Find Largest Value in Each Row

Problem Description

- You need to find the largest value in each row of a binary tree.

Input:

```
----------------------
          1
         / \
        3   2
       / \   \
      5   3   9
----------------------
```

Output: [1, 3, 9]

Possible Solution #1

```
class Solution {
    void traverse(TreeNode* node, vector<int>& res, int level) {
        if (node == NULL) return;

        if (res.size() == level) {
            res.push_back(node->val);
        } else {
            res[level] = std::max(res[level], node->val);
        }

        traverse(node->left, res, level + 1);
        traverse(node->right, res, level + 1);
    }
public:
    vector<int> largestValues(TreeNode* root) {
        vector<int> res(0);
            traverse(root, res, 0);
            return res;
    }
};
```

Complexity: O(n)

Possible solution #2
DFS, BFS using STL stack and queue

Advantages of STL stack comparing with recursive version:

1. Using HEAP instead of program stack
2. Less overhead to consider each tree node
3. Easy to get implementation that uses queue

Comparing queue and stack solutions

- Memory for stack is O(h) – in average, height is O(log(n))
- Memory for queue is O(number of nodes with the same depth) – in average, O(n)

DFS using STL stack (code)

```cpp
typedef pair<TreeNode*, int> nodeDepth;
class Solution {
public:
    vector<int> largestValues(TreeNode* root) {
        vector<int> maxValues;
        if (root == NULL) return maxValues;
        stack<nodeDepth> tree;
        tree.push(nodeDepth(root, 0));
        while (!tree.empty()) {
            nodeDepth node = tree.top();
            tree.pop();
            unsigned int depth = node.second;
            if (maxValues.size() < depth + 1)
                maxValues.push_back(node.first->val); // new level of depth
            else
                maxValues[depth] = std::max(maxValues[depth], node.first->val);

            // consider left and right childs
            if (node.first->left != NULL)
                tree.push(nodeDepth(node.first->left, depth + 1));
            if (node.first->right != NULL)
                tree.push(nodeDepth(node.first->right, depth + 1));
        }
        return maxValues;
    }
};
```

BFS using STL queue (code)

```
typedef pair<TreeNode*, int> nodeDepth;
class Solution {
public:
    vector<int> largestValues(TreeNode* root) {
        vector<int> maxValues;
        if (root == NULL) return maxValues;
         queue<nodeDepth> tree;
        tree.push(nodeDepth(root, 0));
        while (!tree.empty())
        {
            nodeDepth node = tree.front();
            tree.pop();
            unsigned int depth = node.second;
            if (maxValues.size() < depth + 1)
                maxValues.push_back(node.first->val); // new level of depth
            else
                maxValues[depth] = std::max(maxValues[depth], node.first->val);

            // consider left and right childs
            if (node.first->left != NULL)
                tree.push(nodeDepth(node.first->left, depth + 1));
            if (node.first->right != NULL)
                tree.push(nodeDepth(node.first->right, depth + 1));
        }
        return maxValues;
    }
};
```

Possible Solution #3

```
class Solution {
public:
   vector<int> largestValues(TreeNode* root) {
       int row = 0;
       vector<int> result;
       scan_node(root, row);
       for (auto it = max_map.begin(); it != max_map.end(); it++) {
           result.push_back(it->second);
       }
       return result;
   }
private:
   void scan_node(TreeNode* node, int row) {
       if (!node) return;
       auto it = max_map.find(row);
       if (it != max_map.end()) {
           it->second = node->val > it->second ? node->val : it->second;
       }
       else {
           max_map.insert(pair<int,int>(row, node->val));
       }

       scan_node(node->left, row + 1);
       scan_node(node->right, row + 1);
   }
   map<int, int> max_map;
};
```

Possible Solution #4 (Array representation)

```
class Solution {
public:
    vector<int> largestValues(TreeNode* root) {
        int depth = getDepth(root);
        size_t arraySize = getArraySize(depth);

        int *arrayTree = new int [arraySize];
        for (int i = 0; i < arraySize; ++i)
            arrayTree[i] = INT_MIN;

        dumpToArray(root, arrayTree);

        sortRowsInArray(arrayTree, depth);

        return getMaximumInRows(arrayTree, depth);
    }
private:
    int getDepth(TreeNode* root, int depth = 0) {
        if (root == nullptr)
            return depth - 1;
        int maxLeftDepth = getDepth(root->left, depth + 1);
        int maxRightDepth = getDepth(root->right, depth + 1);
        return (maxLeftDepth > maxRightDepth) ? maxLeftDepth :
maxRightDepth;
    }
    inline size_t getArraySize(int depth) { return pow(2, depth + 1) - 1; }
    inline int arrayShift(int depth) { return pow(2, depth) - 1; }
    void dumpToArray(TreeNode *root, int *arr, int depth = 0, int nodeNum = 0) {
        if (root == nullptr)
            return;
        int index = arrayShift(depth) + nodeNum;
        arr[index] = root->val;
        dumpToArray(root->left, arr, depth + 1, nodeNum*2);
        dumpToArray(root->right, arr, depth + 1, nodeNum*2 + 1);
    }
    void sortRowsInArray(int *arr, int depth) {
        for (int i = 1; i <= depth; ++i) {
            int startIndex = arrayShift(i);
            int endIndex = arrayShift(i + 1);
            sort(arr + startIndex, arr + endIndex);
        }
    }
    vector<int> getMaximumInRows(int *arr, int depth) {
        vector<int> res;
        for (int i = 0; i <= depth; ++i) {
            int index = arrayShift(i + 1) - 1;
            res.push_back(arr[index]);
        }
        return res;
    }
};
```

Singleton

Pattern evolution

Что такое

Паттерн, описывающий объект, у которого имеется единственный экземпляр

- Такая переменная доступна всегда. Время жизни глобальной переменной - от запуска программы до ее завершения.
- Предоставляет глобальный доступ, то есть, такая переменная может быть доступна из любой части программы.

- Плюсы
    - контролируемый доступ к единственному экземпляру
- Недостатки
    - нарушает Single Responsibility Principle
    - затрудняет Unit-тестирование

Evolution (1) Classic GoF realization (1994)

```
// Declaration
class Singleton {
public:
     static Singleton* Instance();
protected:
     Singleton();
private:
     static Singleton* _instance;
}

// Implementation
Singleton* Singleton::_instance = 0;
Singleton* Singleton::Instance() {
     if(_instance == 0){
                _instance = new Singleton;
     }
     return _instance;
}
```

Singleton: thread-safe implementation

```
std::mutex Singleton::m_mutex; // Declared as static in private section

Singleton* Singleton::Instance(){
    // Lock
    std::lock_guard<std::mutex> lock(m_mutex);
    if (m_instance == nullptr){
        m_instance = new Singleton;
    }
    return m_instance;
}
```

Disadvantages:

- this approach leads to ***lock contention*** (one thread is holding the lock, the others are waiting for it)
- when singleton is created, there is no need for the lock anymore.

