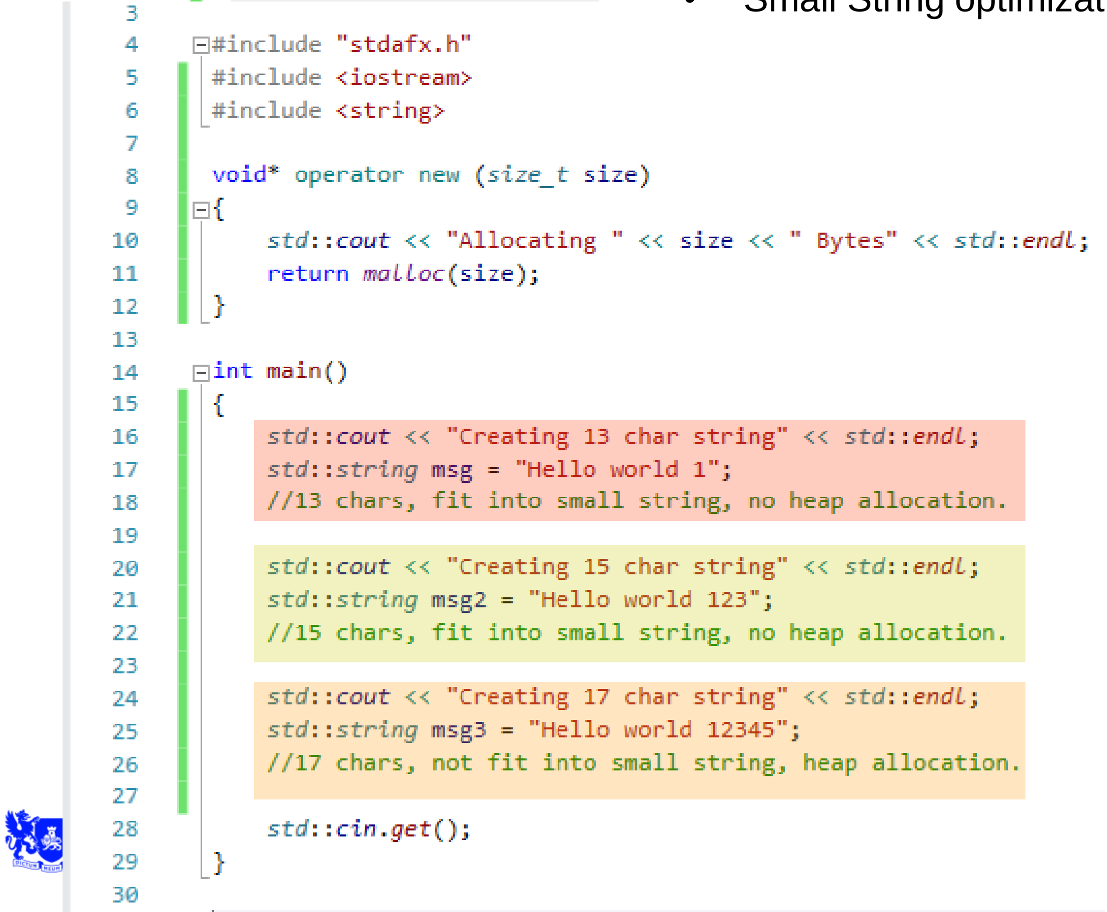
How Strings Work



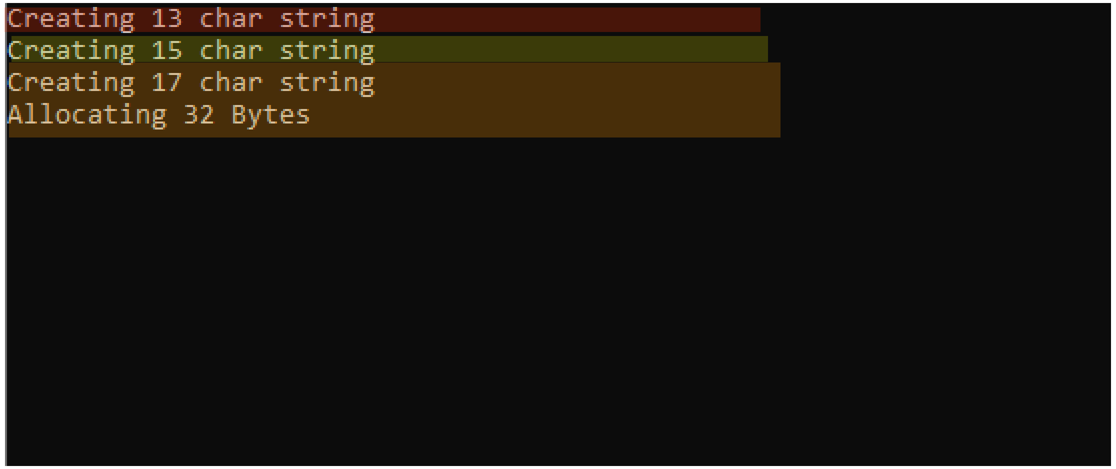
```
struct String
{
    int size;
    int capacity;
    char* data;
};
```

