Outline

- In this lesson, we will:
  - Describe and review the call stack
  - See how the call stack is used to allocate memory for
    - Parameters
    - Local variables
  - Look at two examples in detail

Call stack

- Up to this point, we have seen that:
  - The local variables for `main()` have their memory allocated at the bottom of memory
    - We are calling this region the call stack
  - We also have functions that we may call, and those functions also have local variables

Review of functions

- Recall the behavior of a function:
  - A function is called from within another function
    - It is passed arguments
    - When it returns, it generally returns some value
  - You cannot `jump` into the middle of a function
  - Once a function returns:
    - You cannot go back to continue executing a function
    - You cannot access the parameters or any local variables
  - Functions may call other functions, and those functions may call others
Memory for functions

- Suppose the local variables of `main()` are stored at the bottom of memory
- When we call a function from `main()`, we must allocate memory for any
  - Parameters
  - Local variables
- The obvious location is immediately above the memory allocated for `main()`

Example: `gcd(...)`

- Consider the following:
  ```c++
  #include <iostream>
  // Function declarations
  int main();
  unsigned int gcd( unsigned int m, unsigned int n );
  // Function definitions
  int main() {
    unsigned int val1{42};
    unsigned int val2{91};
    std::cout << gcd( val1 + 10, val2 ) << std::endl;
    return 0;
  }
  unsigned int gcd( unsigned int m, unsigned int n ) {
    if ( m < n ) {
      unsigned int tmp{m};
      m = n;
      n = tmp;
    } // Function declarations
    if ( m == 0 ) {
      return 0;
    }
    unsigned int rem{m%n};
    while ( rem != 0 ) {
      m = n;
      n = rem;
      rem = m%n;
    }
    return n;
  }
  ```
- When executing `main()`, memory is allocated for the two local variables

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>main()</code></td>
<td><code>unsigned int val1</code> <code>unsigned int val2</code></td>
<td></td>
</tr>
<tr>
<td><code>gcd(...)</code></td>
<td><code>unsigned int m</code> <code>unsigned int n</code></td>
<td><code>unsigned int tmp</code> <code>unsigned int rem</code></td>
</tr>
</tbody>
</table>
Example: gcd(…)  

- When calling gcd(...), memory is allocated for the two parameters

```c++
int main() {
    unsigned int val1{42};
    unsigned int val2{91};
    std::cout << gcd( val1, val2 ) << std::endl;
    return 0;
}
```

Example: gcd(…)  

- The function gcd(...) has two local variables

```c++
unsigned int gcd( unsigned int m, unsigned int n ) {
    if ( m < n ) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    }
    if ( m == 0 ) {
        return n;
    }
    unsigned int rem = m%n;
    while ( rem != 0 ) {
        m = n;
        n = rem;
        rem = m%n;
    }
    return n;
}
```

Example: gcd(…)  

- The arguments are evaluated and the values of those arguments is copied to the parameters

```c++
int main() {
    unsigned int val1{42};
    unsigned int val2{91};
    std::cout << gcd( val1, val2 ) << std::endl;
    return 0;
}
```

Example: gcd(…)  

- The local variable tmp is used if \( m < n \), which is true

```c++
unsigned int gcd( unsigned int m, unsigned int n ) {
    if ( m == 0 ) {
        return n;
    }
    unsigned int rem = m%n;
    while ( rem != 0 ) {
        m = n;
        n = rem;
        rem = m%n;
    }
    return n;
}
```
Example: gcd(…)

- The local variable tmp is initialized with the value of m

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m == n) {
        return n;
    }
    unsigned int rem = m % n;
    while (rem != 0) {
        if (m < n) {
            n = rem;
        } else {
            m = rem;
        }
        rem = m % n;
    }
    return n;
}
```

Example: gcd(…)

- The local variable tmp is initialized with the value of m

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m == n) {
        return n;
    }
    unsigned int rem = m % n;
    while (rem != 0) {
        if (m < n) {
            n = rem;
        } else {
            m = rem;
        }
        rem = m % n;
    }
    return n;
}
```

Example: gcd(…)

- m is assigned the value of n

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m == n) {
        return n;
    }
    unsigned int rem = m % n;
    while (rem != 0) {
        if (m < n) {
            n = rem;
        } else {
            m = rem;
        }
        rem = m % n;
    }
    return n;
}
```

Example: gcd(…)

- m is assigned the value of n

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m == n) {
        return n;
    }
    unsigned int rem = m % n;
    while (rem != 0) {
        if (m < n) {
            n = rem;
        } else {
            m = rem;
        }
        rem = m % n;
    }
    return n;
}
```
Example: gcd(...)  

- n is assigned the value of tmp

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    } else if (m == 0) {
        return n;
    } else if (m == s) {
        return s;
    }  
    unsigned int rem = m % n;
    while (rem != 0) {
        m = n;
        n = rem;
        tmp = m % n;
    }
    return s;
```

Example: gcd(...)  

- The condition m == 0 is false, the consequent body is skipped

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    } else if (m == s) {
        return s;
    }  
    unsigned int rem = m % n;
    while (rem != 0) {
        m = n;
        n = rem;
        tmp = m % n;
    }
    return s;
```
The local variable `rem` is initialized with the value of `m%n`.

```c
unsigned int gcd(unsigned int m, unsigned int n) {  
  if (m < n) {  
    unsigned int tmp;  
    m = n;  
    n = tmp;  
  }  
  if (m == 0) {  
    return n;  
  }  
  unsigned int rem(m%n);  
  while (rem != 0) {  
    n = rem;  
    rem = m%n;  
  }  
  return n;  
}
```

Example: `gcd(...)`

- The condition `rem != 0` is true, the loop body is executed

```c
unsigned int gcd(unsigned int m, unsigned int n) {  
  if (m < n) {  
    unsigned int tmp;  
    m = n;  
    n = tmp;  
  }  
  if (m == 0) {  
    return n;  
  }  
  unsigned int rem(m%n);  
  while (rem != 0) {  
    n = rem;  
    rem = m%n;  
  }  
  return n;  
}
```

Example: `gcd(...)`

- `m` is assigned the value of `n`

```c
unsigned int gcd(unsigned int m, unsigned int n) {  
  if (m < n) {  
    unsigned int tmp;  
    m = n;  
    n = tmp;  
  }  
  if (m == 0) {  
    return n;  
  }  
  unsigned int rem(m%n);  
  while (rem != 0) {  
    n = rem;  
    rem = m%n;  
  }  
  return n;  
}
```

Example: `gcd(...)`

- `m` is assigned the value of `n`
Example: \texttt{gcd(...)}

- \texttt{n} is assigned the value of \texttt{rem}

```c
unsigned \texttt{gcd}(unsigned \texttt{m}, unsigned \texttt{n}) {  
if (m <= n) {  
  unsigned \texttt{int} \texttt{tmp} = \texttt{m};  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{tmp};  
}  
if (m == 0) {  
  return \texttt{n};  
}  
unsigned \texttt{int} \texttt{rem} = \texttt{m} % \texttt{n};  
while (\texttt{rem} != 0) {  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{rem};  
  \texttt{rem} = \texttt{m} % \texttt{n};  
}  
return \texttt{n};
```

Example: \texttt{gcd(...)}

- \texttt{rem} is assigned the value of \texttt{m} % \texttt{n}

```c
unsigned \texttt{gcd}(unsigned \texttt{m}, unsigned \texttt{n}) {  
if (m <= n) {  
  unsigned \texttt{int} \texttt{tmp} = \texttt{m};  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{tmp};  
}  
if (m == 0) {  
  return \texttt{n};  
}  
unsigned \texttt{int} \texttt{rem} = \texttt{m} % \texttt{n};  
while (\texttt{rem} != 0) {  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{rem};  
  \texttt{rem} = \texttt{m} % \texttt{n};  
}  
return \texttt{n};
```

Example: \texttt{gcd(...)}

- \texttt{n} is assigned the value of \texttt{rem}

```c
unsigned \texttt{gcd}(unsigned \texttt{m}, unsigned \texttt{n}) {  
if (m <= n) {  
  unsigned \texttt{int} \texttt{tmp} = \texttt{m};  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{tmp};  
}  
if (m == 0) {  
  return \texttt{n};  
}  
unsigned \texttt{int} \texttt{rem} = \texttt{m} % \texttt{n};  
while (\texttt{rem} != 0) {  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{rem};  
  \texttt{rem} = \texttt{m} % \texttt{n};  
}  
return \texttt{n};
```

Example: \texttt{gcd(...)}

- \texttt{rem} is assigned the value of \texttt{m} % \texttt{n}

```c
unsigned \texttt{gcd}(unsigned \texttt{m}, unsigned \texttt{n}) {  
if (m <= n) {  
  unsigned \texttt{int} \texttt{tmp} = \texttt{m};  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{tmp};  
}  
if (m == 0) {  
  return \texttt{n};  
}  
unsigned \texttt{int} \texttt{rem} = \texttt{m} % \texttt{n};  
while (\texttt{rem} != 0) {  
  \texttt{m} = \texttt{n};  
  \texttt{n} = \texttt{rem};  
  \texttt{rem} = \texttt{m} % \texttt{n};  
}  
return \texttt{n};
```
Example: gcd(…)

• The condition rem != 0 is true, the loop body is executed

unsigned int gcd(unsigned int m,
unsigned int n) {
if (m < n) {
unsigned int tmp(n);
 m = n;
 n = tmp;
}
if (m == 0) {
return n;
}
unsigned int rem(m);
while (rem != 0) {
 m = n;
 n = rem;
 rem = m%n;
}
return n;
}

Example: gcd(…)

• m is assigned the value of n

unsigned int gcd(unsigned int m,
unsigned int n) {
if (m < n) {
unsigned int tmp(n);
 m = n;
 n = tmp;
}
if (m == 0) {
return n;
}
unsigned int rem(m);
while (rem != 0) {
 m = n;
 n = rem;
 rem = m%n;
}
return n;
}

Example: gcd(…)

• n is assigned the value of rem

unsigned int gcd(unsigned int m,
unsigned int n) {
if (m < n) {
unsigned int tmp(n);
 m = n;
 n = tmp;
}
if (m == 0) {
return n;
}
unsigned int rem(m);
while (rem != 0) {
 m = n;
 n = rem;
 rem = m%n;
}
return n;
}
Example: gcd(…)

- n is assigned the value of rem

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    }
    while (rem != 0) {
        unsigned int rem = m%n;
        m = n;
        n = rem;
        if (rem == 0) {
            return n;
        }
    }
    return n;
}
```

Example: gcd(…)

- rem is assigned the value of m%n

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    }
    while (rem != 0) {
        unsigned int rem = m%n;
        m = n;
        n = rem;
        if (rem == 0) {
            return n;
        }
    }
    return n;
}
```

Example: gcd(…)

- rem is assigned the value of m%n

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    }
    while (rem != 0) {
        unsigned int rem = m%n;
        m = n;
        n = rem;
        if (rem == 0) {
            return n;
        }
    }
    return n;
}
```

Example: gcd(…)

- The condition rem != 0 is false, we skip the loop body and continue

```c
unsigned int gcd(unsigned int m, unsigned int n) {
    if (m < n) {
        unsigned int tmp = m;
        m = n;
        n = tmp;
    }
    while (rem != 0) {
        unsigned int rem = m%n;
        m = n;
        n = rem;
        if (rem == 0) {
            return n;
        }
    }
    return n;
}
```
We must now return the value \( n \).

Let's put the returned value at the bottom of the memory for the function \( \text{gcd} \).

The function \( \text{main}() \) can now access and use that returned value.

Example: \( \text{gcd}(\ldots) \)

```c
// Example: \text{gcd}(\ldots)

unsigned int \text{gcd}(\ldots) {
    unsigned int \text{tmp};
    \text{if} (m < n) {
        \text{m} = \text{m} + \text{tmp};
        \text{n} = \text{n} + \text{tmp};
    }\text{else if} (m == 0) {
        \text{return} \text{n};
    }\text{unsigned int rem}(\text{m}, \text{n});
    \text{while} ( \text{rem} != 0 ) {
        \text{m} = \text{m} + \text{rem};
        \text{n} = \text{n} + \text{rem};
    }\text{return} \text{n};
}
```

Example: \( \text{gcd}(\ldots) \)

```c
// Example: \text{gcd}(\ldots)

int \text{main}() {
    unsigned int \text{val1}(42);
    unsigned int \text{val2}(91);
    \text{int} \text{tmp} = \text{gcd}(\text{val1}, \text{val2});
    \text{\text{std::cout}} << \text{\text{\text{\text{\text{std::endl}}}}}
    \text{\text{\text{\text{\text{return}}}0};
}
```

Example: \( \text{gcd}(\ldots) \)

```c
// Example: \text{gcd}(\ldots)

int \text{main}() {
    unsigned int \text{val1}(42);
    unsigned int \text{val2}(91);
    \text{int} \text{tmp} = \text{gcd}(\text{val1}, \text{val2});
    \text{\text{std::cout}} << \text{\text{\text{\text{\text{std::endl}}}}}
    \text{\text{\text{\text{\text{return}}}0};
}
```

Example: \( \text{gcd}(\ldots) \)

```c
// Example: \text{gcd}(\ldots)

int \text{main}() {
    unsigned int \text{val1}(42);
    unsigned int \text{val2}(91);
    \text{int} \text{tmp} = \text{gcd}(\text{val1}, \text{val2});
    \text{\text{std::cout}} << \text{\text{\text{\text{\text{std::endl}}}}}
    \text{\text{\text{\text{\text{return}}}0};
}
```
Example: gcd(…)

- A few observations:
  - The scopes of tmp and rem do not overlap, so almost all compilers would use the same memory location for each
  - Much more information must be put on the stack

Example: gcd(…)

Example: is_prime(…)

- Consider the following:

```cpp
#include <iostream>

// Function declarations
int main();
int nprimes(int n);

// Function definitions
int main() {
    int num;
    std::cout << "Enter a number: ";
    std::cin >> num;
    std::cout << nprimes(num) << std::endl;
    return 0;
}

int nprimes(int n) {
    if (n <= 1) {
        return 0;
    }
    assert(n >= 2);
    // 0 and 1 are not prime, 2 is prime
    // assumed to be not prime
    bool is_prime[n + 1]{false, false, true};
    // Assume all odd numbers >= 3 are prime
    // all multiples of 2 are not prime
    for (int k{3}; k <= n; k += 2) {
        is_prime[k] = true;
    }
    // Looking at the odd numbers,
    // if it is prime, flag all multiples of it
    // to be not prime
    for (int k{3}; k <= n; k += 2) {
        if (is_prime[k]) {
            for (int m{k}; m*k <= n; m += 2) {
                is_prime[m*k] = false;
            }
        }
    }
    // Count all the prime numbers
    // and return that value
    int count{0};
    for (int k{2}; k <= n; ++k) {
        if (is_prime[k]) {
            ++count;
        }
    }
    return count;
}
```

Example: is_prime(…)

- Let's tabulate the information:

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>main()</td>
<td>int num</td>
<td></td>
</tr>
<tr>
<td>nprimes(…)</td>
<td>int n</td>
<td>int k, int k, int m, int count, bool is_prime[n + 1]</td>
</tr>
</tbody>
</table>
Example: is_prime(...) • The local variable num is initialized with the value of 0

```cpp
int main() {
    int num[];
    std::cout << "Enter a number: ";
    std::cin >> num;
    std::cout << nprimes( num ) << std::endl;
    return 0;
}
```

Example: is_prime(...) • Suppose the user enters the value 10

```cpp
int main() {
    int num[];
    std::cout << "Enter a number: ";
    std::cin >> num;
    std::cout << nprimes( num ) << std::endl;
    return 0;
}
```

Example: is_prime(...) • We call nprimes(…) which has one parameter n

```cpp
int main() {
    int num[];
    std::cout << "Enter a number: ";
    std::cin >> num;
    std::cout << nprimes( num ) << std::endl;
    return 0;
}
```

Example: is_prime(...) • The value of the argument is copied to the memory for the parameter n

```cpp
int main() {
    int num[];
    std::cout << "Enter a number: ";
    std::cin >> num;
    std::cout << nprimes( num ) << std::endl;
    return 0;
}
```
Understand this memory is used for parameters, local variables, and return values.

Have observed two examples of programs using the call stack.

The call stack starts at the bottom of memory.

The local array is of type int allocated for location immediately above that memory.

When the function is ready to return, the value returned will be copied to the location immediately above that memory allocated for main().

Following this lesson, you now
- Have a basic understanding of the call stack
- Understand this memory is used for parameters, local variables, and return values
- Have observed two examples of programs using the call stack
References


Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using Consolas.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see https://www.rbg.ca/ for more information.

Disclaimer

These slides are provided for the ECE 150 Fundamentals of Programming course taught at the University of Waterloo. The material in it reflects the authors’ best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.