Evolution(2) Double-Check Lock Singleton

```
class Singleton {
public:
  static Singleton * Instance();
protected:
  Singleton();
private:
  static Singleton* m_instance;
};
```

```
Singleton* Singleton::Instance() {
    Singleton* tmp = m_instance;
    // insert memory barrier
    if (tmp == NULL) {
        Lock lock;
        tmp = m_instance;
        if (tmp == NULL) {
            tmp = new Singleton;
            // insert memory barrier
            m_instance = tmp;
        }
    }
    return tmp;
}
```

Evolution(2) Double-Check Lock Singleton (C++ 11)

```
std::atomic<Singleton*> Singleton::m_instance;
std::mutex Singleton::m_mutex;

Singleton* Singleton::Instance() {
    Singleton* tmp = m_instance.load(std::memory_order_relaxed);
    std::atomic_thread_fence(std::memory_order_acquire);

    if (tmp == nullptr) {
        std::lock_guard<std::mutex> lock(m_mutex);
        tmp = m_instance.load(std::memory_order_relaxed);
        if (tmp == nullptr) {
            tmp = new Singleton;

            std::atomic_thread_fence(std::memory_order_release);
            m_instance.store(tmp, std::memory_order_relaxed);
        }
    }
    return tmp;
}
```

Evolution (3) C++11 realization (Scott Meyers)

```
class Singleton {
public:
    static Singleton& Instance() {
        static Singleton s;
        return s;
    }
private:
    Singleton() = default;
    ~Singleton() = default;

    Singleton(Singleton const&) = delete;
    Singleton& operator= (Singleton const&) = delete;
};
```

- Declare constructor, destructor as private.
- Prohibit copy constructor, operator=
- The only way to get access to the singleton is

  ```
  Singleton& instance = Singleton::Instance();
  ```
- Since C++11 this implementation is thread-safe

Evolution(4) constexpr

```
class Singleton {
public:
    static Singleton *getInstance() { return &m_instanc
}
    int getVar() const { return m_var; }
    void setVar(int var) { m_var = var; }

private:
    Singleton() : m_var(10) { }
    ~Singleton() = default;
    Singleton(Singleton const&) = delete;
    Singleton& operator=(Singleton const&) = delete;
    int m_var;
    static Singleton m_instance;
};
Singleton Singleton::m_instance;
```

```
int main()
{
    Singleton *s = Singleton::getInstance();
    s->getVar();
    s->setVar(5);
    s->getVar();
    Singleton *s2 = Singleton::getInstance();
    s2->getVar();
    return 0;
}
```

Evolution(4) constexpr

```
class Singleton {
public:
    constexpr static Singleton *getInstance() noexcept { return &m_instance; }
    constexpr int getVar() const noexcept { return m_var; }
    void setVar(int var) noexcept { m_var = var; }

private:
    constexpr Singleton() noexcept : m_var(10) { }
    ~Singleton() = default;
    Singleton(Singleton const&) = delete;
    Singleton& operator=(Singleton const&) = delete;
    int m_var;
    static Singleton m_instance;
};
Singleton Singleton::m_instance;
```

Evolution(4) constexpr: Proof

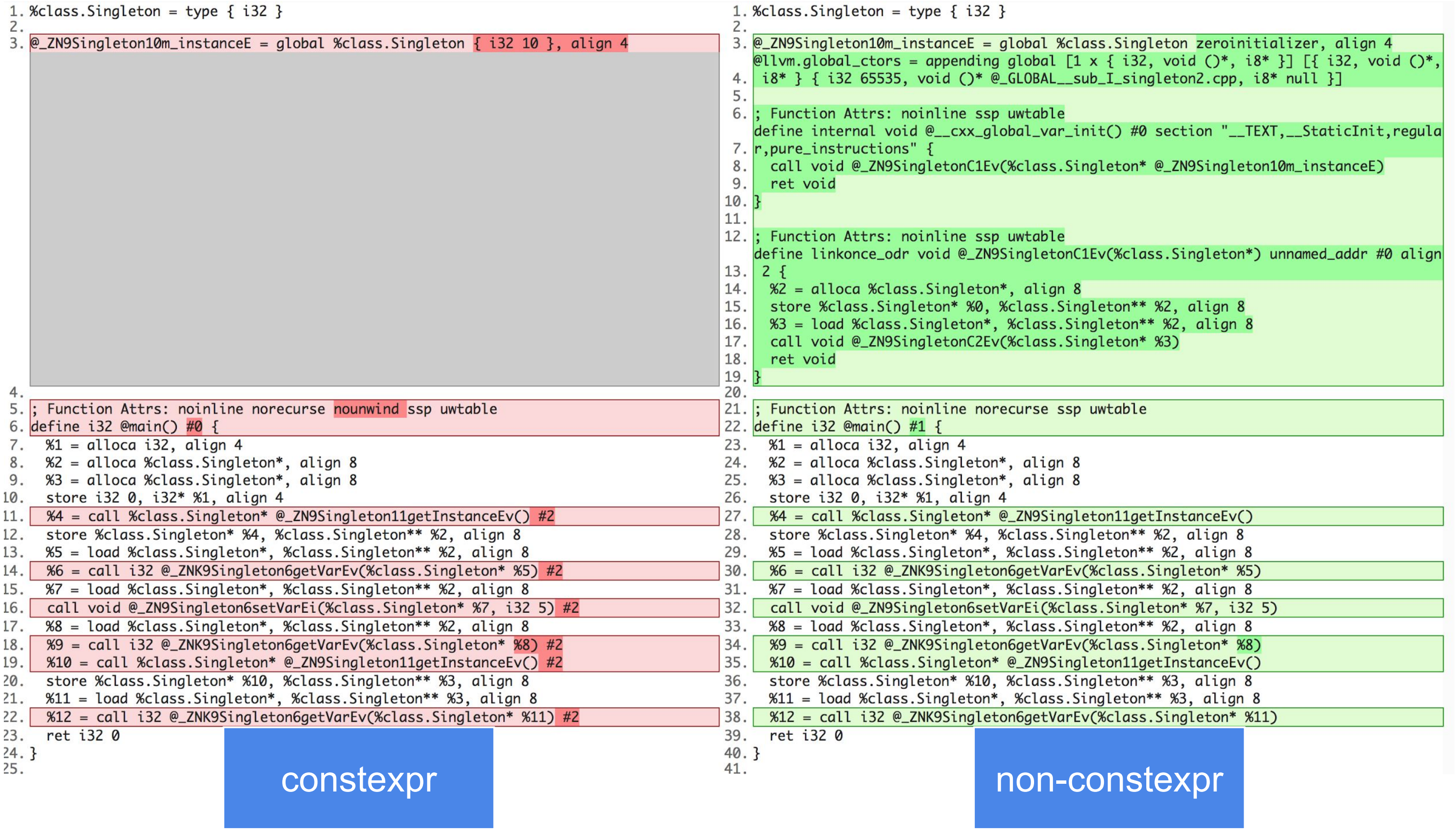
Get llvm ir for checking that singleton was initialized in compile time.