Who think this is a good representation of a string?

- All data allocated in heap
- What about empty string
- Global empty string (1 byte)
- Small String optimization in std::string



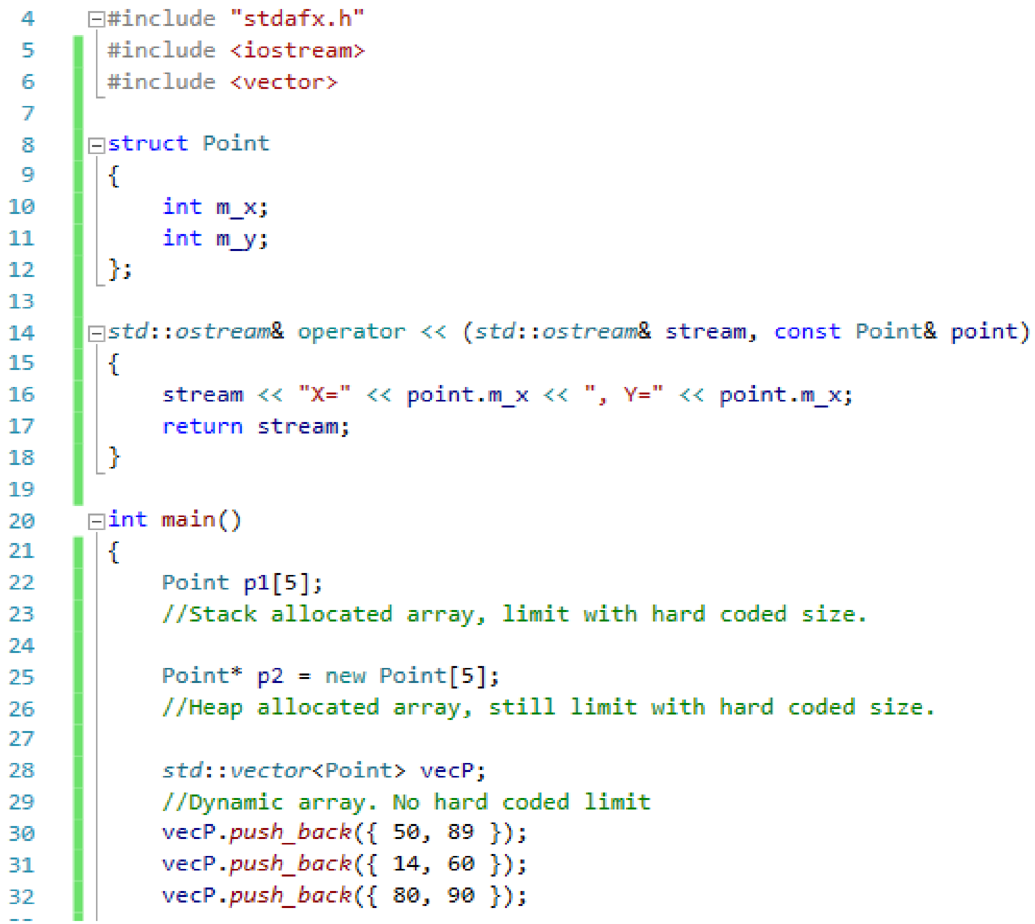
```

#include "stdafx.h"
#include <iostream>
#include <string>

void* operator new (size_t size)
{
    std::cout << "Allocating " << size << " Bytes" << std::endl;
    return malloc(size);
}

int main()
{
    std::cout << "Creating 13 char string" << std::endl;
    std::string msg = "Hello world 1";
    //13 chars, fit into small string, no heap allocation.

    std::cout << "Creating 15 char string" << std::endl;
    std::string msg2 = "Hello world 123";
    //15 chars, fit into small string, no heap allocation.

    std::cout << "Creating 17 char string" << std::endl;
    std::string msg3 = "Hello world 12345";
    //17 chars, not fit into small string, heap allocation.

    std::cin.get();
}
```



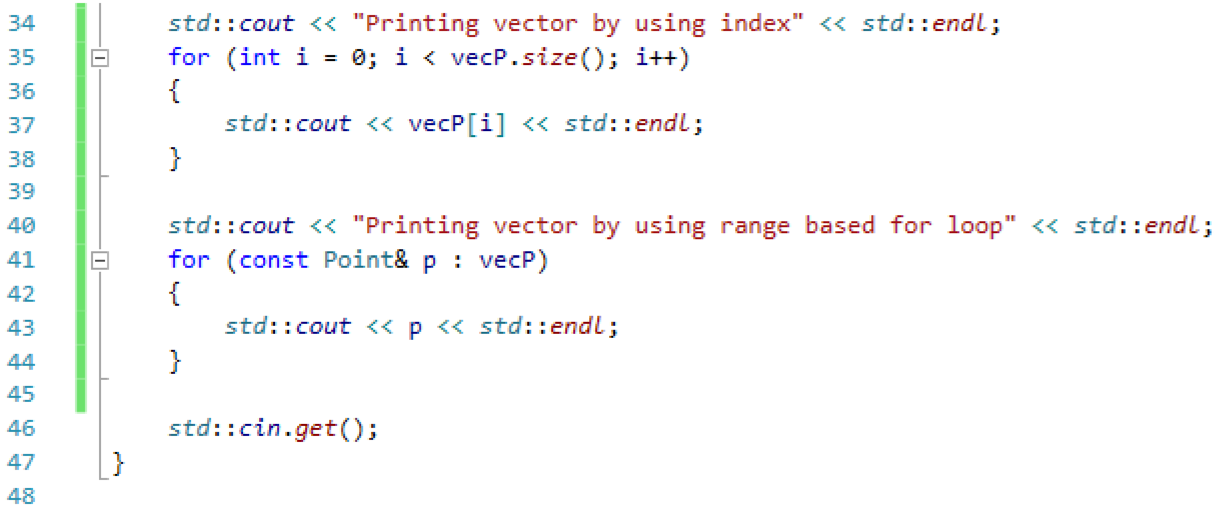
```
Creating 13 char string
Creating 15 char string
Creating 17 char string
Allocating 32 Bytes
```

Dynamic Arrays (Vectors)

- The elements are stored contiguously
- Can access elements through iterators and index
- By default insert/remove at the end



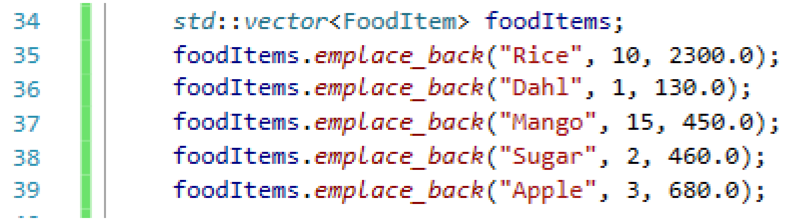
```
#include "stdafx.h"
#include <iostream>
#include <vector>

struct Point
{
    int m_x;
    int m_y;
};

std::ostream& operator << (std::ostream& stream, const Point& point)
{
    stream << "X=" << point.m_x << ", Y=" << point.m_x;
    return stream;
}

int main()
{
    Point p1[5];
    //Stack allocated array, limit with hard coded size.

    Point* p2 = new Point[5];
    //Heap allocated array, still limit with hard coded size.

    std::vector<Point> vecP;
    //Dynamic array. No hard coded limit
    vecP.push_back({ 50, 89 });
    vecP.push_back({ 14, 60 });
    vecP.push_back({ 80, 90 });
```



