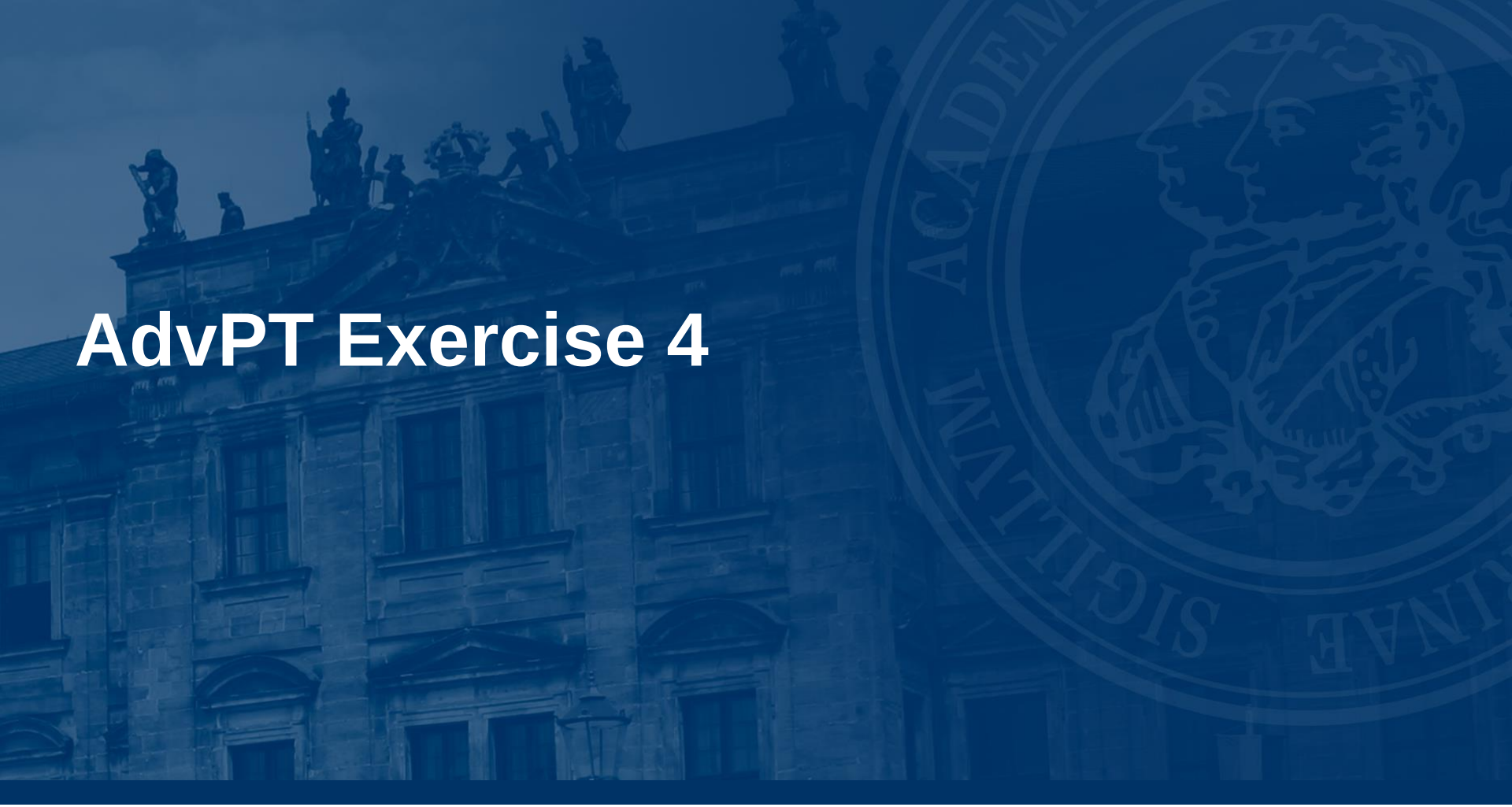


AdvPT Exercise 4

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

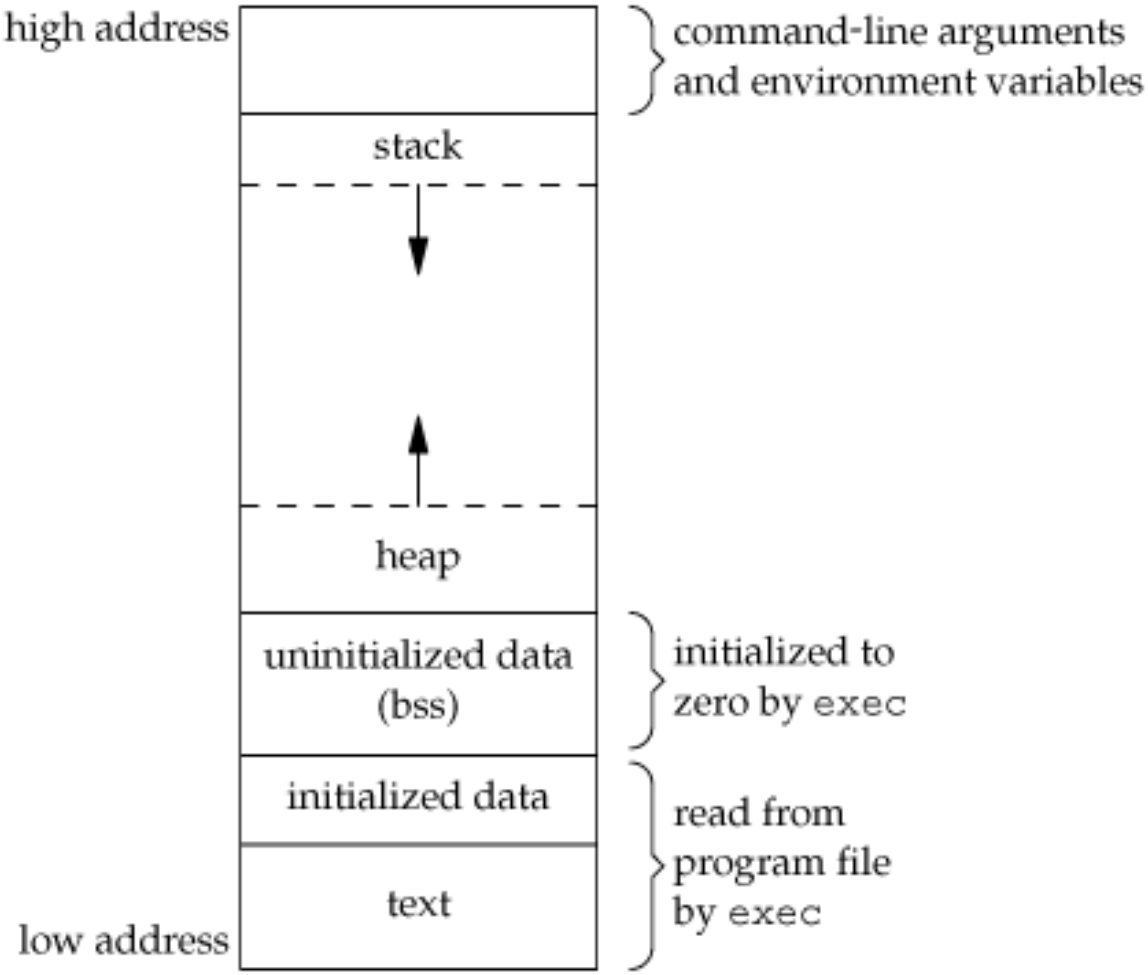
Content

- Memory management
- Valgrind

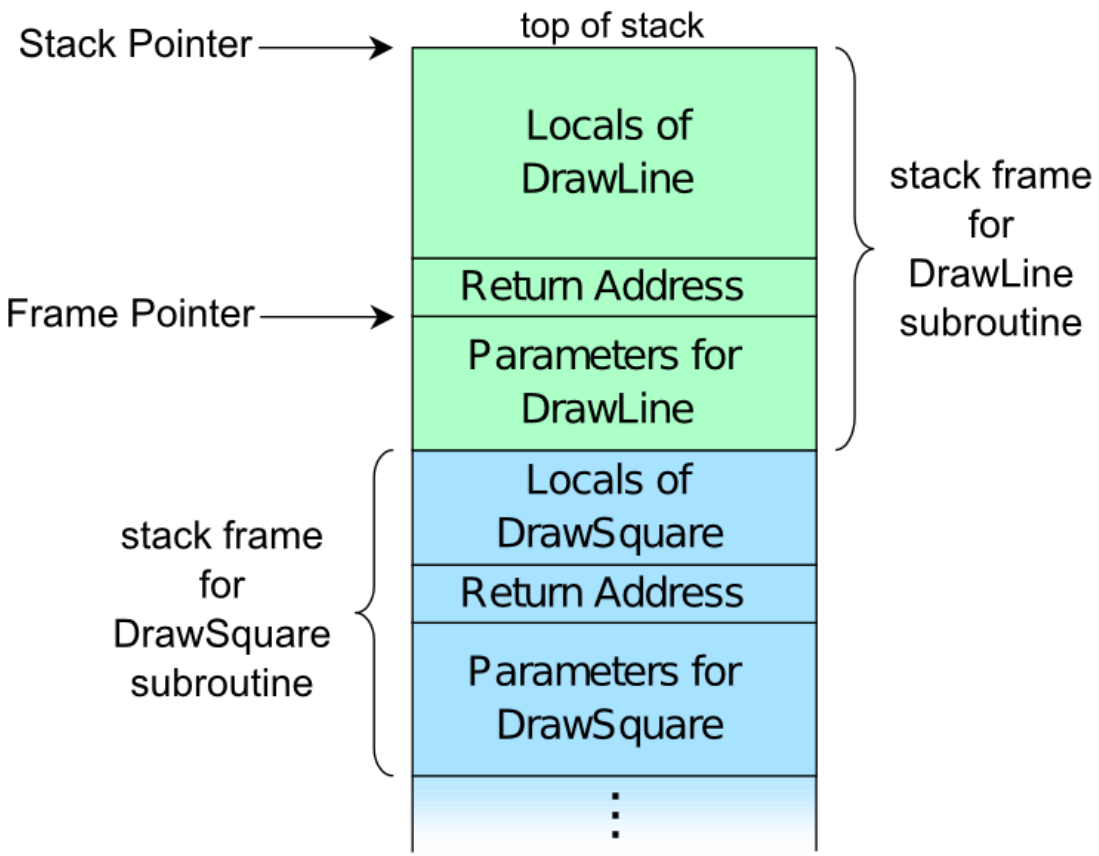
Memory management

- Memory management
  - Pointers & References
  - Multidimentional structures

Memory region



Stack

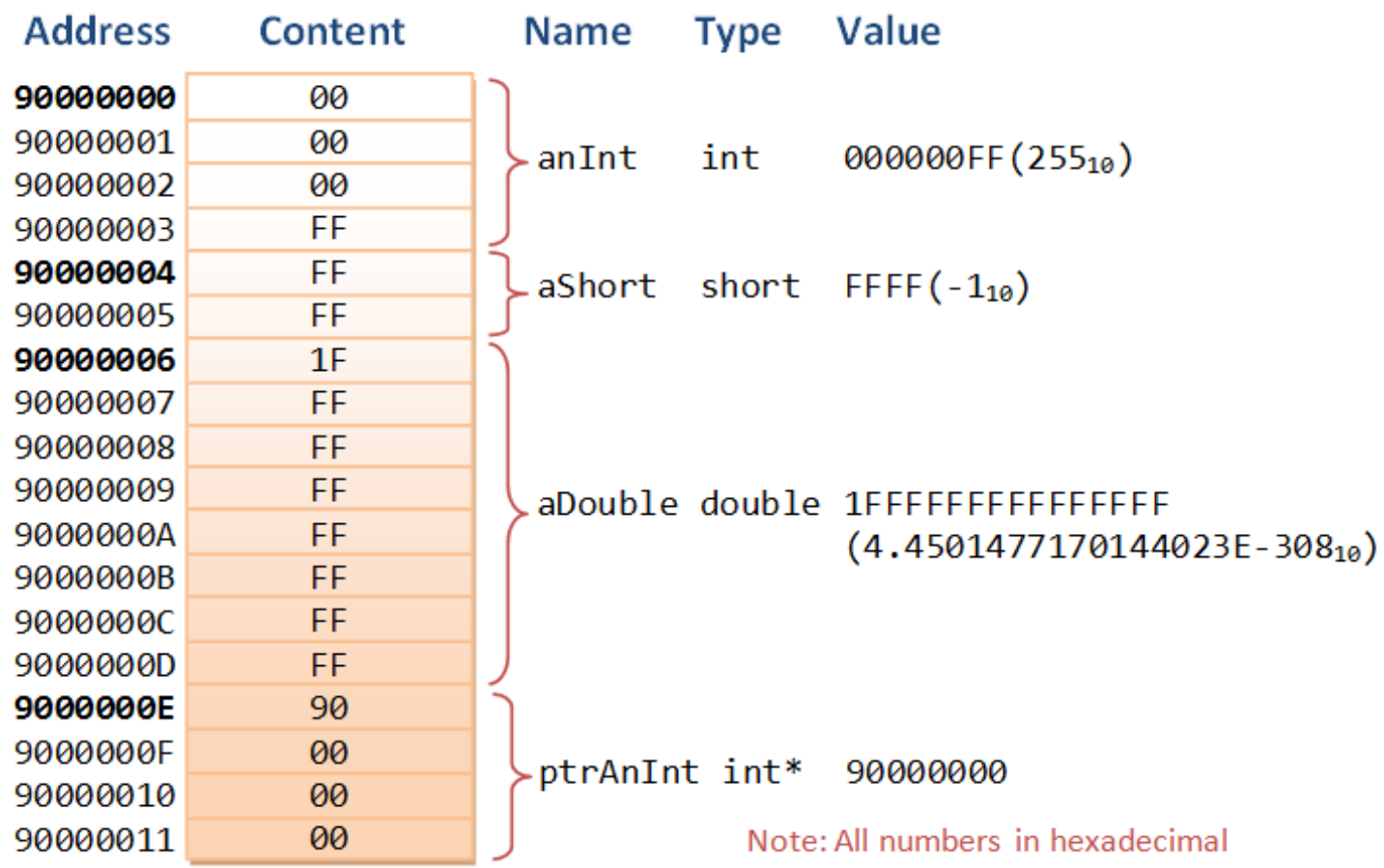

Stack Pointer
top of stack
Locals of DrawLine
Return Address
Frame Pointer
Parameters for DrawLine
stack frame for DrawLine subroutine
Locals of DrawSquare
Return Address
Parameters for DrawSquare
stack frame for DrawSquare subroutine

Pointer

Address	Content	Name	Type	Value
90000000	00			
90000001	00	anInt	int	000000FF(255_{10})
90000002	00			
90000003	FF			
90000004	FF	aShort	short	FFFF(-1_{10})
90000005	FF			
90000006	1F			
90000007	FF			
90000008	FF			
90000009	FF	aDouble	double	1FFFFFFFFFFFFFF
9000000A	FF			($4.4501477170144023E-308_{10}$)
9000000B	FF			
9000000C	FF			
9000000D	FF			

- memory: linear address space
- C/C++ gives you the power (responsibility!) to access arbitrary memory regions
- OS prevents access to certain memory regions (SEGFAULT)
- Pointer store memory addresses

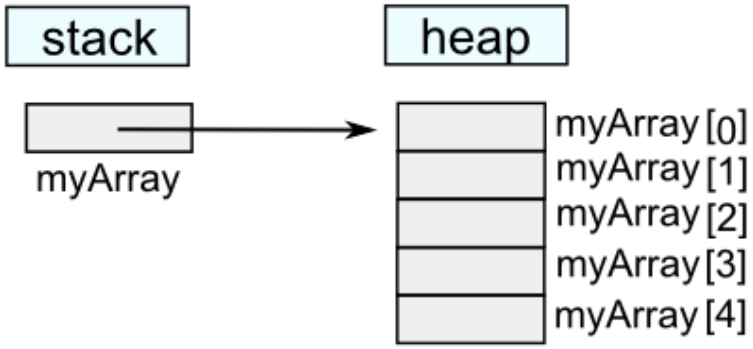
Pointer



- memory: linear address space
- C/C++ gives you the power (responsibility!) to access arbitrary memory regions
- OS prevents access to certain memory regions (SEGFAULT)
- Pointer store memory addresses

Heap

```
int * myArray = new int[5];
```



Memory Leaks

- Memory which is no longer needed is not released.

```
int main( int argc, char**argv)
{
        int * arr = 0;
        arr = new int[20]; //Leak!!

        return 0;
}


int main( int argc, char**argv)
{
        int * arr = 0;
        arr = new int[20];

        delete [] arr;

        return 0;
}
```

Valgrind

- Valgrind is an Open-Source system for debugging and profiling Linux programs (https://www.valgrind.org/)
- Where does the name come from?
  Valgrind is the name of the main entrance to Valhalla, only those judged worthy are allowed to pass through Valgrind
- Tool Suite
  - Memcheck
  - Cachegrind
  - Massif
  - Helgrind
  - And more...
- Focus on Memcheck for this exercise

Valgrind

- Ready to use on CIP-Pool Computers
- How to call Valgrind?

```
valgrind --leak-check=full ./myProg
```

- Important note:
  Compile with `-g` and `-O0`

```
oh08ozem@ciple2:~/Advanced Programming Techniques/Tutor/Valgrind$ valgrind --leak-check=full ./memLeak1
==8599== Memcheck, a memory error detector
==8599== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8599== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==8599== Command: ./memLeak1
==8599==
==8599==
==8599== HEAP SUMMARY:
==8599==     in use at exit: 0 bytes in 0 blocks
==8599==   total heap usage: 2 allocs, 2 frees, 72,784 bytes allocated
==8599==
==8599== All heap blocks were freed -- no leaks are possible
==8599==
==8599== For counts of detected and suppressed errors, rerun with: -v
==8599== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Memory Leaks


```
int main( int argc, char**argv)
{
    int * arr = 0;
    arr = new int[20]; //Leak!!

    return 0;
}
```

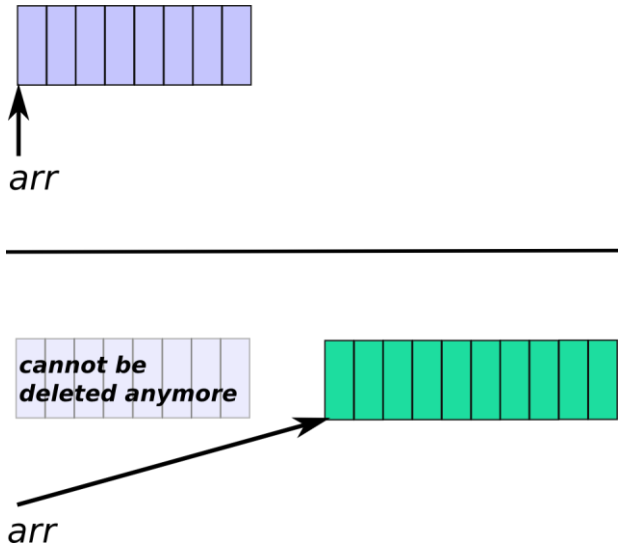
```
==9974== Memcheck, a memory error detector
==9974== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9974== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==9974== Command: ./memLeak1
==9974==
==9974==
==9974== HEAP SUMMARY:
==9974==     in use at exit: 80 bytes in 1 blocks
==9974==   total heap usage: 2 allocs, 1 frees, 72,784 bytes allocated
==9974==
==9974== 80 bytes in 1 blocks are definitely lost in loss record 1 of 1
==9974==    at 0x483650F: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64
-linux.so)
==9974==    by 0x109155: main (memleak1.cpp:4)
==9974==
==9974== LEAK SUMMARY:
==9974==    definitely lost: 80 bytes in 1 blocks
==9974==    indirectly lost: 0 bytes in 0 blocks
==9974==      possibly lost: 0 bytes in 0 blocks
==9974==    still reachable: 0 bytes in 0 blocks
==9974==         suppressed: 0 bytes in 0 blocks
==9974==
==9974== For counts of detected and suppressed errors, rerun with: -v
==9974== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Common errors

```
int main( int argc, char**argv)
{
        int * arr = 0;
        arr = new int[20]; //Leak
        //do something with array, now larger array is needed
        arr = new int[40];

        delete [] arr;

        return 0;
}
```



Common errors


```
int main( int argc, char**argv)
{
      int * arr = 0;
      arr = new int[20]; //Leak
      //do something with array, now larger array is needed
      arr = new int[40];

      delete [] arr;
      return 0;
}
```

```
==12400== Memcheck, a memory error detector
==12400== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12400== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==12400== Command: ./memLeak2
==12400==
==12400==
==12400== HEAP SUMMARY:
==12400==     in use at exit: 80 bytes in 1 blocks
==12400==   total heap usage: 3 allocs, 2 frees, 72,944 bytes allocated
==12400==
==12400== 80 bytes in 1 blocks are definitely lost in loss record 1 of 1
==12400==    at 0x483650F: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd6
4-linux.so)
==12400==    by 0x109165: main (memleak2.cpp:4)
==12400==
==12400== LEAK SUMMARY:
==12400==    definitely lost: 80 bytes in 1 blocks
==12400==    indirectly lost: 0 bytes in 0 blocks
==12400==      possibly lost: 0 bytes in 0 blocks
==12400==    still reachable: 0 bytes in 0 blocks
==12400==         suppressed: 0 bytes in 0 blocks
==12400==
==12400== For counts of detected and suppressed errors, rerun with: -v
==12400== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Common errors


```
#include <iterator>
#include <iostream>

using namespace std;

int main( int argc, char**argv)
{
  int arr1[3] = { 1, 2, 100  };
  int arr2[2] = { 9, 8  };

  for( int i=0; i<=2; ++i )
    arr2[i] += arr1[i];

  copy( arr1, arr1+3, ostream_iterator<int>(cout, "," ) );
  cout << endl;
  copy( arr2, arr2+2, ostream_iterator<int>(cout, "," ) );
  cout << endl;
}
```

Common errors


```cpp
#include <iterator>
#include <iostream>

using namespace std;

int main( int argc, char**argv)
{
  int arr1[3] = { 1, 2, 100  };
  int arr2[2] = { 9, 8  };

  for( int i=0; i<=2; ++i )
    arr2[i] += arr1[i];

  copy( arr1, arr1+3, ostream_iterator<int>(cout, "," ) );
  cout << endl;
  copy( arr2, arr2+2, ostream_iterator<int>(cout, "," ) );
  cout << endl;
}
```

```
==15271== Memcheck, a memory error detector
==15271== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15271== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==15271== Command: ./memLeak3
==15271==
101,2,100,
10,10,
==15271==
==15271== HEAP SUMMARY:
==15271==     in use at exit: 0 bytes in 0 blocks
==15271==   total heap usage: 2 allocs, 2 frees, 73,728 bytes allocated
==15271==
==15271== All heap blocks were freed -- no leaks are possible
==15271==
==15271== For counts of detected and suppressed errors, rerun with: -v
==15271== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Common errors


```
int main( int argc, char** argv )
{
    int * arr = new int [500];

    for( int i=0; i<500; ++i )
        arr[i] = 0;

    delete arr;
}
```

Common errors


```
int main( int argc, char** argv )
{
    int * arr = new int [500];

    for( int i=0; i<500; ++i )
        arr[i] = 0;

    delete arr;
}
```

```
==15837== Memcheck, a memory error detector
==15837== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15837== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==15837== Command: ./memLeak4
==15837==
==15837== Mismatched free() / delete / delete []
==15837==    at 0x483708B: operator delete(void*, unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memch
eck-amd64-linux.so)
==15837==    by 0x1091A2: main (memleak4.cpp:8)
==15837==  Address 0x4db9c80 is 0 bytes inside a block of size 2,000 alloc'd
==15837==    at 0x483650F: operator new[](unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd6
4-linux.so)
==15837==    by 0x10915D: main (memleak4.cpp:3)
==15837==
==15837==
==15837== HEAP SUMMARY:
==15837==     in use at exit: 0 bytes in 0 blocks
==15837==   total heap usage: 2 allocs, 2 frees, 74,704 bytes allocated
==15837==
==15837== All heap blocks were freed -- no leaks are possible
==15837==
==15837== For counts of detected and suppressed errors, rerun with: -v
==15837== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Common errors


```
std::vector<int> & createUnitVector( int len )
{
        std::vector<int> vec ( 3, 1 );
        return vec;
}
int main( int argc, char** argv )
{
        auto myVec = createUnitVector( 4 );
        myVec[4] = 5;
        return 0;
}
```

Common errors


```
==16842== Memcheck, a memory error detector
==16842== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16842== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==16842== Command: ./memLeak5
==16842==
==16842== Invalid write of size 4
==16842==    at 0x109206: main (memleak5.cpp:12)
==16842==  Address 0x4db9c90 is 4 bytes after a block of size 12 alloc'd
==16842==    at 0x4835DEF: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-
linux.so)
==16842==    by 0x10974D: __gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*) (new_allocator.h:111)
==16842==    by 0x1096BA: std::allocator_traits<std::allocator<int> >::allocate(std::allocator<int>&, unsigned long) (a
lloc_traits.h:436)
==16842==    by 0x10960B: std::_Vector_base<int, std::allocator<int> >::_M_allocate(unsigned long) (stl_vector.h:296)
==16842==    by 0x1094F8: std::_Vector_base<int, std::allocator<int> >::_M_create_storage(unsigned long) (stl_vector.h:
311)
==16842==    by 0x10939A: std::_Vector_base<int, std::allocator<int> >::_Vector_base(unsigned long, std::allocator<int>
 const&) (stl_vector.h:260)
==16842==    by 0x10928D: std::vector<int, std::allocator<int> >::vector(unsigned long, int const&, std::allocator<int>
 const&) (stl_vector.h:429)
==16842==    by 0x1091A0: createUnitVector(int) (memleak5.cpp:5)
==16842==    by 0x1091F4: main (memleak5.cpp:11)
==16842==
==16842==
==16842== HEAP SUMMARY:
==16842==     in use at exit: 0 bytes in 0 blocks
==16842==   total heap usage: 2 allocs, 2 frees, 72,716 bytes allocated
==16842==
==16842== All heap blocks were freed -- no leaks are possible
==16842==
==16842== For counts of detected and suppressed errors, rerun with: -v
==16842== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Common errors


```cpp
#include <vector>

std::vector<int> createUnitVector( int len )
{
    std::vector<int> vec ( 3, 1 );
    return vec;
}
int main( int argc, char** argv )
{
    auto myVec = createUnitVector( 4 );
    myVec[4] = 5;
    return 0;
}
```

```
==16842== Memcheck, a memory error detector
==16842== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16842== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==16842== Command: ./memLeak5
==16842==
==16842== Invalid write of size 4
==16842==    at 0x109206: main (memleak5.cpp:12)
==16842==  Address 0x4db9c90 is 4 bytes after a block of size 12 alloc'd
```

Common errors


```cpp
#include <vector>

std::vector<int> createUnitVector( int len )
{
    std::vector<int> vec ( 3, 1 );
    return vec;
}
int main( int argc, char** argv )
{
    auto myVec = createUnitVector( 4 );
    myVec[4] = 5;
    return 0;
}
```

```
==16842==  Address 0x4db9c90 is 4 bytes after a block of size 12 alloc'd
==16842==    at 0x4835DEF: operator new(unsigned long) (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-
linux.so)
==16842==    by 0x10974D: __gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*) (new_allocator.h:111)
==16842==    by 0x1096BA: std::allocator_traits<std::allocator<int> >::allocate(std::allocator<int>&, unsigned long) (a
lloc_traits.h:436)
==16842==    by 0x10960B: std::_Vector_base<int, std::allocator<int> >::_M_allocate(unsigned long) (stl_vector.h:296)
==16842==    by 0x1094F8: std::_Vector_base<int, std::allocator<int> >::_M_create_storage(unsigned long) (stl_vector.h:
311)
==16842==    by 0x10939A: std::_Vector_base<int, std::allocator<int> >::_Vector_base(unsigned long, std::allocator<int>
 const&) (stl_vector.h:260)
==16842==    by 0x10928D: std::vector<int, std::allocator<int> >::vector(unsigned long, int const&, std::allocator<int>
 const&) (stl_vector.h:429)
==16842==    by 0x1091A0: createUnitVector(int) (memleak5.cpp:5)
==16842==    by 0x1091F4: main (memleak5.cpp:11)
```

Common errors


```cpp
#include <vector>

std::vector<int> createUnitVector( int len )
{
    std::vector<int> vec ( 3, 1 );
    return vec;
}
int main( int argc, char** argv )
{
    auto myVec = createUnitVector( 4 );
    myVec[4] = 5;
    return 0;
}
```

```
==16842==    by 0x1091A0: createUnitVector(int) (memleak5.cpp:5)
==16842==    by 0x1091F4: main (memleak5.cpp:11)
==16842==
==16842==
==16842== HEAP SUMMARY:
==16842==     in use at exit: 0 bytes in 0 blocks
==16842==   total heap usage: 2 allocs, 2 frees, 72,716 bytes allocated
==16842==
==16842== All heap blocks were freed -- no leaks are possible
==16842==
==16842== For counts of detected and suppressed errors, rerun with: -v
==16842== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Valgrind – more ressources

- For more information: https://www.valgrind.org/
- Quick Start
- FAQ
- User Manual

Rule of three/five

- One special rule of thumb, which defines three/five special member functions:
  - Destructor
  - Copy constructor
  - Copy assignment operator
  - Move constructor
  - Move assignment operator

  - The Rule of Three/Five claims that **if one** of these had to be defined by the programmer, i.e. the compiler-generated version does not fit the needs of the class in one case and it will probably not fit in the other cases either.

- Example: Vector Class

Thank you and good luck!

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT