CSCI 390 – Special Topics in C++

Lecture 5

9/4/18

Time To Turn Off Cell Phones

Pointers (cont)

- Pointers to out of scope variables are risky.

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <cassert>

#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint32;
  puint32 puint{nullptr};
  std::cout << DUMPVAR(puint) << std::endl;
  {
    uint32_t AnEvenPrime{2u};
    puint = &AnEvenPrime;
    std::cout << DUMPVAR(AnEvenPrime) << std::endl;
    std::cout << DUMPVAR(puint) << std::endl;
    std::cout << DUMPVAR(*puint) << std::endl;
  }
  // AnEvenPrime is now out of scope – poof, gone!
//  double v1{1.0};  double v2{2.0};  double v3{3.0};
//  std::cout << DUMPVAR(v3) << std::endl;
  std::cout << DUMPVAR(puint) << std::endl;
  std::cout << DUMPVAR(*puint) << std::endl;

  return 0;
}
```

Console:

```
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0
Variable: AnEvenPrime, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0x7ffc28b42fec
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0x7ffc28b42fec
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Pointers (cont)

- Pointers to out of scope variables are risky.

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <cassert>

#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint32;
  puint32 puint{nullptr};
  std::cout << DUMPVAR(puint) << std::endl;
  {
    uint32_t AnEvenPrime{2u};
    puint = &AnEvenPrime;
    std::cout << DUMPVAR(AnEvenPrime) << std::endl;
    std::cout << DUMPVAR(puint) << std::endl;
    std::cout << DUMPVAR(*puint) << std::endl;
  }
  // AnEvenPrime is now out of scope – poof, gone!
  double v1{1.0};  double v2{2.0};  double v3{3.0};
  std::cout << DUMPVAR(v3) << std::endl;
  std::cout << DUMPVAR(puint) << std::endl;
  std::cout << DUMPVAR(*puint) << std::endl;

  return 0;
}
```

Old Console:

```
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0
Variable: AnEvenPrime, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0x7ffc28b42fec
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffc28b42ff8, Value: 0x7ffc28b42fec
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffc28b42fec, Value: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Console:

```
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffed09f4168, Value: 0
Variable: AnEvenPrime, Type: unsigned int, Length: 4, Address: 0x7ffed09f4160, Value: 2
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffed09f4168, Value: 0x7ffed09f4160
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffed09f4160, Value: 2
Variable: v3, Type: double, Length: 8, Address: 0x7ffed09f4170, Value: 3
Variable: puint, Type: unsigned int*, Length: 8, Address: 0x7ffed09f4168, Value: 0x7ffed09f4160
Variable: *puint, Type: unsigned int, Length: 4, Address: 0x7ffed09f4160, Value: 11115576


...Program finished with exit code 0
Press ENTER to exit console.
```

Pointers (cont)

- `sizeof()` returns the number of bytes required by a <type> or <rvalue>. It returns a `size_t`.

//main.cpp:

```
#include <iostream>
#include <cstdint>

#include "Helper.h"

int main(void)
{
  double pi{3.14};
  std::cout << DUMPTYPE(size_t) << std::endl;
  std::cout << DUMPVAL(sizeof(0u)) << std::endl;
  std::cout << DUMPVAL(sizeof(uint32_t)) << std::endl;
  std::cout << DUMPVAL(sizeof(pi)) << std::endl;

  return 0;
}
```

Console:

```
Type: unsigned long, Length: 8
Expression: sizeof(0u), Type: unsigned long, Length: 8, Value: 4
Expression: sizeof(uint32_t), Type: unsigned long, Length: 8, Value: 4
Expression: sizeof(pi), Type: unsigned long, Length: 8, Value: 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Pointers (cont)
std::malloc

std::malloc (https://en.cppreference.com/w/cpp/memory/c/malloc)
Defined in header <cstdlib>

```
void* malloc(size_t size);
```

Allocates size bytes of uninitialized storage from the heap.

If allocation succeeds, returns a pointer to the lowest (first) byte in the allocated memory block that is suitably aligned for any intrinsic type.

If size is zero, the behavior is implementation defined
(null pointer may be returned, or some non-null pointer
may be returned that may not be used to access storage,
but has to be passed to std::free)

That which is std::malloc'ed must be std::free'ed.

Pointers (cont)
std::free

std::free (https://en.cppreference.com/w/cpp/memory/c/free)
Defined in header <cstdlib>

```
void* free(void *ptr);
```

Deallocates the space previously allocated by std::malloc
If `ptr` is a null pointer, the function does nothing.

The behavior is undefined if the value of `ptr` does not equal a value returned earlier by std::malloc

Pointers (cont)
malloc/free simple example

```
//main.cpp:
#include <iostream>
#include <cstdint>
#include <cstdlib>

#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint;

  puint pHeapInt{nullptr};

  std::cout << DUMPVAR(pHeapInt) << std::endl;

  pHeapInt = puint(malloc(sizeof(uint32_t)));

  std::cout << DUMPVAR(pHeapInt) << std::endl;
  std::cout << DUMPVAR(*pHeapInt) << std::endl;

  *pHeapInt = 23u;
  std::cout << DUMPVAR(*pHeapInt) << std::endl;

  std::free(pHeapInt);
  pHeapInt = nullptr;

  return 0;
}
```

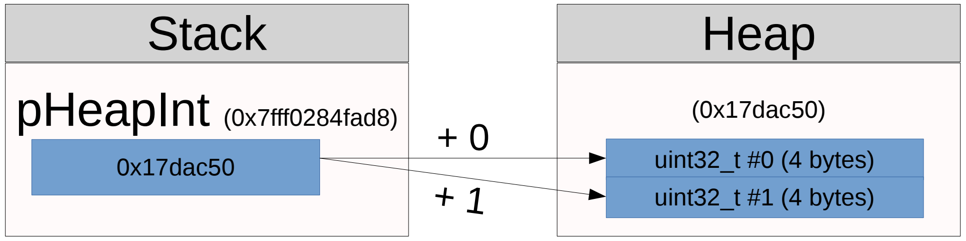
```
Console:
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7ffe6a52c738, Value: 0
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7ffe6a52c738, Value: 0x1d18c50
Variable: *pHeapInt, Type: unsigned int, Length: 4, Address: 0x1d18c50, Value: 0
Variable: *pHeapInt, Type: unsigned int, Length: 4, Address: 0x1d18c50, Value: 23


...Program finished with exit code 0
Press ENTER to exit console.
```

Pointers (cont)
Pointer Arithmetic By Example

```
//main.cpp:
#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <cassert>
#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint;

  puint pHeapInt{nullptr};

  std::cout << DUMPVAR(pHeapInt) << std::endl;

  pHeapInt = puint(malloc(2 * sizeof(uint32_t)));
  assert(pHeapInt);
  std::cout << DUMPVAR(pHeapInt) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 0) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 1) << std::endl;

  std::free(pHeapInt);
  pHeapInt = nullptr;

  return 0;
}
```

Console:

```
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7fff0284fad8, Value: 0
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7fff0284fad8, Value: 0x17dac50
Expression: pHeapInt + 0, Type: unsigned int*, Length: 8, Value: 0x17dac50
Expression: pHeapInt + 1, Type: unsigned int*, Length: 8, Value: 0x17dac54


...Program finished with exit code 0
Press ENTER to exit console.
```



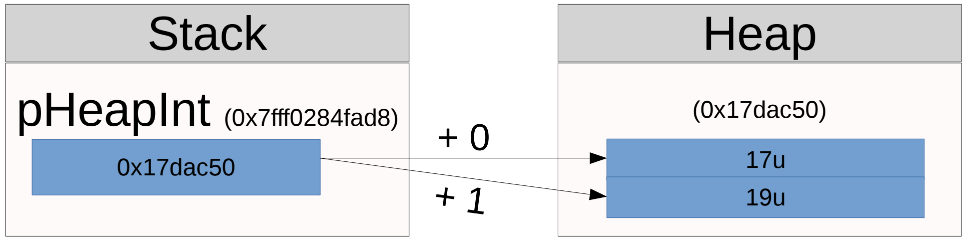
Pointers (cont)
Pointer Arithmetic By Example

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <cassert>
#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint;
  puint pHeapInt{nullptr};

  pHeapInt = puint(malloc(2 * sizeof(uint32_t)));
  assert(pHeapInt);
  std::cout << DUMPVAR(pHeapInt) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 0) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 1) << std::endl;

  *(pHeapInt + 0) = 17u;
  *(pHeapInt + 1) = 19u;

  std::cout << DUMPVAR(*(pHeapInt + 0)) << std::endl;
  std::cout << DUMPVAR(*(pHeapInt + 1)) << std::endl;

  auto& HeapInt1{*(pHeapInt + 1)};
  std::cout << DUMPVAR(HeapInt1) << std::endl;

  std::free(pHeapInt);
  pHeapInt = nullptr;

  // Danger! HeapInt1 now references unallocated memory.

  return 0;
}
```

Console:

```
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7fff00da5be8, Value: 0xebdc20
Expression: pHeapInt + 0, Type: unsigned int*, Length: 8, Value: 0xebdc20
Expression: pHeapInt + 1, Type: unsigned int*, Length: 8, Value: 0xebdc24
Variable: *(pHeapInt + 0), Type: unsigned int, Length: 4, Address: 0xebdc20, Value: 17
Variable: *(pHeapInt + 1), Type: unsigned int, Length: 4, Address: 0xebdc24, Value: 19
Variable: HeapInt1, Type: unsigned int, Length: 4, Address: 0xebdc24, Value: 19


...Program finished with exit code 0
Press ENTER to exit console.
```



Pointers (cont)
The Truth About The [ ] Operaor

- x[y] is defined to be *(x + y)
- y must be an integral (signed or unsigned) <rvalue>
- (x + y) must be a non-generic pointer <rvalue>
- The definition does not mention arrays.

Pointers (cont)
[ ] Example

```
//main.cpp:
#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <cassert>
#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint;
  puint pHeapInt{nullptr};

  pHeapInt = puint(malloc(2 * sizeof(uint32_t)));
  assert(pHeapInt);
  std::cout << DUMPVAR(pHeapInt) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 0) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 1) << std::endl;

  pHeapInt[0] = 17u;  // *(pHeapInt + 0) = 17u;
  pHeapInt[1] = 19u;  // *(pHeapInt + 1) = 19u;

  std::cout << DUMPVAR(*(pHeapInt + 0)) << std::endl;
  std::cout << DUMPVAR(*(pHeapInt + 1)) << std::endl;

  std::free(pHeapInt);
  pHeapInt = nullptr;

  return 0;
}
```

```
Console:
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7ffd6a282588, Value: 0x2298c20
Expression: pHeapInt + 0, Type: unsigned int*, Length: 8, Value: 0x2298c20
Expression: pHeapInt + 1, Type: unsigned int*, Length: 8, Value: 0x2298c24
Variable: pHeapInt[0], Type: unsigned int, Length: 4, Address: 0x2298c20, Value: 17
Variable: pHeapInt[1], Type: unsigned int, Length: 4, Address: 0x2298c24, Value: 19


...Program finished with exit code 0
Press ENTER to exit console.
```

Pointers (cont)
[ ] – Complex Example

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <cassert>
#include "Helper.h"

int main(void)
{
  typedef uint32_t *puint;
  puint pHeapInt{nullptr};

  pHeapInt = puint(malloc(3 * sizeof(uint32_t)));
  assert(pHeapInt);
  puint Median{pHeapInt + 1};
  Median[-1] = 3u;  // Same as pHeapInt + 0
  Median[0] = 5u;   // Same as pHeapInt + 1
  Median[1] = 7u;   // Same as pHeapInt + 2

  std::cout << DUMPVAR(pHeapInt) << std::endl;
  std::cout << DUMPVAL(pHeapInt + 1) << std::endl;
  std::cout << DUMPVAL(Median) << std::endl;
  std::cout << DUMPVAR(pHeapInt[0]) << std::endl;
  std::cout << DUMPVAR(pHeapInt[1]) << std::endl;
  std::cout << DUMPVAR(pHeapInt[2]) << std::endl;
  std::cout << DUMPVAR((pHeapInt + 1)[1]) << std::endl;
  std::cout << DUMPVAR(pHeapInt[2]) << std::endl;

  std::free(pHeapInt);
  pHeapInt = nullptr;

  return 0;
}
```

Console:

```
Variable: pHeapInt, Type: unsigned int*, Length: 8, Address: 0x7fff282899a8, Value: 0x1651c20
Expression: pHeapInt + 1, Type: unsigned int*, Length: 8, Value: 0x1651c24
Expression: Median, Type: unsigned int*, Length: 8, Value: 0x1651c24
Variable: pHeapInt[0], Type: unsigned int, Length: 4, Address: 0x1651c20, Value: 3
Variable: pHeapInt[1], Type: unsigned int, Length: 4, Address: 0x1651c24, Value: 5
Variable: pHeapInt[2], Type: unsigned int, Length: 4, Address: 0x1651c28, Value: 7
Variable: (pHeapInt + 1)[1], Type: unsigned int, Length: 4, Address: 0x1651c28, Value: 7
Variable: pHeapInt[2], Type: unsigned int, Length: 4, Address: 0x1651c28, Value: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

Understand this slide and you have grasped pointers!

By definition Median[-1] is:
*(Median - 1) = *(pHeapInt + 1 – 1) =
*(pHeapInt + 0) = pHeapInt[0]

By definition (pHeapInt +1)[1] is:
*(pHeapInt + 1 + 1) = *(pHeapInt + 2) =
pHeapInt[2]

By definition pHeapInt[2] is:
*(pHeapInt + 2)

Pointers – The C++ Way
Smart Pointers

- There are three smart pointers:
  - unique_ptr, shared_ptr and weak_ptr
    - See Smart Pointers in https://en.cppreference.com/w/cpp/memory
  - We study only unique_ptr.
    - See https://en.cppreference.com/w/cpp/memory/unique_ptr
- Smart pointers automatically free memory when they go out of scope.
- #include <memory>

Pointers – The C++ Way (cont) unique_ptr Scalar Example

- Header: <memory>
- Syntax:
  std::unique_ptr<<type>**>** <identifier> **{new** <type>**};**

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <memory>
#include "Helper.h"

int main(void)
{
  std::unique_ptr<uint32_t> HeapInt{new uint32_t};

  std::cout << DUMPTYPE(HeapInt) << std::endl;

  *HeapInt = 3u;

  std::cout << DUMPVAR(*HeapInt) << std::endl;

  //  Do not free!  Smart pointer takes care of this.

  return 0;
}
```

Console:

```
Type: std::unique_ptr<unsigned int, std::default_delete<unsigned int> >, Length: 8
Variable: *HeapInt, Type: unsigned int, Length: 4, Address: 0xf3fc20, Value: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

No [ ] operator!

Pointers – The C++ Way (cont) unique_ptr Array Example

- Header: <memory>
- Syntax:
  std::unique_ptr<<type>**[ ]>** <identifier> **{new** <type>**(**<rvalue>**)};**

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include <memory>
#include "Helper.h"

int main(void)
{
  std::unique_ptr<uint32_t[ ]> HeapInt{new uint32_t(3)};

  std::cout << DUMPTYPE(HeapInt) << std::endl;

  HeapInt[0] = 3u;
  HeapInt[1] = 5u;
  HeapInt[2] = 7u;

  std::cout << DUMPVAR(HeapInt[0]) << std::endl;
  std::cout << DUMPVAR(HeapInt[1]) << std::endl;
  std::cout << DUMPVAR(HeapInt[2]) << std::endl;

  //  Do not free!  Smart pointer takes care of this.

  return 0;
}
```

Console:

```
Type: std::unique_ptr<unsigned int [], std::default_delete<unsigned int []> >, Length: 8
Variable: HeapInt[0], Type: unsigned int, Length: 4, Address: 0x2370c20, Value: 3
Variable: HeapInt[1], Type: unsigned int, Length: 4, Address: 0x2370c24, Value: 5
Variable: HeapInt[2], Type: unsigned int, Length: 4, Address: 0x2370c28, Value: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

[ ] operator!

Arrays

- Syntax:
  <type> <identifier>**[**<positive rvalue>**]**[rvalue initializer>]**;**
  <type> <identifier>**[ ]** <rvalue initializer>**;**
- Declares multiple instances of <type> and optionally initializes.
- [ ] is NOT an operator.

//main.cpp:

```
#include <iostream>
#include <cstdint>
#include "Helper.h"

int main(void)
{
  uint32_t Array1[3];
  uint32_t Array2[3]{3u, 5u, 7u};
  uint32_t Array3[]{3u, 5u, 7u};

  std::cout << DUMPTYPE(Array1) << std::endl;
  std::cout << DUMPTYPE(Array2) << std::endl;
  std::cout << DUMPTYPE(Array3) << std::endl;

  std::cout << DUMPTYPE(Array1 + 0) << std::endl;

  std::cout << DUMPVAR(Array3[0]) << std::endl;
  std::cout << DUMPVAR(Array3[1]) << std::endl;
  std::cout << DUMPVAR(Array3[2]) << std::endl;

  return 0;
}
```

Console:

```
Type: unsigned int [3], Length: 12
Type: unsigned int [3], Length: 12
Type: unsigned int [3], Length: 12
Type: unsigned int*, Length: 8
Variable: Array3[0], Type: unsigned int, Length: 4, Address: 0x7ffd40900800, Value: 3
Variable: Array3[1], Type: unsigned int, Length: 4, Address: 0x7ffd40900804, Value: 5
Variable: Array3[2], Type: unsigned int, Length: 4, Address: 0x7ffd40900808, Value: 7


...Program finished with exit code 0
Press ENTER to exit console.
```

Note: {3u, 5u, 7u} is a unint32_t const array.