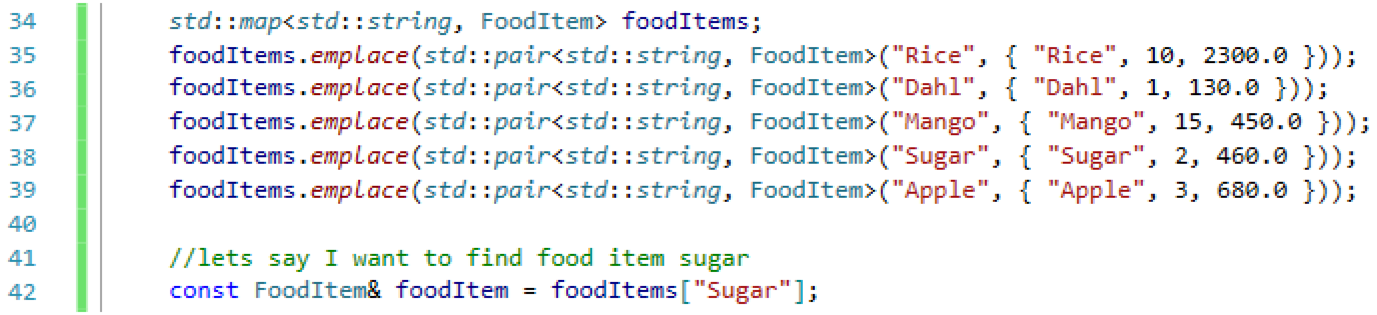
```
    std::cout << "Printing vector by using index" << std::endl;
    for (int i = 0; i < vecP.size(); i++)
    {
        std::cout << vecP[i] << std::endl;
    }

    std::cout << "Printing vector by using range based for loop" << std::endl;
    for (const Point& p : vecP)
    {
        std::cout << p << std::endl;
    }

    std::cin.get();
}
```

map/unordered_map

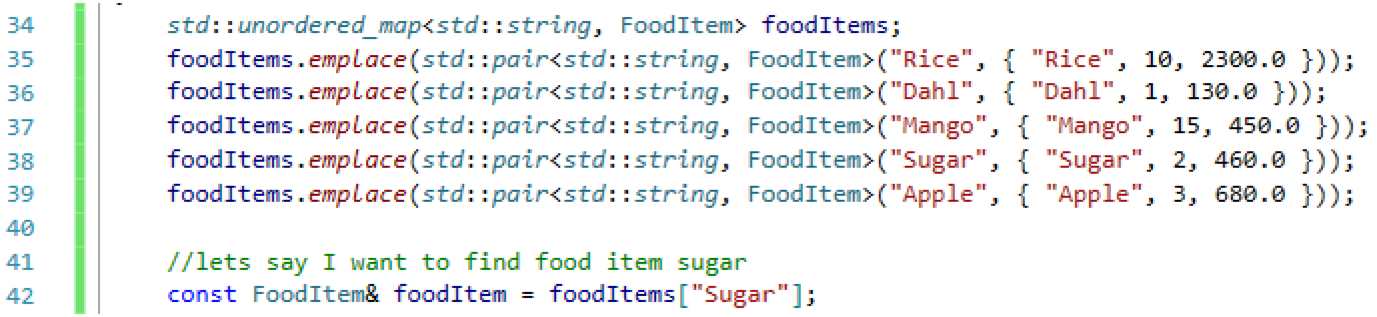
- std::map is a sorted associative container
- std::unordered_map is also an associative container, but not sorted
- contains key-value pairs with unique keys



```
std::vector<FoodItem> foodItems;
foodItems.emplace_back("Rice", 10, 2300.0);
foodItems.emplace_back("Dahl", 1, 130.0);
foodItems.emplace_back("Mango", 15, 450.0);
foodItems.emplace_back("Sugar", 2, 460.0);
foodItems.emplace_back("Apple", 3, 680.0);
```



