CSCI 390 – Special Topics in C++

Lecture 4

8/30/18

Time To Turn Off Cell Phones

Global Variables

- Declared outside all local scopes (i.e., not inside braces) AND/OR declared with `static` attribute.
  - Can be accessed with `::` global scope qualifier when needed.
- The static attribute can be used inside a local scope.
- Global variables should be initialized.
- `extern` attribute allows a compile unit to access a global variable defined in another compile unit.
  - `extern` declarations CANNOT be initialized.

Global Variables (cont)

- Declared outside all local scopes (i.e., not inside braces) AND/OR declared with `static` attribute.
  - Can be accessed with `::` global scope qualifier when needed.

//main.cpp:

```
#include <iostream>

#include "Helper.h"

const double pi{3.14159265358979323846264 33};

int main(void)
{
  const double pi{3.14};

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

  return 0;
}
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x7ffeaf1658b8, Value: 3.14
Variable: ::pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Global Variables (cont)

- `extern` attribute allows a compile unit to access a global variable defined in another compile unit.

//main.cpp:

```
#include <iostream>

#include "Helper.h"

extern double pi;

int main(void)
{
  const double pi{3.14};

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

  return 0;
}
```

//globals.cpp:

```
double pi{3.14159265358979323846264433};
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x7fffbd85d408, Value: 3.14
Variable: ::pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Global Variables (cont)

- `static` restricts visibility to compile unit.

//main.cpp:

```
#include <iostream>

#include "Helper.h"

extern double pi;

int main(void)
{
  const double pi{3.14};

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

  return 0;
}
```

//globals.cpp:

```
static double pi{3.14159265358979323846264 33};
```

Console:

```
/var/tmp/cc9EHfN5.o: In function `main':
main.cpp:(.text.startup+0x115): undefined reference to `pi'
main.cpp:(.text.startup+0x1b4): undefined reference to `pi'
collect2: error: ld returned 1 exit status
```

Global Variables (cont)

- `static` inside local scope restricts visibility to local scope.

//main.cpp:

```
#include <iostream>

#include <cstdint>

uint32_t  GetSequence(void);

int main(void)
{
  std::cout << "Seq: " << GetSequence() << std::endl;
  std::cout << "Seq: " << GetSequence() << std::endl;
  std::cout << "Seq: " << GetSequence() << std::endl;

  return 0;
}

uint32_t  GetSequence(void)
{
  static auto SequenceNumber{0u};

  return ++SequenceNumber;
}
```

Console:

```
Seq: 1
Seq: 2
Seq: 3


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

Global Variables (cont)

- Notes:
  - Linker resolves externs.
    - Typically gives one error per use. Beware of error storms!
  - Linker does not check type, but you can work around this.

Global Variables (cont)

- Linker does not check type, but you can work around this.

//main.cpp:

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

#include "Helper.h"

#include "globals.h"

int main(void)
{
  std::cout << DUMPVAR(pi) << std::endl;

  return 0;
}
```

//globals.h:

```
#include <cstdint>

extern uint64_t pi;
```

//globals.cpp:

```
#include "globals.h"

double pi{3.14159265358979323846264 33};
```

Console:

```
globals.cpp:4:8: error: conflicting declaration 'double pi'
 double pi{3.14159265358979323846264 33};
        ^
In file included from globals.cpp:2:0:
globals.h:4:17: note: previous declaration as 'uint64_t pi'
 extern uint64_t pi;
                 ^
```

Global Variables (cont)

- Linker does not check type, but you can work around this.

//main.cpp:

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

#include "Helper.h"

#include "globals.h"

int main(void)
{
  std::cout << DUMPVAR(pi) << std::endl;

  return 0;
}
```

//globals.h:

```
#include <cstdint>

extern double pi;
```

//globals.cpp:

```
#include "globals.h"

double pi{3.14159265358979323846264 33};
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Namespaces

- Namespaces help prevent name conflicts in large projects by gathering declarations under a name.
  - Program components can be types, variables and globals.
  - Use the binary operator `::` to access components in a namespace:
    <namespace>::<identifier>
    e.g., std::cout
  - Use unary `::` operator to access global components:
    ::<global>

Namespace (cont)

- Namespace definition:

```
namespace <identifier>
{
  <declarations>
}
```

- <identifier> is the name of the namespace.
- Namespaces ca be nested.

Namespaces (cont)

- Defining and accessing a namespace.

//main.cpp:

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

#include "Helper.h"

#include "globals.h"

int main(void)
{
  std::cout << DUMPVAR(CSCI390::pi) << std::endl;

  return 0;
}
```

//globals.h:

```
#include <cstdint>
namespace CSCI390
{
 extern double pi;
}
```

//globals.cpp:

```
#include "globals.h"
namespace CSCI390
{
  double pi{3.14159265358979323846264 33};
}
```

Console:

```
Variable: CSCI390::pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Namespaces (cont)

- Accessing a namespace.
  - **`using namespace`** <identifier>**`;`**
  - Defeats purpose of namespace.

//main.cpp:

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

#include "Helper.h"

#include "globals.h"

using namespace CSCI390;

int main(void)
{
  std::cout << DUMPVAR(pi) << std::endl;

  return 0;
}
```

//globals.h:

```
#include <cstdint>
namespace CSCI390
{
  extern double pi;
}
```

//globals.cpp:

```
#include "globals.h"
namespace CSCI390
{
  double pi{3.14159265358979323846264 33};
}
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Namespaces (cont)

- Accessing a namespace.
  - **`using`** <namespace>**`::`**<identifier>**`;`**
  - Safer than: **`using namespace`** <identifier>**`;`**

//main.cpp:

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

#include "Helper.h"

#include "globals.h"

using CSCI390::pi;

int main(void)
{
  std::cout << DUMPVAR(pi) << std::endl;

  return 0;
}
```

//globals.h:

```
#include <cstdint>
namespace CSCI390
{
  extern double pi;
}
```

//globals.cpp:

```
#include "globals.h"
namespace CSCI390
{
  double pi{3.14159265358979323846264 33};
}
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x6020b8, Value: 3.14159


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

Pointers

- Pointer are a <type>.
- They contain the address.
- Their address can change.
- Often confused with references, which they are NOT.
- Pointer syntax:
  <type> ∗<identifer>;
  - The ∗ is NOT an operator;
  - Create a (pointer) variable named <identifier>, which contains the address of which has a <type>.

Pointers (cont)

- The type **`size_t`** is automatically created and is the size of an address.

//main.cpp:

```
#include <iostream>

#include "Helper.h"

int main(void)
{

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

  return 0;
}
```

Console:

```
Type: unsigned long, Length: 8


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

Pointers (cont)

- Example.
  - **&** IS the unary operator “address of”.


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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};

  uint32_t *pi{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;

  return 0;
}
```

```
Console:
Variable: i, Type: unsigned int, Length: 4, Address: 0x7ffe658ebbac, Value: 23
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7ffe658ebbb8, Value: 0x7ffe658ebbac


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

Pointers (cont)

- You can “dereference” a pointer.
  - ∗ IS the unary operator “get contents of”.

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};

  uint32_t *pi{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;

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

  return 0;
}
```

```
Console:
Variable: i, Type: unsigned int, Length: 4, Address: 0x7ffcc8ac49fc, Value: 23
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7ffcc8ac4a08, Value: 0x7ffcc8ac49fc
Variable: *pi, Type: unsigned int, Length: 4, Address: 0x7ffcc8ac49fc, Value: 23


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

Pointers (cont)

- *<pointer> is an <lvalue>

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};

  uint32_t *pi{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;
  std::cout << DUMPVAR(*pi) << std::endl;

  *pi = 29u;

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;
  std::cout << DUMPVAR(*pi) << std::endl;

  return 0;
}
```

Console:

```
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff85a97e9c, Value: 23
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7fff85a97ea8, Value: 0x7fff85a97e9c
Variable: *pi, Type: unsigned int, Length: 4, Address: 0x7fff85a97e9c, Value: 23
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff85a97e9c, Value: 29
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7fff85a97ea8, Value: 0x7fff85a97e9c
Variable: *pi, Type: unsigned int, Length: 4, Address: 0x7fff85a97e9c, Value: 29


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

Pointers (cont)

- Pointers are a type and can be typedef'ed.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  typedef uint32_t *tpuint32;
  tpuint32 pi{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;
  std::cout << DUMPVAR(*pi) << std::endl;

  *pi = 29u;

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(pi) << std::endl;
  std::cout << DUMPVAR(*pi) << std::endl;

  return 0;
}
```

Console:

```
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff6793080c, Value: 23
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7fff67930818, Value: 0x7fff6793080c
Variable: *pi, Type: unsigned int, Length: 4, Address: 0x7fff6793080c, Value: 23
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff6793080c, Value: 29
Variable: pi, Type: unsigned int*, Length: 8, Address: 0x7fff67930818, Value: 0x7fff6793080c
Variable: *pi, Type: unsigned int, Length: 4, Address: 0x7fff6793080c, Value: 29


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

Pointers (cont)

- Pointers can point to different variables.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  uint32_t j{17u};

  typedef uint32_t *tpuint32;
  tpuint32 puint32{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  puint32 = &j;

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  return 0;
}
```

Console:

```
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d638, Value: 23
Variable: j, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d63c, Value: 17
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7fff9ac8d648, Value: 0x7fff9ac8d638
Variable: *puint32, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d638, Value: 23
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d638, Value: 23
Variable: j, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d63c, Value: 17
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7fff9ac8d648, Value: 0x7fff9ac8d63c
Variable: *puint32, Type: unsigned int, Length: 4, Address: 0x7fff9ac8d63c, Value: 17


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

Pointers (cont)

- You can prevent changes via the pointer.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  uint32_t j{19u};

  typedef const uint32_t *tpuint32;
  tpuint32 puint32{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  puint32 = &j;
  *puint32 = 11u;

  return 0;
}
```

Console:

```
main.cpp: In function 'int main()':
main.cpp:21:12: error: assignment of read-only location '* puint32'
   *puint32 = 11u;
            ^
```

Pointers (cont)

- You can prevent changes to the pointer.
  - Pointer must be initialized.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  uint32_t j{19u};

  typedef uint32_t * const tpuint32;
  tpuint32 puint32{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  *puint32 = 11u;
  puint32 = &j;

  return 0;
}
```

Console:

```
main.cpp: In function 'int main()':
main.cpp:21:11: error: assignment of read-only variable 'puint32'
   puint32 = &j;
           ^
```

Pointers (cont)

- Never dereference the nullptr.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  uint32_t j{17u};

  typedef uint32_t *tpuint32;
  tpuint32 puint32{&i};

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  puint32 = nullptr;
  std::cout << DUMPVAR(puint32) << std::endl;

  std::cout << "Goodbye world." << std::endl;

  *puint32 = 11u;

  return 0;
}
```

Console:

```
Variable: i, Type: unsigned int, Length: 4, Address: 0x7fffbd853b88, Value: 23
Variable: j, Type: unsigned int, Length: 4, Address: 0x7fffbd853b8c, Value: 17
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7fffbd853b98, Value: 0x7fffbd853b88
Variable: *puint32, Type: unsigned int, Length: 4, Address: 0x7fffbd853b88, Value: 23
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7fffbd853b98, Value: 0
Goodbye world.
Segmentation fault (core dumped)


...Program finished with exit code 139
Press ENTER to exit console.
```

Pointers (cont)

- Check for valid pointer before using.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  uint32_t i{23u};
  uint32_t j{17u};

  typedef uint32_t *tpuint32;
  tpuint32 puint32{&i};
  assert(puint32);

  std::cout << DUMPVAR(i) << std::endl;
  std::cout << DUMPVAR(j) << std::endl;
  std::cout << DUMPVAR(puint32) << std::endl;
  std::cout << DUMPVAR(*puint32) << std::endl;

  puint32 = nullptr;
  std::cout << DUMPVAR(puint32) << std::endl;

  std::cout << "Goodbye world." << std::endl;
  assert(puint32);
  *puint32 = 11u;

  return 0;
}
```

Console:

```
Variable: i, Type: unsigned int, Length: 4, Address: 0x7ffd28cfff28, Value: 23
Variable: j, Type: unsigned int, Length: 4, Address: 0x7ffd28cfff2c, Value: 17
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7ffd28cfff38, Value: 0x7ffd28cfff28
Variable: *puint32, Type: unsigned int, Length: 4, Address: 0x7ffd28cfff28, Value: 23
Variable: puint32, Type: unsigned int*, Length: 8, Address: 0x7ffd28cfff38, Value: 0
Goodbye world.
a.out: main.cpp:26: int main(): Assertion `puint32' failed.
Aborted (core dumped)


...Program finished with exit code 134
Press ENTER to exit console.
```

Pointers (cont)

- Generic pointer.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  void *GenericPointer{nullptr};
  std::cout << DUMPVAR(GenericPointer) << std::endl;

  return 0;
}
```

Console:

```
Variable: GenericPointer, Type: void*, Length: 8, Address: 0x7fff120b0388, Value: 0


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

Pointers (cont)

- You can cast pointers, but very risky.

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  typedef uint64_t *puint64;
  typedef double *pdouble;

  double pi{3.14159265358979323846264 33};

  pdouble pDoublePi{&pi};
  puint64 pIntPi;

  pIntPi = puint64(pDoublePi);

  std::cout << DUMPVAR(pi) << std::endl;
  std::cout << DUMPVAR(pDoublePi) << std::endl;
  std::cout << DUMPVAR(pIntPi) << std::endl;
  std::cout << DUMPVAR(*pDoublePi) << std::endl;
  std::cout << DUMPVAR(*pIntPi) << std::endl;

  return 0;
}
```

Console:

```
Variable: pi, Type: double, Length: 8, Address: 0x7ffd61760a78, Value: 3.14159
Variable: pDoublePi, Type: double*, Length: 8, Address: 0x7ffd61760a80, Value: 0x7ffd61760a78
Variable: pIntPi, Type: unsigned long*, Length: 8, Address: 0x7ffd61760a88, Value: 0x7ffd61760a78
Variable: *pDoublePi, Type: double, Length: 8, Address: 0x7ffd61760a78, Value: 3.14159
Variable: *pIntPi, Type: unsigned long, Length: 8, Address: 0x7ffd61760a78, Value: 4614256656552045848


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