CSCI 390 – Special Topics in C++

Lecture 6

9/6/18

Time To Turn Off Cell Phones

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.

Arrays
Multi-Dimensional Arrays

//main.cpp:

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

int main(void)
{
  typedef uint32_t tIntArray[2];
  tIntArray Array1{1u, 2u};
  tIntArray Array2[2]{{1u, 2u}, {3, 4}};
  uint32_t Array3[2][2]{{1u, 2u}, {3, 4}};

  std::cout << DUMPTYPE(Array1) << std::endl;
  std::cout << DUMPTYPE(Array2) << std::endl;
  std::cout << DUMPTYPE(Array3) << std::endl;
  std::cout << DUMPTYPE(Array3[0]) << std::endl;

  std::cout << DUMPVAR(Array1[0]) << std::endl;
  std::cout << DUMPVAR(Array1[1]) << std::endl;

  std::cout << DUMPVAR(Array2[0][0]) << std::endl;
  std::cout << DUMPVAR(Array2[0][1]) << std::endl;
  std::cout << DUMPVAR(Array2[1][0]) << std::endl;
  std::cout << DUMPVAR(Array2[1][1]) << std::endl;

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

  return 0;
}
```

Console:

```
Type: unsigned int [2], Length: 8
Type: unsigned int [2][2], Length: 16
Type: unsigned int [2][2], Length: 16
Type: unsigned int [2], Length: 8
Variable: Array1[0], Type: unsigned int, Length: 4, Address: 0x7ffe968286f0, Value: 1
Variable: Array1[1], Type: unsigned int, Length: 4, Address: 0x7ffe968286f4, Value: 2
Variable: Array2[0][0], Type: unsigned int, Length: 4, Address: 0x7ffe96828710, Value: 1
Variable: Array2[0][1], Type: unsigned int, Length: 4, Address: 0x7ffe96828714, Value: 2
Variable: Array2[1][0], Type: unsigned int, Length: 4, Address: 0x7ffe96828718, Value: 3
Variable: Array2[1][1], Type: unsigned int, Length: 4, Address: 0x7ffe9682871c, Value: 4
Variable: Array3[0][0], Type: unsigned int, Length: 4, Address: 0x7ffe96828720, Value: 1
Variable: Array3[0][1], Type: unsigned int, Length: 4, Address: 0x7ffe96828724, Value: 2
Variable: Array3[1][0], Type: unsigned int, Length: 4, Address: 0x7ffe96828728, Value: 3
Variable: Array3[1][1], Type: unsigned int, Length: 4, Address: 0x7ffe9682872c, Value: 4


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

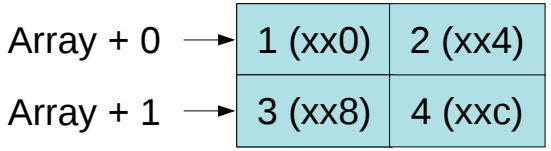
Arrays
Multi-Dimensional Arrays

//main.cpp:

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

int main(void)
{
  uint32_t Array[2][2]{{1u, 2u}, {3, 4}};

  std::cout << DUMPTYPE(Array) << std::endl;
  std::cout << DUMPVAL(Array) << std::endl;
  std::cout << DUMPVAL(Array + 0) << std::endl;
  std::cout << DUMPVAL(Array + 1) << std::endl;
  std::cout << DUMPVAL(*(Array + 1) + 1) << std::endl;
  std::cout << DUMPVAL(Array[1] + 1) << std::endl;
  std::cout << DUMPVAR(Array[1][1]) << std::endl;

  return 0;
}
```

Console:

```
Type: unsigned int [2][2], Length: 16
Expression: Array, Type: unsigned int [2][2], Length: 16, Value: 0x7ffd893bbca0
Expression: Array + 0, Type: unsigned int (*) [2], Length: 8, Value: 0x7ffd893bbca0
Expression: Array + 1, Type: unsigned int (*) [2], Length: 8, Value: 0x7ffd893bbca8
Expression: *(Array + 1) + 1, Type: unsigned int*, Length: 8, Value: 0x7ffd893bbcac
Expression: Array[1] + 1, Type: unsigned int*, Length: 8, Value: 0x7ffd893bbcac
Variable: Array[1][1], Type: unsigned int, Length: 4, Address: 0x7ffd893bbcac, Value: 4