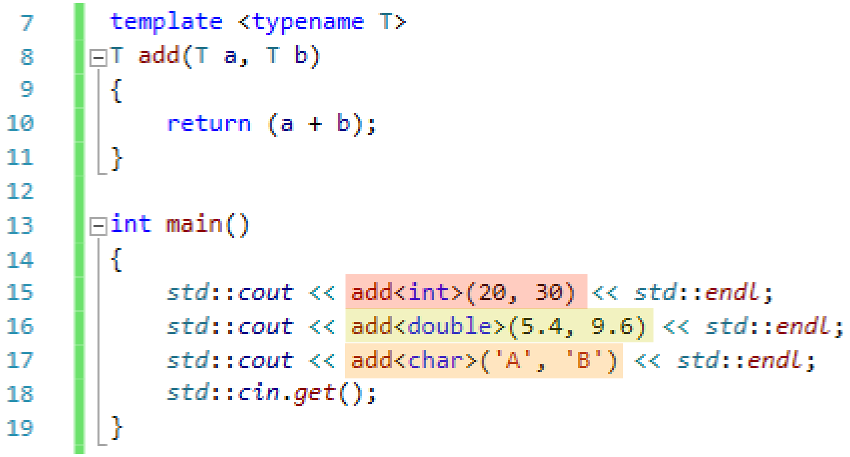
```
std::map<std::string, FoodItem> foodItems;
foodItems.emplace(std::pair<std::string, FoodItem>("Rice", { "Rice", 10, 2300.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Dahl", { "Dahl", 1, 130.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Mango", { "Mango", 15, 450.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Sugar", { "Sugar", 2, 460.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Apple", { "Apple", 3, 680.0 }));

//lets say I want to find food item sugar
const FoodItem& foodItem = foodItems["Sugar"];
```



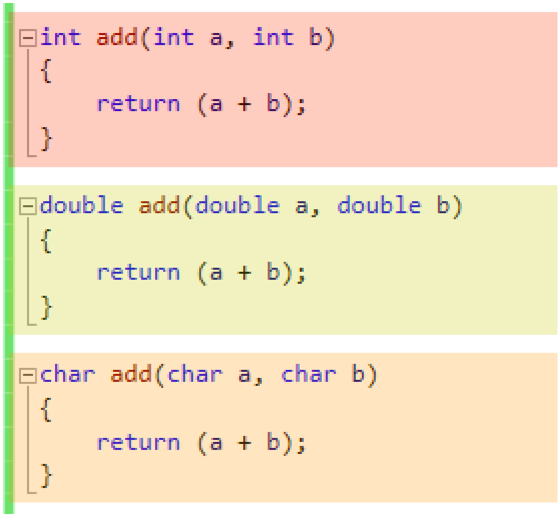
```
std::unordered_map<std::string, FoodItem> foodItems;
foodItems.emplace(std::pair<std::string, FoodItem>("Rice", { "Rice", 10, 2300.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Dahl", { "Dahl", 1, 130.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Mango", { "Mango", 15, 450.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Sugar", { "Sugar", 2, 460.0 }));
foodItems.emplace(std::pair<std::string, FoodItem>("Apple", { "Apple", 3, 680.0 }));

//lets say I want to find food item sugar
const FoodItem& foodItem = foodItems["Sugar"];
```

Templates

- Very powerful tool in C++
- Compile time code generation
- Function Templates, Class Templates

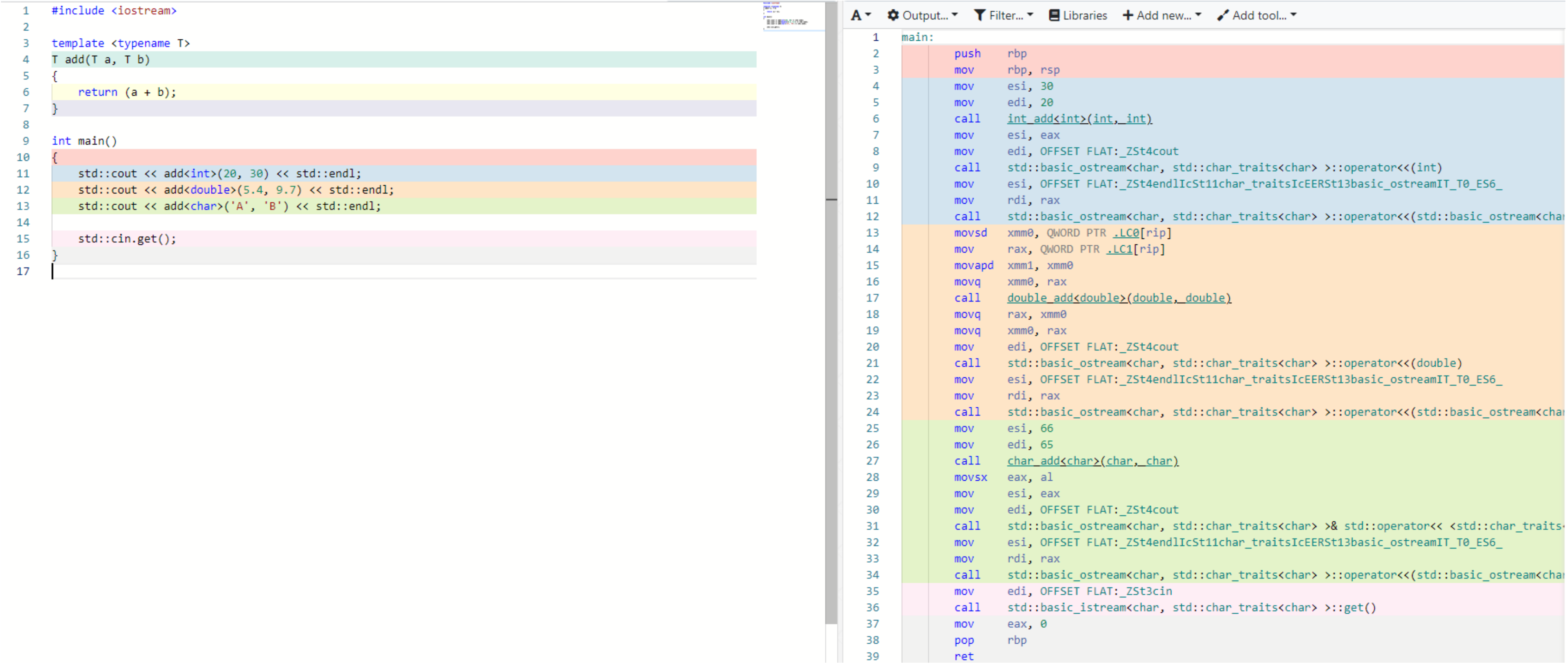


```cpp
template <typename T>
T add(T a, T b)
{
    return (a + b);
}

int main()
{
    std::cout << add<int>(20, 30) << std::endl;
    std::cout << add<double>(5.4, 9.6) << std::endl;
    std::cout << add<char>('A', 'B') << std::endl;
    std::cin.get();
}
```



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

double add(double a, double b)
{
    return (a + b);
}

char add(char a, char b)
{
    return (a + b);
}
```

Templates



```cpp
#include <iostream>

template <typename T>
T add(T a, T b)
{
    return (a + b);
}

int main()
{
    std::cout << add<int>(20, 30) << std::endl;
    std::cout << add<double>(5.4, 9.7) << std::endl;
    std::cout << add<char>('A', 'B') << std::endl;

    std::cin.get();
}
```

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     esi, 30
        mov     edi, 20
        call    int add<int>(int, int)
        mov     esi, eax
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >::operator<<(int)
        mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char
        movsd   xmm0, QWORD PTR .LC0[rip]
        mov     rax, QWORD PTR .LC1[rip]
        movapd  xmm1, xmm0
        movq    xmm0, rax
        call    double add<double>(double, double)
        movq    rax, xmm0
        movq    xmm0, rax
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >::operator<<(double)
        mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char
        mov     esi, 66
        mov     edi, 65
        call    char add<char>(char, char)
        movsx   eax, al
        mov     esi, eax
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >& std::operator<< <std::char_traits
        mov     esi, OFFSET FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >::operator<<(std::basic_ostream<char
        mov     edi, OFFSET FLAT:_ZSt3cin
        call    std::basic_istream<char, std::char_traits<char> >::get()
        mov     eax, 0
        pop     rbp
        ret
```

Smart Pointers

- Wrapper around raw pointer
- Mange new/delete automatically
- std::unique_ptr
- std::shared_ptr
- std::weak_ptr

STL Algorithms

- C++ Standard library come with pre defined algorithms
- std::find, std::find_if, std::find_if_not
- std::for_each
- std::max, std::min
- std::sort
- and many more. refer https://en.cppreference.com/w/cpp/algorithm

Boost

- Set of libraries
- Data structures and algorithms not in standard library (some)
- Web service support
- Some advanced date time support
- Advanced regex support
- Advanced string formatting support
- https://www.boost.org/