Anatomy of a program

Pre-processor directives

```
#include <iostream>  // An include directive

int main();

int main() {
    std::cout << "Hello world!";
    std::cout << std::endl;
    return 0;
}
```

Outline

- In this presentation, we will:
  - Define the components of a program
    - Pre-processor directives
    - Statements
    - Blocks of statements
    - Function declarations and definitions

Pre-processor directives

- Include directives indicates that a particular file should be included in the compilation
  - C++ Standard libraries contain functionality available to all programmers except perhaps in embedded systems
  - Possible to include other source code you or others have written
    ```
    #include <iostream>
    
    #include <cmath>
    http://www.cplusplus.com/reference/
    ```

- All pre-processor directives start with a "#"
  - Another example is a define directive:
    ```
    #define N_DEBUG
    ```
Statements

• Some sample statements from the NASA core Flight System Memory Manager Application:

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!";
    std::cout << std::endl;
    return 0;
}
```

Statements

• A statement may always be described as
  – The introduction (or "declaration") of a name (or "identifier")
  – An action that is being performed on data

```cpp
int main()
{
    std::cout << "Hello world!";
    std::cout << std::endl;
    return 0;
}
```

  – A statement is always terminated by a semi-colon

A block of statements

• A block of statements is zero or more statements surrounded by braces
  – Statements are executed one at a time in the order they appear
  – One statement must finish executing before the next starts

```cpp
{
    std::cout << "Hello world!";
    std::cout << std::endl;
    return 0;
}
```
A block of statements

• A sample block of statements from the NASA core Flight System Memory Manager Application:

```cpp
{  
  Valid = TRUE;
  MM_AppData.LastAction = MM_LOAD_FROM_FILE;
  MM_AppData.MemType = FileHeader->MemType;
  MM_AppData.Address = DestAddress;
  MM_AppData.BytesProcessed = BytesProcessed;
  strncpy( MM_AppData.FileName, FileName, OS_MAX_PATH_LEN );
}
```

Function declarations and definitions

```cpp
#include <iostream>

int main();  // Function declaration

int main() {  // Function definition
  std::cout << "Hello world!";
  std::cout << std::endl;
  return 0;
}
```

The statements executed when the function is called is also referred to as the function body.

Function declarations

• From your secondary school mathematics courses, you have seen:

\[
\sin \left( \frac{\pi}{6} \right) = \gcd(91, 119)
\]

• The names of these functions are sin and gcd:
  – sin has one real parameter and evaluates to a real
  – gcd has two integer parameters and evaluates to an integer

Function declarations

• A function declaration indicates to the compiler:
  – That a function with a specific name exists
    • The name is called the identifier of the function
  – The function's parameters
  – What it returns

• The declaration:

```cpp
int main();
```

indicates:
  – The name of the function is main
  – It has no parameters
  – It returns an int
    • An integer
Anatomy of a program

**Function declarations**

- The function declaration for gcd would be:
  ```
  int gcd(int m, int n);
  ```
  - The name of the function is gcd
  - It has two parameters, both of which are integers
  - It returns an integer

- The function declaration for sin would be:
  ```
  double sin(double x);
  ```
  - The name of the function is sin
  - It has one parameter, a representation of a real number
  - It returns a representation of a real number

**Function definition**

- A function definition is the function declaration immediately followed by a block of statements
  - This block of statements is also called the body of the function
  - These statements are executed when the function is run
  - The three statements executed when main() is called include:
    - Printing "Hello world!" to the console output
    - Printing an end-of-line character to the console output
    - Returning the value 0

```
int main() {
  std::cout << "Hello world!";
  std::cout << std::endl;
  return 0;
}
```

**Function definition**

- The main function is especially important in C++
  - There can by many functions, but if source code is compiled into an executable, when that executable is run, it is the main() function immediately called

**Symbols and identifiers**

- Apart from the pre-processor directives, a program is made up of:
  - Literal data
  - Identifiers
  - Operators
  - Delimiters
  - Literal data are numbers, characters and text that appears in the program
  - We have seen 0 and "Hello world!"
  - Identifiers are names like main, sin and gcd
  - It also std::cout, std::endl and return
  - The only way to manipulate data is with operators:
    ```
    ! % & && * ~ + = += - = *= | || ?: , .
    ```
  - The pairs (~) [ ] { } < > group expressions
Anatomy of a program

Matching delimiters

• As in your mathematics courses, ( ), [ ], and { } are used to group mathematical expressions
  (, [, and { are referred to as opening delimiters
  ), ], and } are referred to as closing delimiters

• These delimiters always come in pairs
  – The opening delimiter must always come before the matching closing delimiter

• You may recall that you cannot mix delimiters:
  – Valid: {3 + [4 × 5 + 6 × (7 – 8)] + 5}
  – Invalid: )3 + [4 × 5 + 6 × (7 – 8)] ÷ 5{ + [}

References

https://en.wikipedia.org/wiki/Statement_(computer_science)

Summary

• In this presentation, you now
  – Described the include pre-processor directive
  – Defined a statement
    • Operators and function calls terminated by a semi-colon
  – Defined a block of statements
    • Zero or more statements surrounded by braces
  – Defined function declarations and definitions
  – Described symbols (operators and delimiters) and identifiers

Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.
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.