- Compile program using clang:

```
$ clang -S singleton.cpp -O0 -std=c++11 -emit-llvm -o singleton.ll
```

- Look on a differences in llvm ir code for the implementations with constexpr and without it...

Evolution(4) constexpr: Proof



```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton { i32 10 }, align 4

; Function Attrs: noinline norecurse nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5) #2
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5) #2
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8) #2
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11) #2
  ret i32 0
}
```

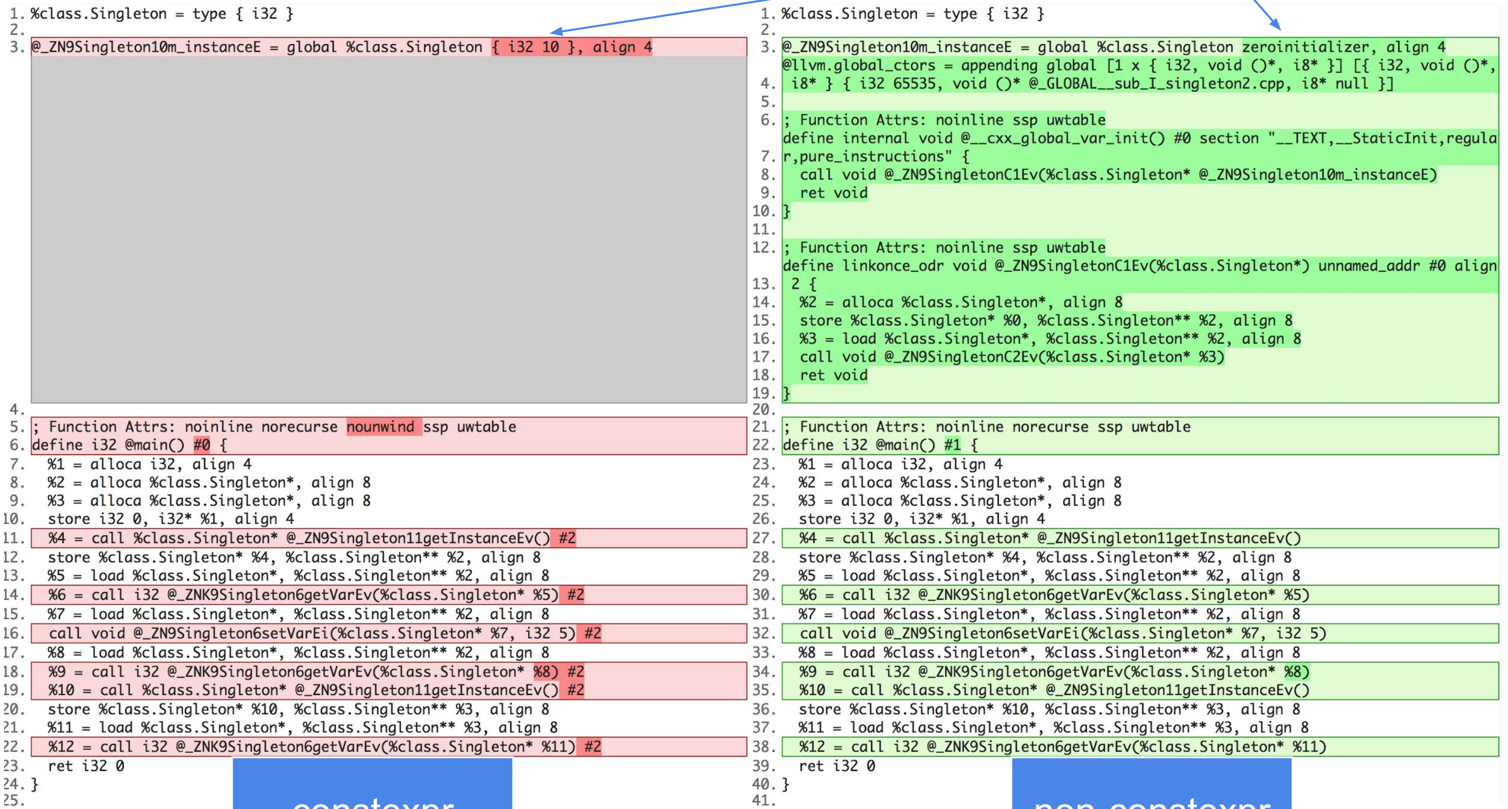
constexpr

```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton zeroinitializer, align 4
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_singleton2.cpp, i8* null }]

; Function Attrs: noinline ssp uwtable
define internal void @__cxx_global_var_init() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @_ZN9SingletonC1Ev(%class.Singleton* @_ZN9Singleton10m_instanceE)
  ret void
}

; Function Attrs: noinline ssp uwtable
define linkonce_odr void @_ZN9SingletonC1Ev(%class.Singleton*) unnamed_addr #0 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9SingletonC2Ev(%class.Singleton* %3)
  ret void
}

; Function Attrs: noinline norecurse ssp uwtable
define i32 @main() #1 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5)
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5)
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8)
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11)
  ret i32 0
}
```

non-constexpr

Evolution(4) constexpr: Proof

Initialization



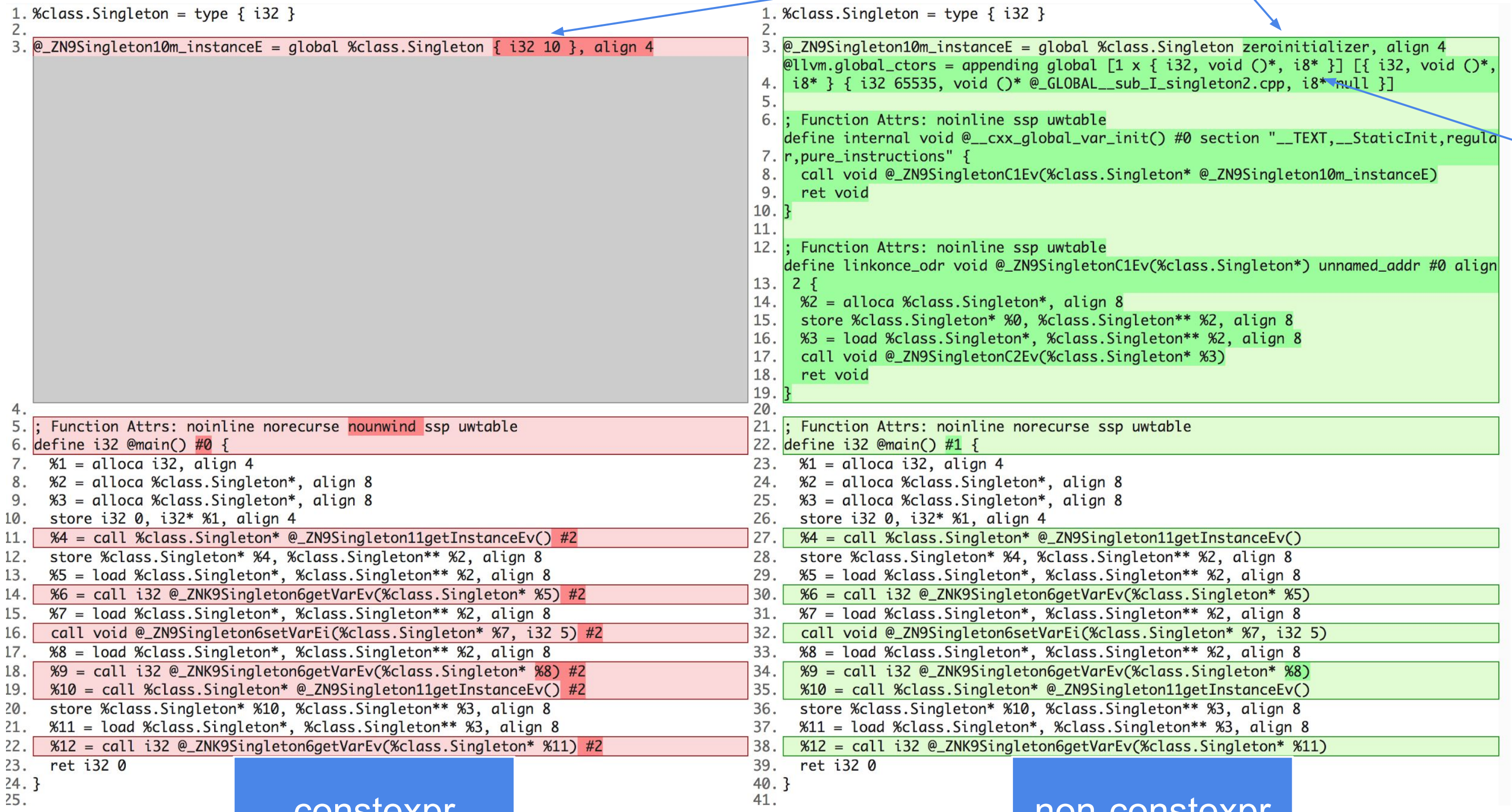
```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton { i32 10 }, align 4

; Function Attrs: noinline norecurse nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5) #2
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5) #2
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8) #2
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11) #2
  ret i32 0
}
```

```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton zeroinitializer, align 4
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_singleton2.cpp, i8* null }]

; Function Attrs: noinline ssp uwtable
define internal void @__cxx_global_var_init() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @_ZN9SingletonC1Ev(%class.Singleton* @_ZN9Singleton10m_instanceE)
  ret void
}

; Function Attrs: noinline ssp uwtable
define linkonce_odr void @_ZN9SingletonC1Ev(%class.Singleton*) unnamed_addr #0 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9SingletonC2Ev(%class.Singleton* %3)
  ret void
}

; Function Attrs: noinline norecurse ssp uwtable
define i32 @main() #1 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5)
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5)
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8)
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11)
  ret i32 0
}
```

constexpr

non-constexpr

Evolution(4) constexpr: Proof



```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton { i32 10 }, align 4

; Function Attrs: noinline norecurse nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5) #2
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5) #2
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8) #2
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11) #2
  ret i32 0
}
```

```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton zeroinitializer, align 4
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_singleton2.cpp, i8* null }]

; Function Attrs: noinline ssp uwtable
define internal void @__cxx_global_var_init() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @_ZN9SingletonC1Ev(%class.Singleton* @_ZN9Singleton10m_instanceE)
  ret void
}

; Function Attrs: noinline ssp uwtable
define linkonce_odr void @_ZN9SingletonC1Ev(%class.Singleton*) unnamed_addr #0 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9SingletonC2Ev(%class.Singleton* %3)
  ret void
}

; Function Attrs: noinline norecurse ssp uwtable
define i32 @main() #1 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5)
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5)
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8)
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11)
  ret i32 0
}
```

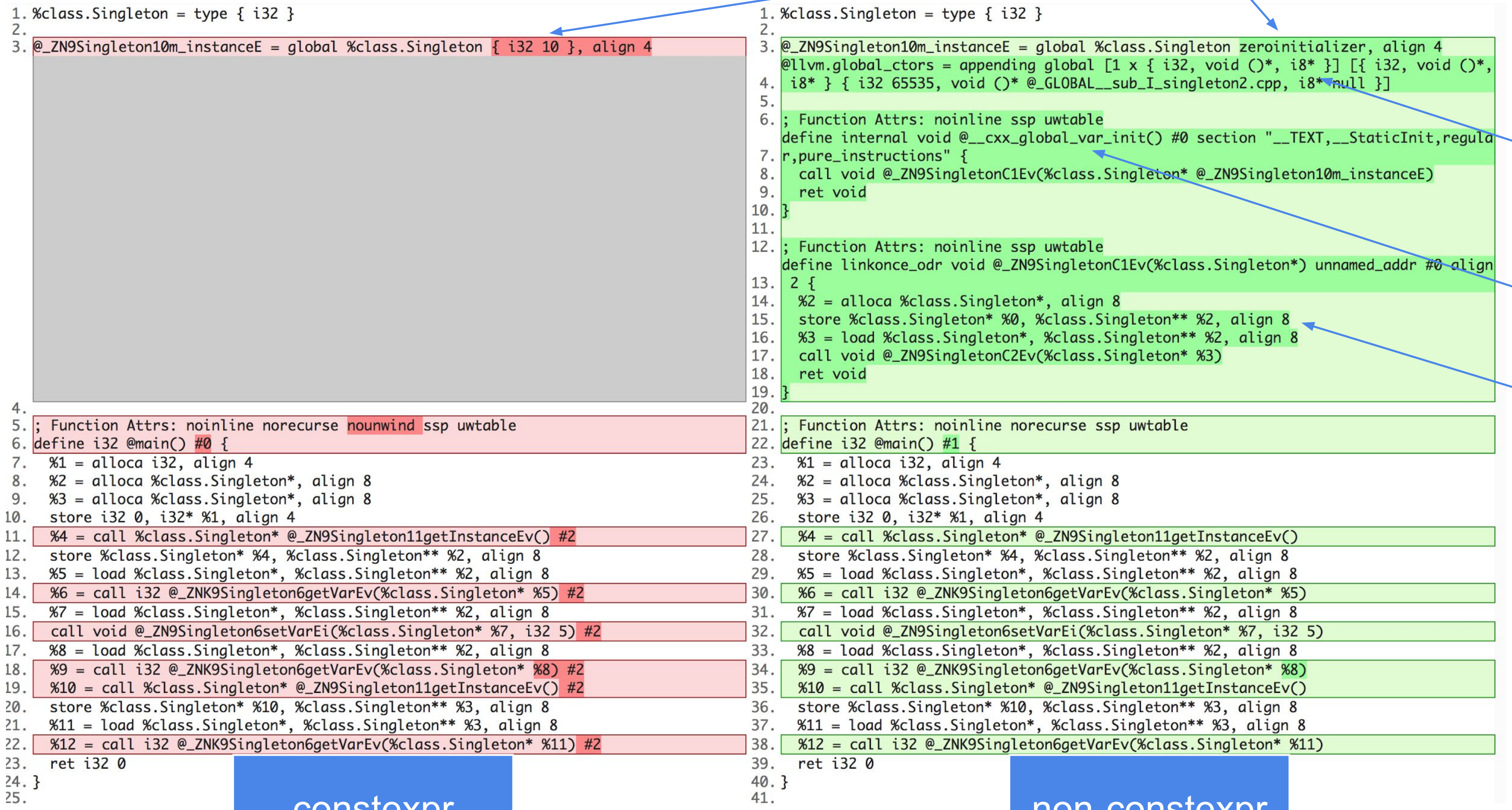
Initialization

global_ctors array

constexpr

non-constexpr

Evolution(4) constexpr: Proof



```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton { i32 10 }, align 4

; Function Attrs: noinline norecurse nounwind ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5) #2
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5) #2
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8) #2
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv() #2
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11) #2
  ret i32 0
}

```

constexpr

```
%class.Singleton = type { i32 }

@_ZN9Singleton10m_instanceE = global %class.Singleton zeroinitializer, align 4
@llvm.global_ctors = appending global [1 x { i32, void ()*, i8* }] [{ i32, void ()*, i8* } { i32 65535, void ()* @_GLOBAL__sub_I_singleton2.cpp, i8* null }]

; Function Attrs: noinline ssp uwtable
define internal void @__cxx_global_var_init() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @_ZN9SingletonC1Ev(%class.Singleton* @_ZN9Singleton10m_instanceE)
  ret void
}

; Function Attrs: noinline ssp uwtable
define linkonce_odr void @_ZN9SingletonC1Ev(%class.Singleton*) unnamed_addr #0 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9SingletonC2Ev(%class.Singleton* %3)
  ret void
}

; Function Attrs: noinline norecurse ssp uwtable
define i32 @main() #1 {
  %1 = alloca i32, align 4
  %2 = alloca %class.Singleton*, align 8
  %3 = alloca %class.Singleton*, align 8
  store i32 0, i32* %1, align 4
  %4 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %4, %class.Singleton** %2, align 8
  %5 = load %class.Singleton*, %class.Singleton** %2, align 8
  %6 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %5)
  %7 = load %class.Singleton*, %class.Singleton** %2, align 8
  call void @_ZN9Singleton6setVarEi(%class.Singleton* %7, i32 5)
  %8 = load %class.Singleton*, %class.Singleton** %2, align 8
  %9 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %8)
  %10 = call %class.Singleton* @_ZN9Singleton11getInstanceEv()
  store %class.Singleton* %10, %class.Singleton** %3, align 8
  %11 = load %class.Singleton*, %class.Singleton** %3, align 8
  %12 = call i32 @_ZNK9Singleton6getVarEv(%class.Singleton* %11)
  ret i32 0
}

```

non-constexpr

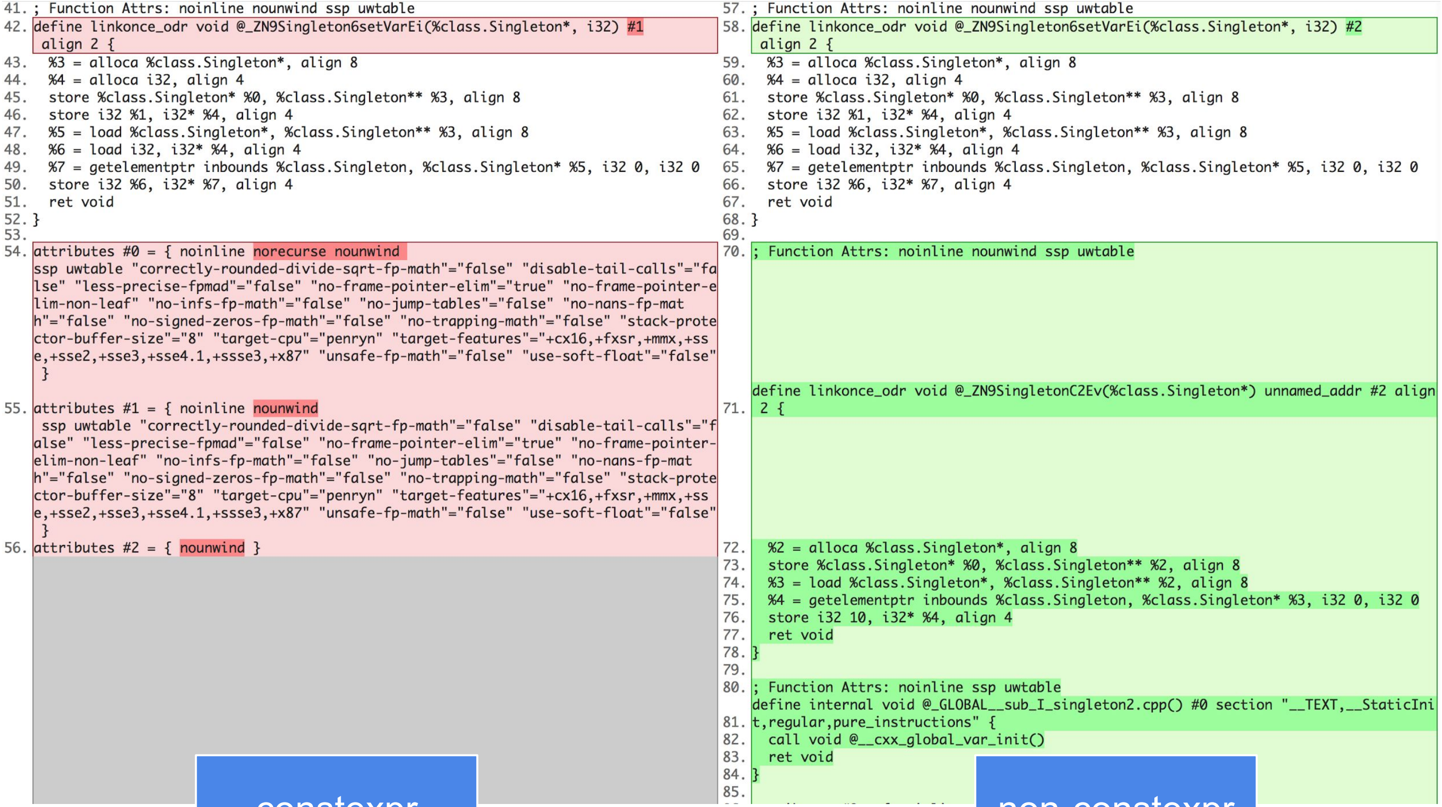
Initialization

global_ctors array

Create global variables

Constructor

Evolution(4) constexpr: Proof



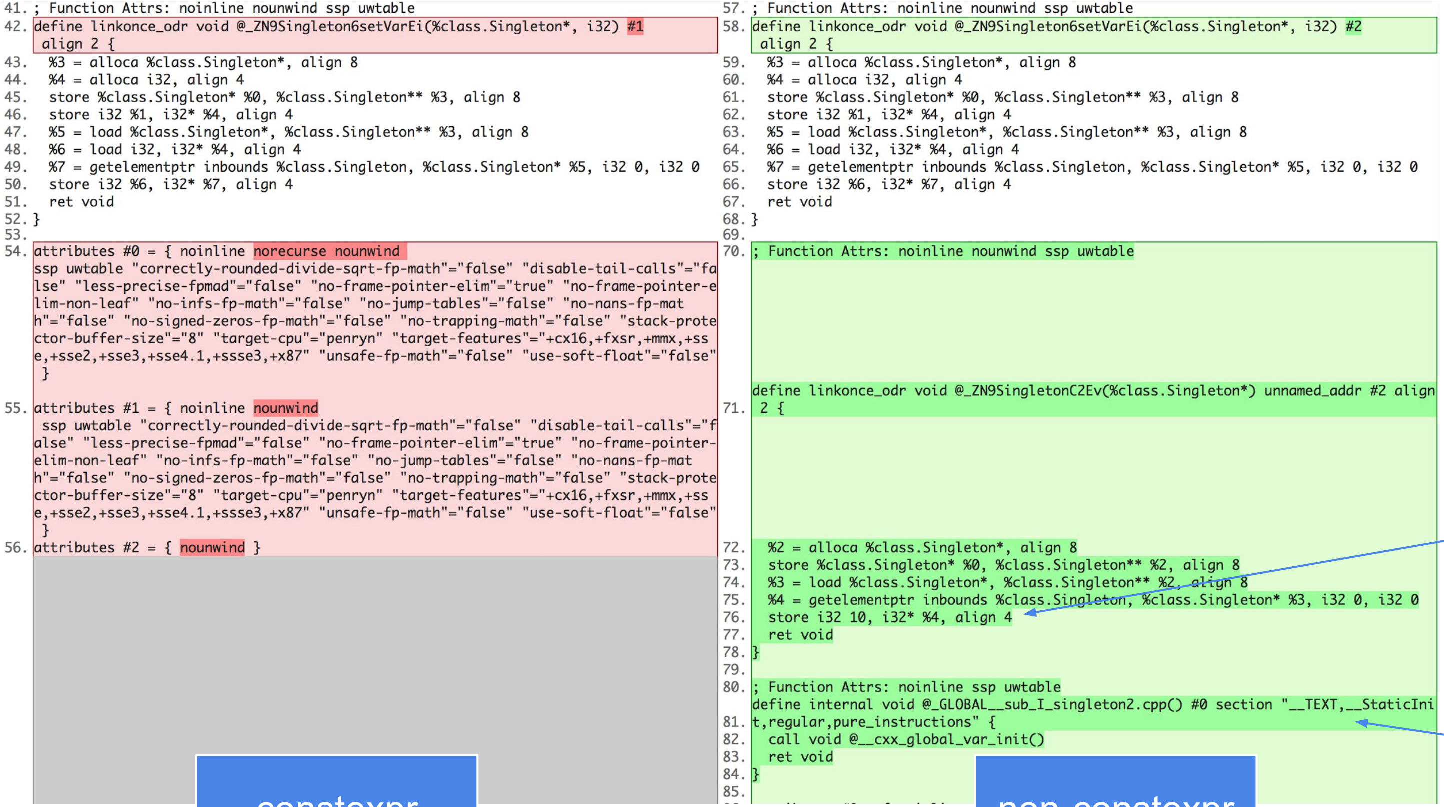
```
; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9Singleton6setVarEi(%class.Singleton*, i32) #1 align 2 {
  %3 = alloca %class.Singleton*, align 8
  %4 = alloca i32, align 4
  store %class.Singleton* %0, %class.Singleton** %3, align 8
  store i32 %1, i32* %4, align 4
  %5 = load %class.Singleton*, %class.Singleton** %3, align 8
  %6 = load i32, i32* %4, align 4
  %7 = getelementptr inbounds %class.Singleton, %class.Singleton* %5, i32 0, i32 0
  store i32 %6, i32* %7, align 4
  ret void
}

attributes #0 = { noinline norecurse nounwind ssp uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { noinline nounwind ssp uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #2 = { nounwind }
```

```
; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9Singleton6setVarEi(%class.Singleton*, i32) #2 align 2 {
  %3 = alloca %class.Singleton*, align 8
  %4 = alloca i32, align 4
  store %class.Singleton* %0, %class.Singleton** %3, align 8
  store i32 %1, i32* %4, align 4
  %5 = load %class.Singleton*, %class.Singleton** %3, align 8
  %6 = load i32, i32* %4, align 4
  %7 = getelementptr inbounds %class.Singleton, %class.Singleton* %5, i32 0, i32 0
  store i32 %6, i32* %7, align 4
  ret void
}

; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9SingletonC2Ev(%class.Singleton*) unnamed_addr #2 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  %4 = getelementptr inbounds %class.Singleton, %class.Singleton* %3, i32 0, i32 0
  store i32 10, i32* %4, align 4
  ret void
}

; Function Attrs: noinline ssp uwtable
define internal void @_GLOBAL__sub_I_singleton2.cpp() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @__cxx_global_var_init()
  ret void
}

```

constexpr

non-constexpr

Evolution(4) constexpr: Proof



```
; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9Singleton6setVarEi(%class.Singleton*, i32) #1 align 2 {
  %3 = alloca %class.Singleton*, align 8
  %4 = alloca i32, align 4
  store %class.Singleton* %0, %class.Singleton** %3, align 8
  store i32 %1, i32* %4, align 4
  %5 = load %class.Singleton*, %class.Singleton** %3, align 8
  %6 = load i32, i32* %4, align 4
  %7 = getelementptr inbounds %class.Singleton, %class.Singleton* %5, i32 0, i32 0
  store i32 %6, i32* %7, align 4
  ret void
}

attributes #0 = { noinline norecurse nounwind ssp uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }

attributes #1 = { noinline nounwind ssp uwtable "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="penryn" "target-features"="+cx16,+fxsr,+mmx,+sse,+sse2,+sse3,+sse4.1,+ssse3,+x87" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #2 = { nounwind }
```

constexpr

```
; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9Singleton6setVarEi(%class.Singleton*, i32) #2 align 2 {
  %3 = alloca %class.Singleton*, align 8
  %4 = alloca i32, align 4
  store %class.Singleton* %0, %class.Singleton** %3, align 8
  store i32 %1, i32* %4, align 4
  %5 = load %class.Singleton*, %class.Singleton** %3, align 8
  %6 = load i32, i32* %4, align 4
  %7 = getelementptr inbounds %class.Singleton, %class.Singleton* %5, i32 0, i32 0
  store i32 %6, i32* %7, align 4
  ret void
}

; Function Attrs: noinline nounwind ssp uwtable
define linkonce_odr void @_ZN9SingletonC2Ev(%class.Singleton*) unnamed_addr #2 align 2 {
  %2 = alloca %class.Singleton*, align 8
  store %class.Singleton* %0, %class.Singleton** %2, align 8
  %3 = load %class.Singleton*, %class.Singleton** %2, align 8
  %4 = getelementptr inbounds %class.Singleton, %class.Singleton* %3, i32 0, i32 0
  store i32 10, i32* %4, align 4
  ret void
}

; Function Attrs: noinline ssp uwtable
define internal void @_GLOBAL__sub_I_singleton2.cpp() #0 section "__TEXT,__StaticInit,regular,pure_instructions" {
  call void @__cxx_global_var_init()
  ret void
}
```

non-constexpr

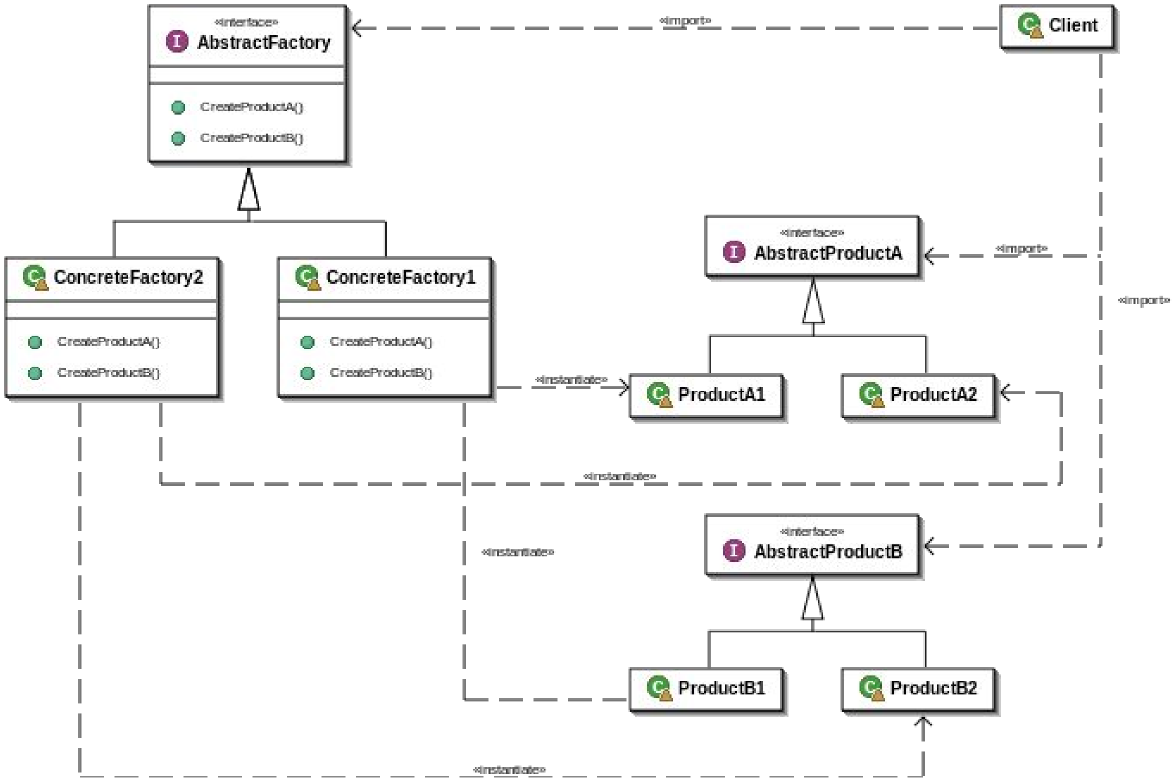
Initializing class members

global_ctors calls it.

CRTP pattern for Singleton

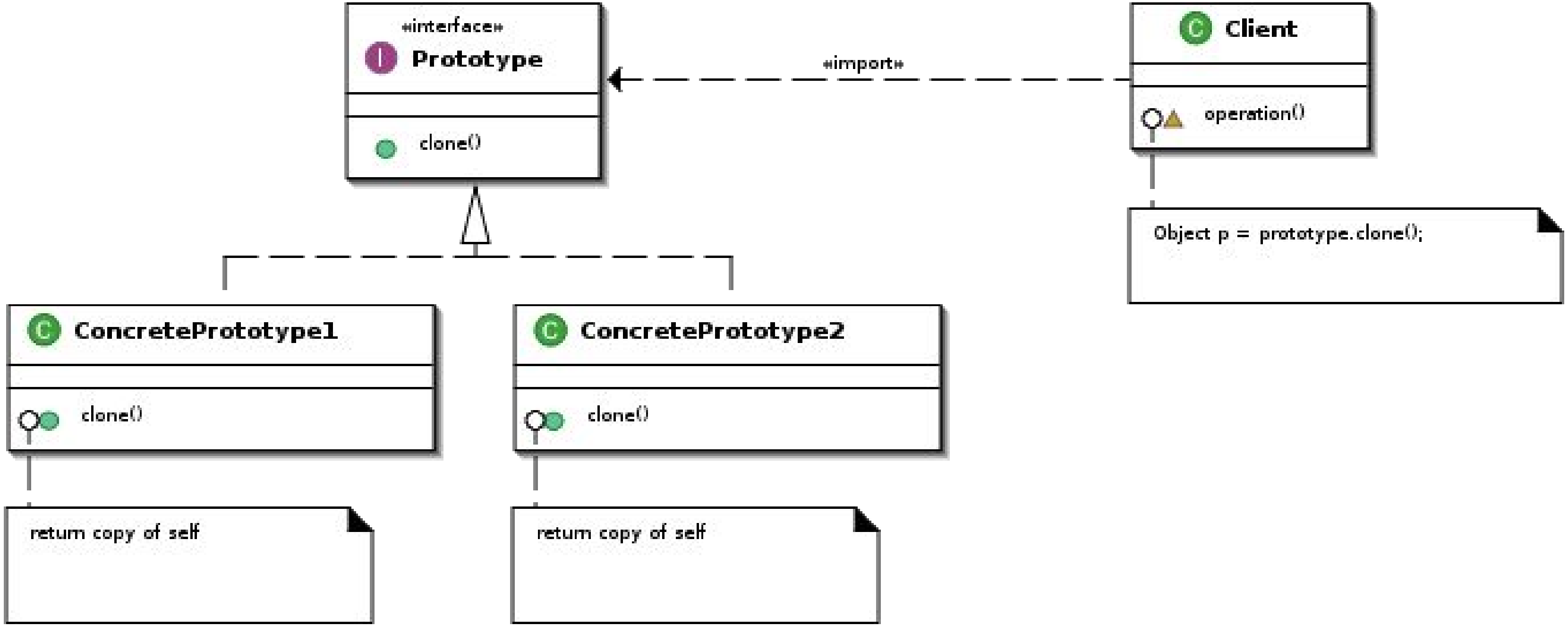
```
template<typename ActualClass> // Singleton policy class
class Singleton {
public:

    template<typename... Args>
    static ActualClass& getInstance(Args... args) // Singleton
    {
        // Guaranteed to be destroyed.
        // Instantiated on first use.
        // Thread safe in C++11
        static ActualClass instance;
        return instance;
    }

protected:
    Singleton() = default;
    ~Singleton() = default;
    Singleton(const Singleton&) = delete;
    Singleton& operator=(const Singleton&) = delete;
};

class Foo: public Singleton<Foo> {
    friend class Singleton<Foo>;
    Foo() = default;
    //Rest of functionality for class Foo
}
```

Singleton and Abstract Factory



Concrete Factories are usually needed in a single copy.

Singleton and Prototype



Singleton and Prototype

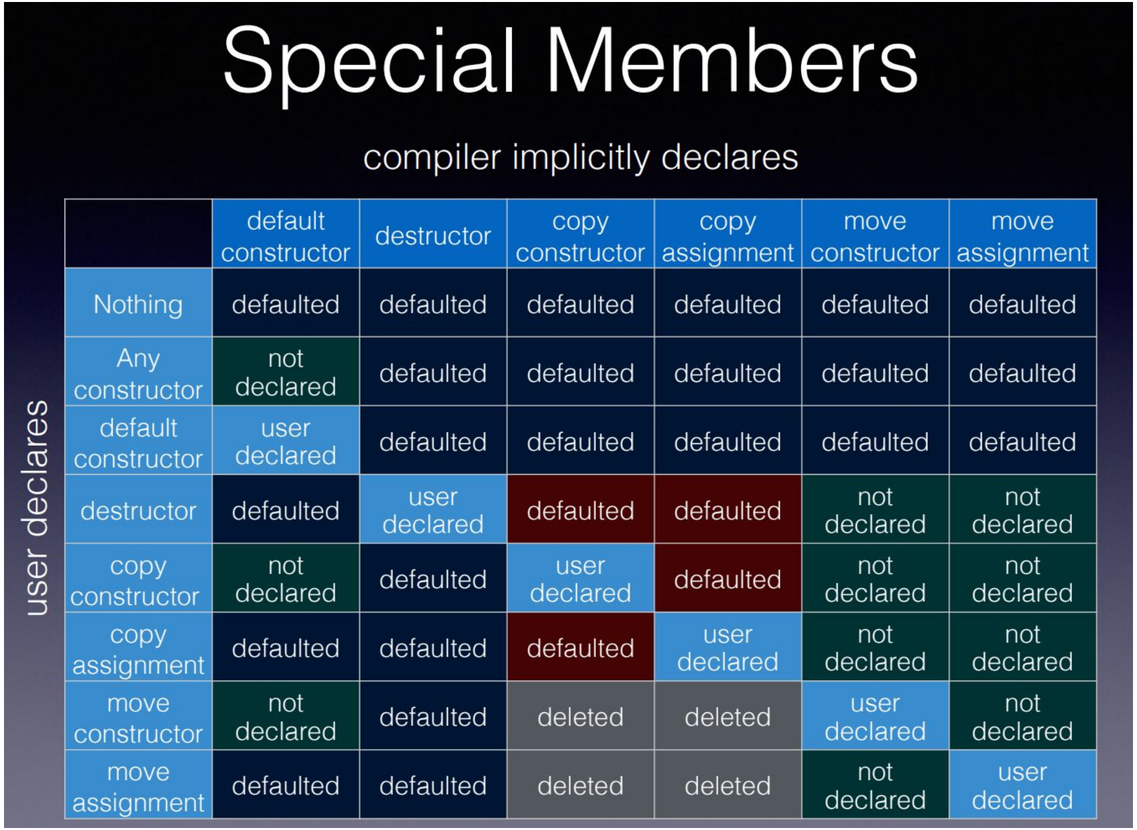
```
class Warrior;
typedef map<std::string, Warrior*> Registry;
Registry& getRegistry() {
      static Registry _instance;
      return _instance;
}
class Warrior {
public:
      virtual Warrior* clone() = 0;
      virtual ~Warrior() {}
      static Warrior* createWarrior(std::string id) {
            Registry& r = getRegistry();
            if (r.find(id) != r.end()) {
                  return r[id]->clone();
            }
            return nullptr;
      }

protected:
      static void addPrototype(std::string id, Warrior* prototype) {
            Registry& r = getRegistry();
            r[id] = prototype;
      }
}
```

```
class Archer: public Warrior {
public:
      Warrior* clone() {
      ...
      }
private:
      Archer() {
        Warrior::addPrototype("archer", this);
        ...
      }
      static Archer prototype;
      ...
}
class Horseman: public Warrior {
public:
      Warrior* clone() {
      ...
      }
private:
      Horseman() {
        Warrior::addPrototype("horseman", this);
        ...
        }
      static Horseman prototype;
      ...
}
```

References

- Evolution - https://msdn.microsoft.com/en-us/library/ee817670.aspx
- Double-Check Lock - http://preshing.com/20130930/double-checked-locking-is-fixed-in-cpp11/
- CRTP = https://stackoverflow.com/questions/4173254/what-is-the-curiously-recurring-template-pattern-crtp
- LLVM Language Reference Manual - https://llvm.org/docs/LangRef.html

Default compiler initialization of constructors..



Special Members

compiler implicitly declares

user declares	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted	defaulted	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted	not declared	not declared
copy assignment	defaulted	defaulted	defaulted	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared