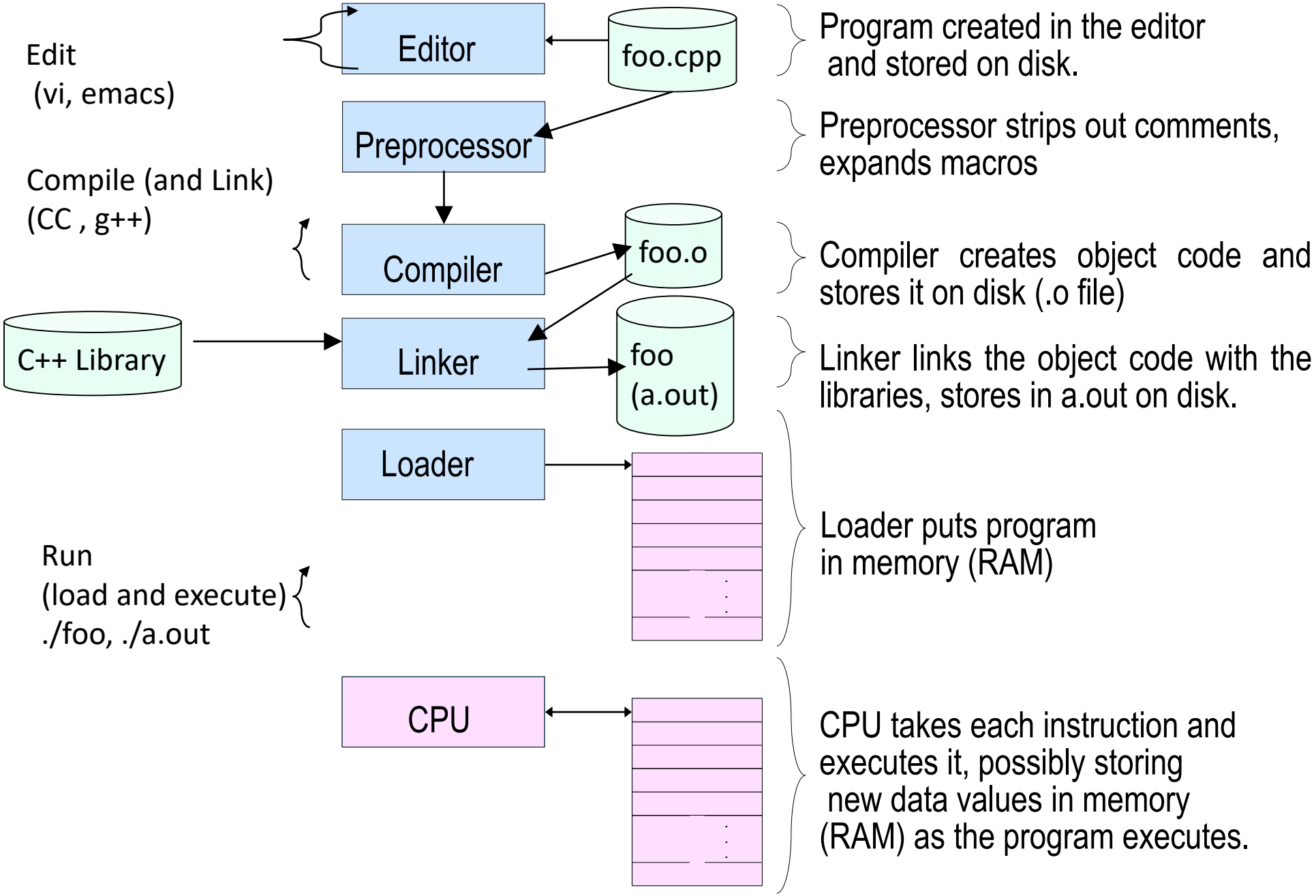
Introduction to C++

Basics of a Typical C++ Environment

- C++ Language definition
- Program-development environment (tools)
  - compiler, linker, editor, debugger
- C++ Standard Library (software)
  - precompiled routines you can use

Basics of a Typical C++ Environment



The Typical Structure of a C++ program

```
//include headers; these are modules that include
  functions that you may use in your
 //program; we will almost always need to include
  the header that
// defines cin and cout; the header is called
  iostream.h
```

```
#include <iostream.h>
int main() {
//variable declaration
//read values input from user
//computation and print output to user
return 0;
}
```

After you write a C++ program you compile it; that is, you run a program called **compiler** that checks whether the program follows the C++ syntax

if it finds errors, it lists them

If there are no errors, it translates the C++ program into a program in machine language which you can execute

Notes

- what follows after **//** on the same line is considered comment
- indentation is for the convenience of the reader; compiler ignores all spaces and new line ; the delimiter for the compiler is the semicolon
- all statements ended by semicolon
- **Lower vs. upper case matters!!**
  - Void is different than void
  - Main is different that main

The infamous **Hello world** program

When learning a new language, the first program people usually write is one that salutes the world :)

Here is the Hello world program in C++.

```
#include <iostream.h>
int main() {
 cout << “Hello world!”;
  return 0;
}
```

A Simple Program: Printing a Line of Text

A Simple Program: Printing a Line of Text

```
// abc.cpp
// A first program in C++.
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome to C++!\n";

   return 0;   // indicate that program ended successfully

} // end function main
```

```
Welcome to C++!
```

A Simple Program: Printing a Line of Text

- Standard output stream object
  - **`std::cout`**
  - "Connected" to screen
  - **`<<`**
    - Stream insertion operator
    - Value to right (right operand) inserted into output stream
- Namespace
  - **`std::`** specifies using name that belongs to "namespace" **`std`**
  - **`std::`** removed through use of **`using`** statements
- Escape characters
  - `\`
  - Indicates "special" character output

A Simple Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double quote character.

A Simple Program: Printing a Line of Text

```
// xyz.cpp
// Printing a line with multiple statements.
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome ";
   std::cout << "to C++!\n";

   return 0;   // indicate that program ended successfully

} // end function main
```

Multiple stream insertion statements produce one line of output.

```
Welcome to C++!
```

A Simple Program: Printing a Line of Text

```
// Fig. 1.5: fig01_05.cpp
// Printing multiple lines with a single statement
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";

   return 0;   // indicate that program ended successfully

} // end function main
```

Using newline characters to print on multiple lines.

```
Welcome
to

C++!
```

Another Simple Program: Adding Two Integers

- Variables
  - Location in memory where value can be stored
  - Common data types
    - **`int`** - integer numbers
    - **`char`** - characters
    - **`double`** - floating point numbers
  - Declare variables with name and data type before use

```
int integer1;
int integer2;
int sum;
```

  - Can declare several variables of same type in one declaration
    - Comma-separated list

```
int integer1, integer2, sum;
```

Another Simple Program: Adding Two Integers

- Variables
  - Variable names
    - Valid identifier
      - Series of characters (letters, digits, underscores)
      - Cannot begin with digit
      - Case sensitive

Another Simple Program: Adding Two Integers

- Input stream object
  - **`>>`** (stream extraction operator)
    - Used with **`std::cin`**
    - Waits for user to input value, then press *Enter* (Return) key
    - Stores value in variable to right of operator
      - Converts value to variable data type
- **`=`** (assignment operator)
  - Assigns value to variable
  - Binary operator (two operands)
  - Example:

```
sum = variable1 + variable2;
```

```cpp
// Fig. 1.6: fig01_06.cpp
// Addition program.
#include <iostream>

// function main begins program execution
int main()
{
   int integer1;  // first number to be input by user
   int integer2;  // second number to be input by user
   int sum;       // variable in which sum will be stored

   std::cout << "Enter first integer\n";  // prompt
   std::cin >> integer1;                  // read an integer

   std::cout << "Enter second integer\n"; // prompt
   std::cin >> integer2;                  // read an integer

   sum = integer1 + integer2;  // assign result to sum

   std::cout << "Sum is " << sum << std::endl; // print sum

   return 0;   // indicate that program ended successfully

} // end function main
```

Variable declaration

`type variable-name;`

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

- `int` `//integer`
- `double` `//real number`
- `char` `//character`

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

Input statements

```
cin >> variable-name;
```

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-character;
```

Output statements

```
cout << variable-name;
```

Meaning: print the value of variable <variable-name> to the user

```
cout << "any message ";
```

Meaning: print the message within quotes to the user

```
cout << endl;
```

Meaning: print a new line

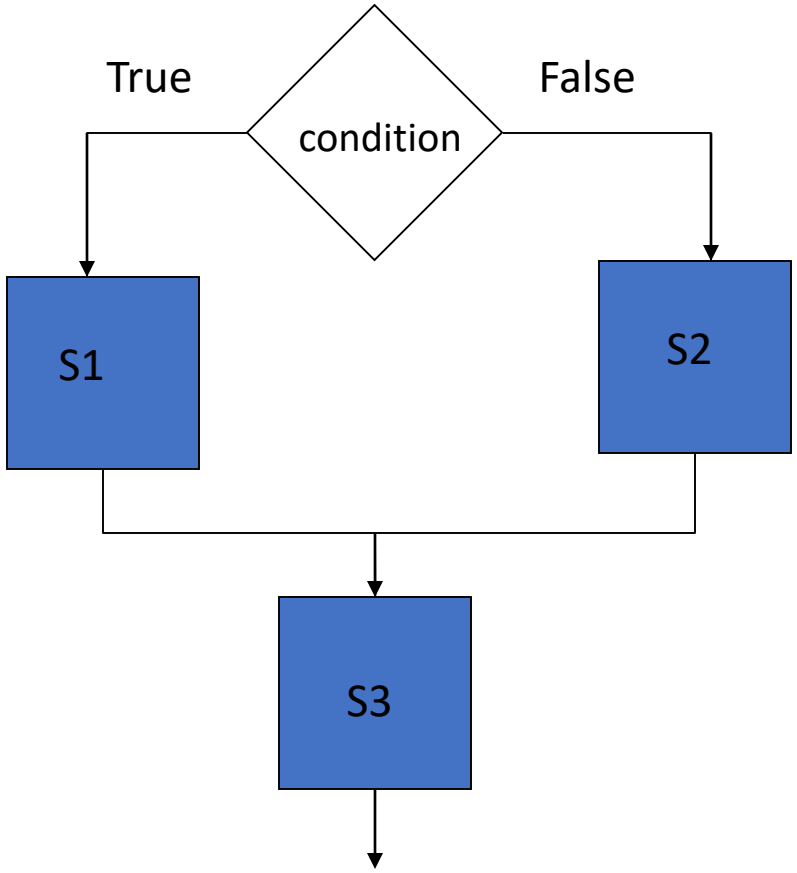
Example:

```
cout << a;
cout << b << c;
cout << "This is my character: " << my-character
<< " he he he"
     << endl;
```

Conditional Statements

If statements

```
if (condition) {
    S1;
 }
 else {
    S2;
 }
 S3;
```



Boolean conditions

..are built using

- Comparison operators

==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

- Boolean operators

&&	and
||	or
!	not

Examples

Assume we declared the following variables:

```
int a = 2, b=5, c=10;
```

Here are some examples of boolean conditions we can use:

- `if (a == b) …`
- `if (a != b) …`
- `if (a <= b+c) …`
- `if(a <= b) && (b <= c) …`
- `if !((a < b) && (b<c)) …`

If example

```
#include <iostream.h>

void main() {
int a,b,c;
cin >> a >> b >> c;

if (a <=b) {
   cout << “min is “ << a << endl;
   }
else {
   cout << “ min is “ << b << endl;
}
cout << “happy now?” << endl;
}
```

Iterative Statements

OVERVIEW

OVERVIEW

- We often need to do repetitive calculations in order to solve complex problems
- To perform repetitive calculations in a program we need <u>iterative statements</u> that let us execute the same block of code multiple times
- C++ has three kinds of iterative statements
  - The while loop
  - The for loop
  - The do-while loop

The for loop

Example 1: Printing Numbers From 1 to 5

```
#include <iostream>

using namespace std;

int main() {
      for (int i = 1; i <= 5; ++i) {
      cout << i << " ";
   }
   return 0;
}
```

Example 2: Display a text 5 times

```
// C++ Program to display a text 5 times

#include <iostream>

using namespace std;

int main() {
   for (int i = 1; i <= 5; ++i) {
      cout <<  "Hello World! " << endl;
   }
   return 0;
}
```

Example 3: Find the sum of first n Natural Numbers

```
// C++ program to find the sum of first n natural numbers
// positive integers such as 1,2,3,...n are known as natural numbers

#include <iostream>

using namespace std;

int main() {
   int num, sum;
   sum = 0;

   cout << "Enter a positive integer: ";
   cin >> num;

   for (int i = 1; i <= num; ++i) {
      sum += i;
   }

   cout << "Sum = " << sum << endl;

   return 0;
}
```

The while loop

Example 1: Printing Numbers From 1 to 5

```
// C++ Program to print numbers from
1 to 5

#include <iostream>

using namespace std;

int main() {
   int i = 1;

   // while loop from 1 to 5
   while (i <= 5) {
      cout << i << " ";
      ++i;
   }

   return 0;
}
```

Example 2: Display a text 5 times

```
// program to find the sum of positive numbers
// if the user enters a negative number, the loop ends
// the negative number entered is not added to the sum

#include <iostream>
using namespace std;

int main() {
   int number;
   int sum = 0;

   // take input from the user
   cout << "Enter a number: ";
   cin >> number;

   while (number >= 0) {
      // add all positive numbers
      sum += number;

      // take input again if the number is positive
      cout << "Enter a number: ";
      cin >> number;
   }

   // display the sum
   cout << "\nThe sum is " << sum << endl;

   return 0;
}
```

The do-while loop

Example 1: Printing Numbers From 1 to 5

```
// C++ Program to print numbers from
1 to 5

#include <iostream>

using namespace std;

int main() {
   int i = 1;

   // do...while loop from 1 to 5
   do {
      cout << i << " ";
      ++i;
   }
   while (i <= 5);

   return 0;
}
```

Example 2: Display a text 5 times

```
// program to find the sum of positive numbers
// If the user enters a negative number, the loop ends
// the negative number entered is not added to the sum

#include <iostream>
using namespace std;

int main() {
   int number = 0;
   int sum = 0;

   do {
      sum += number;

      // take input from the user
      cout << "Enter a number: ";
      cin >> number;
   }
   while (number >= 0);

   // display the sum
   cout << "\nThe sum is " << sum << endl;

   return 0;
}
```