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



Arrays
`const char` Array Constants

- Syntax: **"**<string>**"** [**"**<string>**"** …]
  - Same escape sequences as char constant:
    - `\' \" \? \\ \a \b \f \n \r \t \v`

//main.cpp:

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

int main(void)
{
  std::cout << DUMPVAL("Hello World!") << std::endl;
  std::cout << DUMPVAL("Hello" " " "World!") << std::endl;
  std::cout << DUMPVAL("Hello" " " "World!" + 1) << std::endl;
  std::cout << DUMPVAL("Hello" " " "World!"[1]) << std::endl;

  auto pHelloThere{"Hello" " " "World!"};
  char HelloThere[ ]{"Hello" " " "World!"};

  std::cout << DUMPVAL(pHelloThere) << std::endl;
  std::cout << DUMPVAL(HelloThere) << std::endl;

  return 0;
}
```

Console:

```
Expression: "Hello World!", Type: char [13], Length: 13, Value: Hello World!
Expression: "Hello" " " "World!", Type: char [13], Length: 13, Value: Hello World!
Expression: "Hello" " " "World!" + 1, Type: char const*, Length: 8, Value: ello World!
Expression: "Hello" " " "World!"[1], Type: char, Length: 1, Value: e
Expression: pHelloThere, Type: char const*, Length: 8, Value: Hello World!
Expression: HelloThere, Type: char [13], Length: 13, Value: Hello World!


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

Arrays
Getting The Size Of An Array

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  int32_t Array[3]{1, 2, 3};

  std::cout << "Array is " << sizeof(Array) / sizeof(*Array) <<
    " of " DUMPTYPE(*Array) <<
    ", Array: " DUMPTYPE(Array) << std::endl;
  std::cout << DUMPVAR(Array[0]) << std::endl;
  std::cout << DUMPVAR(Array[1]) << std::endl;
  std::cout << DUMPVAR(Array[2]) << std::endl;

  return 0;
}
```

Console:

```
Array is 3 of Type: int, Length: 4, Array: Type: int [3], Length: 12
Variable: Array[0], Type: int, Length: 4, Address: 0x7ffe2c915210, Value: 1
Variable: Array[1], Type: int, Length: 4, Address: 0x7ffe2c915214, Value: 2
Variable: Array[2], Type: int, Length: 4, Address: 0x7ffe2c915218, Value: 3


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

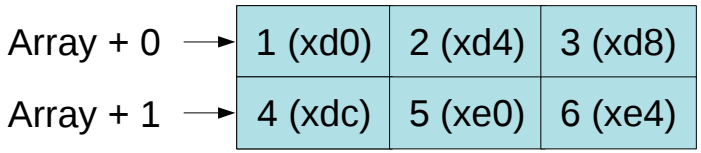
Arrays
Getting The Size Of An Array

//main.cpp:

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

#include "Helper.h"

int main(void)
{
  int32_t Array[2][3]{{1, 2, 3}, {4, 5, 6}};

  std::cout << "Array is " << sizeof(Array) / sizeof(*Array) <<
    " of " DUMPTYPE(*Array) <<
    ", Array: " DUMPTYPE(Array) << std::endl;
  std::cout << DUMPVAR(Array[0][0]) << std::endl;
  std::cout << DUMPVAR(Array[0][1]) << std::endl;
  std::cout << DUMPVAR(Array[0][2]) << std::endl;
  std::cout << DUMPVAR(Array[1][0]) << std::endl;
  std::cout << DUMPVAR(Array[1][1]) << std::endl;
  std::cout << DUMPVAR(Array[1][2]) << std::endl;

  return 0;
}
```

Console:

```
Array is 2 of Type: int [3], Length: 12, Array: Type: int [2][3], Length: 24
Variable: Array[0][0], Type: int, Length: 4, Address: 0x7fff600f9bd0, Value: 1
Variable: Array[0][1], Type: int, Length: 4, Address: 0x7fff600f9bd4, Value: 2
Variable: Array[0][2], Type: int, Length: 4, Address: 0x7fff600f9bd8, Value: 3
Variable: Array[1][0], Type: int, Length: 4, Address: 0x7fff600f9bdc, Value: 4
Variable: Array[1][1], Type: int, Length: 4, Address: 0x7fff600f9be0, Value: 5
Variable: Array[1][2], Type: int, Length: 4, Address: 0x7fff600f9be4, Value: 6


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



Pointers And Arrays
Tidying Up

- Pointer subtraction
  - The difference between two pointers is the signed number of typed elements between them.

//main.cpp:

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

int main(void)
{
  uint32_t Array[4];

  uint32_t *p0{Array};
  uint32_t *p3{Array + 3};

  std::cout << DUMPVAR(p0) << std::endl;
  std::cout << DUMPVAR(p3) << std::endl;
  std::cout << DUMPTYPE(p3 - p0) << std::endl;
  std::cout << DUMPVAL(p3 - p0) << std::endl;
  std::cout << DUMPVAL(p0 - p3) << std::endl;

  return 0;
}
```

Console:

```
Variable: p0, Type: unsigned int*, Length: 8, Address: 0x7ffc48723580, Value: 0x7ffc48723590
Variable: p3, Type: unsigned int*, Length: 8, Address: 0x7ffc48723588, Value: 0x7ffc4872359c
Type: long, Length: 8
Expression: p3 - p0, Type: long, Length: 8, Value: 3
Expression: p0 - p3, Type: long, Length: 8, Value: -3


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

Functions

- A function is a parameterized segment of code that optionally returns a value.
- It can be placed anywhere a global declaration can be placed. Like any global it can be prefixed with **`static`**, which limits scope to the current compilation unit.
  - You cannot declare a function inside a function.
- The parameters in the declaration are called *formal parameters*.

Functions (cont)

- A function can be invoked (called) by giving its name followed by the comma separated *actual parameters* enclosed in parentheses.
- At invocation, the actual parameters initialize the formal parameters.
- The **`return`** statement returns control to the calling program.
- Syntax of header:
  <type>|**`void`** <name>**`(`**<comma sep parms>|**`void`**`)`

Functions (cont)

- Function declarations have two parts: An interface declaration called a prototype, and an implementation.
  - Prototype declaration usually in a .h file, which clients include.
  - Implementation usually in a .cpp file.
- The actual and formal parameters must agree in type and have the same order.

Functions (cont)
Simple Example

//main.cpp:

```
#include <iostream>
#include "Function.h"

int main(void)
{
  std::cout << "Calling Hello():" << std::endl;
  Hello();
  std::cout << "Returned from Hello():" << std::endl;

  std::cout << "Calling Hello() again:" << std::endl;
  Hello();
  std::cout << "Returned from Hello():" << std::endl;

  return 0;
}
```

Console:

```
Calling Hello():
Hello World!
Returned from Hello():
Calling Hello() again:
Hello World!
Returned from Hello():


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

//Function.h:

```
void Hello(void);
```

//Function.cpp:

```
#include <iostream>
#include "Function.h"

void Hello(void)
{
  std::cout << "Hello World!" << std::endl;

  return;
}
```

Functions (cont)
Parameter Example

//main.cpp:

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

int main(void)
{
  uint32_t Number{2u};
  std::cout << DUMPVAR(Number) << std::endl;
  std::cout << "Calling ShowNum(Number):" << std::endl;
  ShowNum(Number);
  std::cout << "Returned from ShowNum(Number):" << std::endl;
  std::cout << DUMPVAR(Number) << std::endl;

  return 0;
}
```

Console:

```
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffceb37709c, Value: 2
Calling ShowNum(Number):
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffceb37704c, Value: 2
Number: 2
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffceb37704c, Value: 37
Returned from ShowNum(Number):
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffceb37709c, Value: 2


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

//Function.h:

```
#include <cstdint>

void ShowNum(uint32_t Number);
```

//Function.cpp:

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

void ShowNum(uint32_t Number)
{
  std::cout << DUMPVAR(Number) << std::endl;
  std::cout << "Number: " << Number << std::endl;
  Number = 37u;
  std::cout << DUMPVAR(Number) << std::endl;

  return;
}
```

Functions (cont)
Reference Example

//main.cpp:

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

int main(void)
{
  uint32_t Number{2u};
  std::cout << DUMPVAR(Number) << std::endl;
  std::cout << "Calling ShowNum(Number):" << std::endl;
  ShowNum(Number);
  std::cout << "Returned from ShowNum(Number):" << std::endl;
  std::cout << DUMPVAR(Number) << std::endl;

  return 0;
}
```

Console:

```
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffdbf15fcdc, Value: 2
Calling ShowNum(Number):
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffdbf15fcdc, Value: 2
Number: 2
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffdbf15fcdc, Value: 37
Returned from ShowNum(Number):
Variable: Number, Type: unsigned int, Length: 4, Address: 0x7ffdbf15fcdc, Value: 37


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

//Function.h:

```
#include <cstdint>

void ShowNum(uint32_t &Number);
```

//Function.cpp:

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

void ShowNum(uint32_t &Number)
{
  std::cout << DUMPVAR(Number) << std::endl;
  std::cout << "Number: " << Number << std::endl;
  Number = 37u;
  std::cout << DUMPVAR(Number) << std::endl;

  return;
}
```

Functions (cont)
Default Parameter Example

//main.cpp:

```
#include <iostream>
#include "Function.h"

int main(void)
{
  Message();
  Message("Good bye");

  return 0;
}
```

Console:

```
Hello
Good bye


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

//Function.h:

```
void Message(const char *Msg = "Hello");
```

//Function.cpp:

```
#include <iostream>
#include "Function.h"

void Message(const char *Msg)
{
  std::cout << Msg << std::endl;

  return;
}
```

Functions (cont)
Prototypes Are Declarations

//main.cpp:

```
#include <iostream>
#include "Function.h"

int main(void)
{
  Message("Hello");
  Message("Good bye");

  return 0;
}
```

Console:

```
Hello
Good bye


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

//Function.h:

```
// void Message(const char *Msg);

typedef void tMessage(const char *Msg);

tMessage Message;
```

//Function.cpp:

```
#include <iostream>
#include "Function.h"

void Message(const char *Msg)
{
  std::cout << Msg << std::endl;

  return;
}
```

Functions (cont)
Prototypes Are Declarations – Really

//main.cpp:

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

int main(void)
{
  std::cout << DUMPVAR(Dispatcher) << std::endl;
  std::cout << DUMPVAR(Dispatcher[Hello]) << std::endl;

  Dispatcher[Hello]();
  Dispatcher[Goodbye]();

  return 0;
}
```

Console:

```
Variable: Dispatcher, Type: void (* [2])(), Length: 16, Address: 0x6020e0, Value: 0x6020e0
Variable: Dispatcher[Hello], Type: void (*)(), Length: 8, Address: 0x6020e0, Value: 1
Hello
Good bye


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

//Function.h:

```
#include <cstdint>

enum eDispatch : uint32_t {Hello, Goodbye};

typedef void (*tMessage)(void);

extern tMessage Dispatcher[2];
```

//Function.cpp:

```
#include <iostream>
#include "Function.h"

static void Msg1(void)
{
  std::cout << "Hello" << std::endl;

  return;
}

static void Msg2(void)
{
  std::cout << "Good bye" << std::endl;

  return;
}
// Global declaration of Dispatcher.
tMessage Dispatcher[2]{Msg1, Msg2};
```