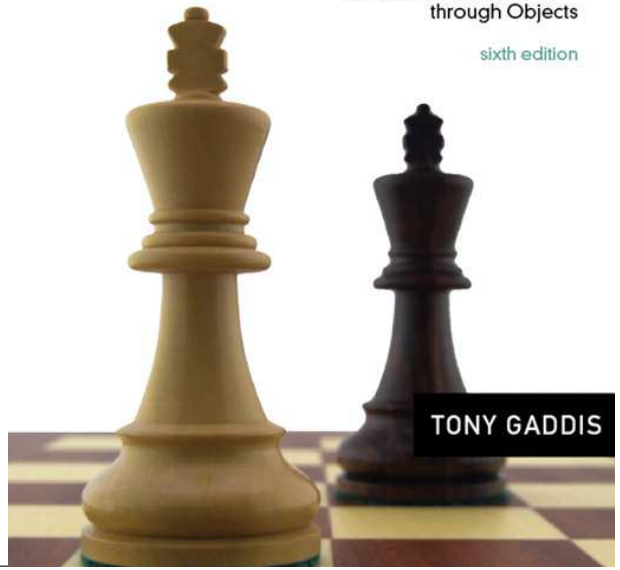
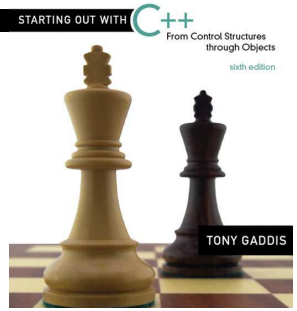


Chapter 2:

Introduction to C++



2.1

Parts of a C++ Program

Parts of a C++ Program

```
// sample C++ program
#include <iostream>
using namespace std;
int main()
{
     cout << "Hello, there!";
     return 0;
}
```

comment

preprocessor directive

which namespace to use

beginning of function named main

beginning of block for main

output statement

string literal

send 0 to operating system

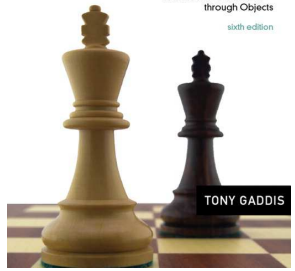
end of block for main

Parts of a C++ Program

- **Comments**: Explain about the program, can appear anywhere, and are ignored by the compiler
- **Directives**: commands to the preprocessor to set up your source code for the compiler.
- **using name space**: Specifies where the names of the variables, functions, and other entities are declared
- **int main()**: the beginning of the main function; The instructions are included within the pair of curly braces. It returns an integer.
- *Note: C++ is case sensitive.*

Special Characters

Character	Name	Meaning
`//`	Double slash	Beginning of a comment
`#`	Pound sign	Beginning of preprocessor directive
`< >`	Open/close brackets	Enclose filename in #include
`( )`	Open/close parentheses	Used when naming a function
`{ }`	Open/close brace	Encloses a group of statements
`" "`	Open/close quotation marks	Encloses string of characters
`;`	Semicolon	End of a programming statement



2.2

The `cout` Object

The `cout` Object

- Represents the standard output stream, which is set to the console in most systems by default
- Displays output on the computer screen
- You use the *stream insertion operator* << to send output to `cout`:

```
cout << "Programming is fun!";
```


The `cout` Object

- Can be used to send more than one item to *cout*:

```
cout << "Hello " << "there!";
```

Or:

```
cout << "Hello ";
cout << "there!";
```

Both produce the same output as:

```
cout << "Hello there!";
```

The `cout` Object

- What if we need to display something in different lines?
- This produces one line of output:

```
cout << "Programming is ";
cout << "fun!";
```

- *cout* does not start a new line unless told to do so

The `endl` Manipulator

- You can use the ***endl*** *manipulator* to start a new line of output. This will produce two lines of output:

```
cout << "Programming is" << endl;
cout << "fun!";
```


The `endl` Manipulator

```
cout << "Programming is" << endl;
cout << "fun!";
```

The `endl` Manipulator

- You do NOT put quotation marks around **`endl`**
- The last character in **`endl`** is a lowercase L, not the number 1.

`endl`

This is a lowercase L

The \n Escape Sequence

- You can also use the ***\n*** *escape sequence* to start a new line of output. This will produce two lines of output:

```
cout << "Programming is\n";
cout << "fun!";
```

Notice that the \n is INSIDE the string. Do not use forward slash (/n).

The \n Escape Sequence

```
cout << "Programming is\n";
cout << "fun!";
```

Common Escape Sequences

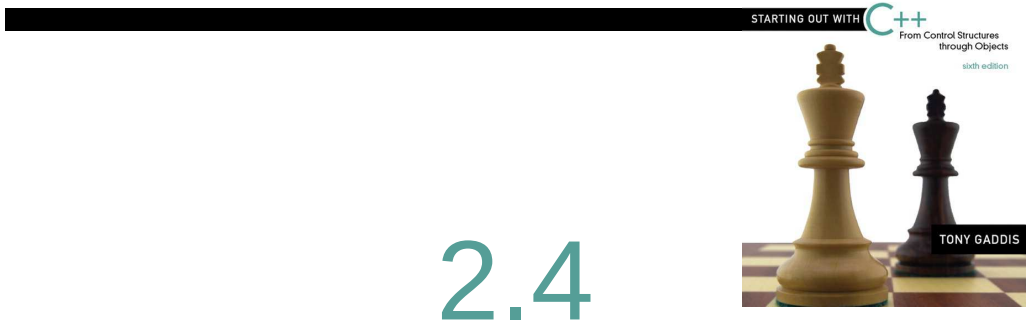
Escape Sequence	Name	Description
\n	Newline	Moves the cursor to the beginning of the next line.
\t	Tab	Moves the cursor to the next tab stop.
\a	Alarm	Causes the computer to beep.
\b	Backspace	Moves the cursor left one position.
\r	Return	Moves the cursor to the beginning of current line.
\\	Backslash	Print a backslash.
\'	Single quote	Print a single quotation mark.
\"	Double quote	Print a double quotation mark.

2.3

The #include Directive

The `#include` Directive

- Inserts the contents of another file into the program.
  - *#include <iostream>* – inserts the **iostream** *header file*, which contains information about *cout* and is needed for the compilation of programs using *cout* object
- This is a preprocessor directive, not part of C++ language
- `#include` lines will not be seen by the compiler
- Do not place a semicolon at end of `#include` line



2.4

Variables and Literals

Variables and Literals

- The program uses variables to store data and literals to represent data
- Variable: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:

```
int item;
```


Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int number;
8
9     number = 5;
10    cout << "The value in number is " << number << endl;
11    return 0;
12 }
```

Variable Definition

Program Output

```
The value in number is 5
```

Literals

- Literal: a value that is written into a program's code.

`"hello, there"`	(string literal)
`12`	(integer literal)
`"number"`	(string literal)
`"12"`	(string literal)

Note: integer 12 is different from string "12".

Program 2-9

```
1  // This program has literals and a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

20 is an integer literal

Program Output

```
Today we sold 20 bushels of apples.
```


Program 2-9

```
1  // This program has literals and a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

This is a string literal

Program Output

```
Today we sold 20 bushels of apples.
```


Program 2-9

```
1  // This program has literals and a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

This is also a string literal

Program Output

```
Today we sold 20 bushels of apples.
```

Literals

- What will the following program segment print out?

```
number = 20;
cout << "number = " << "number";
```

- Is the following program segment correct?

```
int grade;
grade = "80";
```


2.5

Identifiers

Identifiers

- An identifier is a programmer-defined name for some element of a program: variables, functions, etc.

C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

Table 2-4 The C++ Key Words (total=74)

and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Variable Names

- A variable name should represent the purpose of the variable. For example, instead of **`x`**, use:

```
itemsOrdered
```

The purpose of this variable is to hold the number of items ordered.

Identifier Rules

- The first character of an identifier must be an alphabetic character or an underscore (_),
- After the first character you may use alphabetic characters, numbers, or underscore characters.
- Upper- and lowercase characters are distinct

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_Sales`	Yes	
`total.Sales`	No	Cannot contain **.**
`4thQtrSales`	No	Cannot begin with a digit
`totalSale$`	No	Cannot contain $

2.6

Integer Data Types

Integer Data Types

- There are many different types of data
- Integer variables can hold whole numbers such as 12, 7, and -99.
- Integer data type is defined using
  - `int`
  - `unsigned int`
- There are other integer data types, each representing integers with different range of values.

Type Name	Bytes	Range of Values
int	4	–2,147,483,648 to 2,147,483,647
unsigned int	4	0 to 4,294,967,295
__int8	1	–128 to 127
unsigned __int8	1	0 to 255
__int16	2	–32,768 to 32,767
unsigned __int16	2	0 to 65,535
__int32	4	–2,147,483,648 to 2,147,483,647
unsigned __int32	4	0 to 4,294,967,295
__int64	8	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned __int64	8	0 to 18,446,744,073,709,551,615

Defining Variables

- Variables of the same type can be defined
  - On separate lines:

```
int length;
int width;
unsigned int area;
```

  - On the same line:

```
int length, width;
unsigned int area;
```

- Variables of different types must be in different definitions

Program 2-10

```
// This program has variables of several of the integer types.
#include <iostream>
using namespace std;

int main()
{
   int checking;
   unsigned int miles;
   long days;

   checking = -20;
   miles = 4276;
   days = 189000;
   cout << "We have made a long journey of " << miles;
   cout << " miles.\n";
   cout << "Our checking account balance is " << checking;
   cout << "\nAbout " << days << " days ago Columbus ";
   cout << "stood on this spot.\n";
   return 0;
}
```

This program has three variables: checking, miles, and days

Integer Literals

- An integer literal is an integer value that does not contain a decimal point. For example:

```
itemsOrdered = 15;
```

In this code, 15 is an integer literal.

Program 2-10

```
// This program has variables of several of the integer types.
#include <iostream>
using namespace std;

int main()
{
   int checking;
   unsigned int miles;
   long days;

   checking = -20;
   miles = 4276;
   days = 109000;
   cout << "We have made a long journey of " << miles;
   cout << " miles.\n";
   cout << "Our checking account balance is " << checking;
   cout << "\nAbout " << days << " days ago Columbus ";
   cout << "stood on this spot.\n";
   return 0;
}
```

Integer Literals

Integer Literals

- Integer literals are stored in memory as base 10 `ints` by default
- C++ allows integers to be represented in other bases
- Constants that begin with '`0`' (zero) are base 8: `075`
- Constants that begin with '`0x`' are base 16: `0x75A`

2.7

The char Data Type

The char Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:
```
char letter;
letter = 'C';
```

MEMORY:
letter

67

⇒ 01000011

Character Literals

- Character literals must be enclosed in single quote marks. Example:

```
char ans;
ans = 'Y';      // correct
ans = "Y";      // wrong
```


Program 2-13

```
// This program uses character literals.
#include <iostream>
using namespace std;

int main()
{
   char letter;

   letter = 'A';
   cout << letter << endl;
   letter = 'B';
   cout << letter << endl;
   return 0;
}
```

Program Output

```
A
B
```


Character Strings

- A series of characters in consecutive memory locations:
  `"Hello"`
- Stored with the <u>null terminator</u>, `\0`, at the end:
- Comprised of the characters between the `"  "`

H	e	l	l	o	\0

Character vs. String

- 'A' is stored as | A |
- "A" is stored as | A | \0 |

```
char letter;
letter = 'A';      // This is OK
letter = "A";      // This will not work
```


The `string` Data Type

- Allows a sequence of characters to be stored in one memory location
- Actually a class and not a true data type built into the language
- Needs to include `<string>` header file

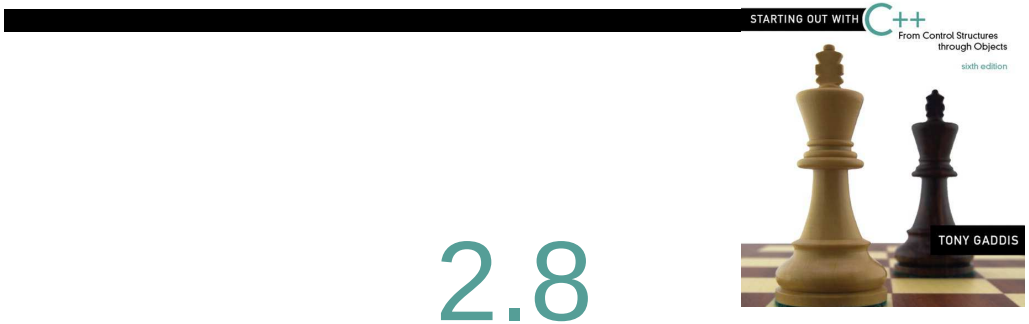
The `string` Data Type

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string stName;

    stName = "Steven Moore";
    cout << "Welcome " << stName << endl;

    return 0;
}
```

2.8

Floating-Point Data Types

Floating-Point Data Types

- Used to define variables that hold real numbers
- The floating-point data types are:

```
float
double
long double
```

- They can hold real numbers such as:

  `123.45` `-0.38`

- Stored in a form similar to scientific notation:

  1.2345×10^{2} -3.8×10^{-1}

- All floating-point numbers are signed

Floating-Point Data Types

Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	`float`	4 bytes. Numbers between ±3.4E-38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308
Long double precision	`long double*`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308

* Some compilers use 10 bytes for long double.

Floating-point Literals

- Can be represented in two ways
  - Fixed point (decimal) notation:

    `31.4159` `0.0000625`
  - E notation:

    `3.14159E1` `6.25e-5`
- Are `double` by default
- Can be forced to be `float` by appending F or f (`3.14159f`) or `long double` (`0.0000625L`)

Program 2-15

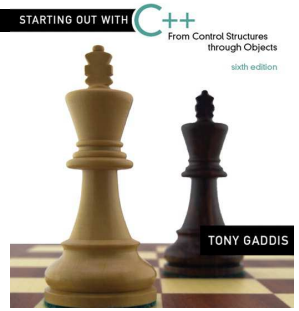
```
// This program uses floating point data types.
#include <iostream>
using namespace std;

int main()
{
   float distance;
   double mass;

   distance = 1.495979E11;
   mass = 1.989E30;
   cout << "The Sun is " << distance << " meters away.\n";
   cout << "The Sun\'s mass is " << mass << " kilograms.\n";
   return 0;
}
```

Program Output

```
The Sun is 1.49598e+011 meters away.
The Sun's mass is 1.989e+030 kilograms.
```



2.9

The bool Data Type

The bool Data Type

- Represents binary values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;
bool finished = false;
```

allDone	finished
1	0

Program 2-16

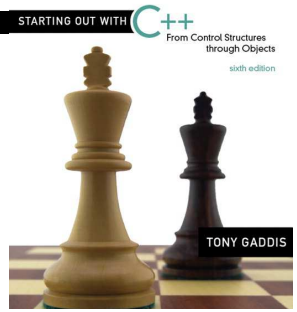
```
// This program demonstrates boolean variables.
#include <iostream>
using namespace std;

int main()
{
   bool boolValue;

   boolValue = true;
   cout << boolValue << endl;
   boolValue = false;
   cout << boolValue << endl;
   return 0;
}
```

Program Output

```
1
0
```

2.10

Determining the Size of a Data Type

Determining the Size of a Data Type

The size of some data type may be different from machine to machine

The `sizeof` operator gives the size of any ***data type*** or ***variable***:

```
double amount;
cout << "A double is stored in "
     << sizeof(double) << "bytes\n";
cout << "Variable amount is stored in "
     << sizeof(amount)
     << "bytes\n";
```


2.11

Variable Assignments and Initialization

Variable Assignments and Initialization

- An assignment statement uses the assignment = operator to store a value in a variable.

```
item = 12;
```

- This statement assigns the value 12 to the `item` variable.
- Can also assign other data types

```
ans = 'A';
```


Assignment

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

```
// ERROR!
12 = item;
```

Variable Initialization

- A variable just defined can have any value
- To initialize a variable means to assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```


Program 2-18

```
// This program shows variable initialization.
#include <iostream>
using namespace std;

int main()
{
   int month = 2, days = 28;

   cout << "Month " << month << " has " << days << " days.\n";
   return 0;
}
```

Program Output

```
Month 2 has 28 days.
```


Named Constants

- A named constant represents a value that cannot change while the program is running
- Declared with keyword `const` and capital letters
- Must be initialized when declared
  - For example:
    - `const double  PI = 3.1415926;`
    - `const int WIDTH;  // ERROR! Must`
    - `const int WIDTH = 10;  // initializ`

2.12

Scope

Scope

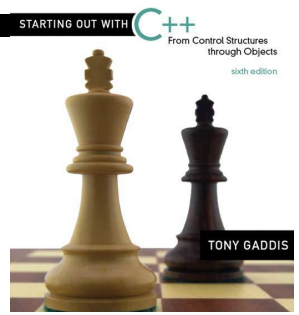
- The scope of a variable: the part of the program in which the variable can be used
- The rules are complex and will be introduced gradually
- A variable cannot be used before it is defined

Program 2-19

```
// This program can't find its variable.
#include <iostream>
using namespace std;

int main()
{
   cout << value; // ERROR! value not defined yet!

   int value = 100;
   return 0;
}
```

2.13

Arithmetic Operators

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators:
  - unary (1 operand) `-5`
  - binary (2 operands) `13 - 7`
  - ternary (3 operands) `exp1 ? exp2 : exp3`
    (Only one in C++, will be covered in Chapter 4)

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF ans
+	addition	`ans = 7 + 3;`	10
-	subtraction	`ans = 7 - 3;`	4
*	multiplication	`ans = 7 * 3;`	21
/	division	`ans = 7 / 3;`	2
%	modulus	`ans = 7 % 3;`	1

Assignment Statement

- **Format:**

```
Variable = Expression;
```

- **It evaluates the value of the expression and assign it to the variable**

```
double area, radius;
radius = 100.0;
area = 3.14 * radius * radius;
cout << "Area of the circle is "
     << area << endl;
```


A Closer Look at the / Operator

- When an expression is used with *cout*, its value will be evaluated and the result will be inserted into *cout*
- / (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;     // displays 2
cout << 91 / 7;     // displays 13
```


A Closer Look at the % Operator

- If either operand is floating point, the result is floating point

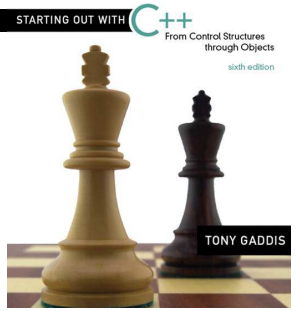
```
cout << 13 / 5.0;   // displays 2.6
cout << 91.0 / 7;   // displays 13.0
```

- % (modulus) operator computes the remainder resulting from integer division

```
cout << 13 % 5;    // displays 3
```

- % requires integers for both operands

```
cout << 13 % 5.0; // error
```



2.14

Comments

Comments

- Used to document parts of the program
- Intended for persons reading the source code of the program:
  - Indicate the purpose of the program
  - Describe the use of variables
  - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

Begin with `//` through to the end of line:

```
int length = 12;        // length in inches
int width = 15;         // width in inches
int area;               // calculated area

// calculate rectangle area
area = length * width;
```

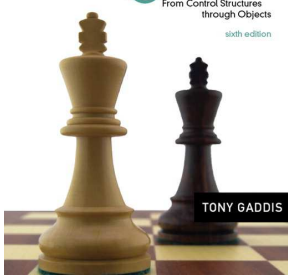

Multi-Line Comments

- Begin with `/*`, end with `*/`
- Can span multiple lines:

```
/* This is the first line of the comment
   ......
   This is the last line of the comment
*/
```

- Can begin and end on the same line:

```
int area;    /* calculated area */
```



2.15

Programming Style

Programming Style

- The visual organization of the source code
- Includes the use of spaces, tabs, and blank lines
- Does not affect the syntax of the program
- Affects the look and readability of the source code

Programming Style

Common elements to improve readability:

- Braces { } aligned vertically
- Indentation and alignment of statements within a set of braces

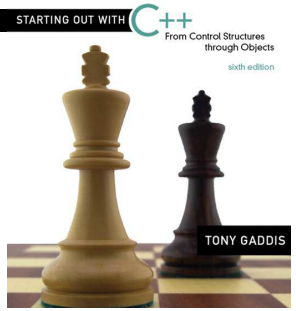
```
int main()
{
int number;        // not indented
number = 12;
cout << "Number = " << number << endl;
return 0;
    }              // not aligned
```


Programming Style

- Blank lines between variable declaration and other statements
- Long statements wrapped over multiple lines with aligned operators

```
int fahrenheit=98.7, celsius=37;

cout  << "The Fahrenheit temperature is "
      << fahrenheit
      << " and the Celsius temperature is "
      << celsius
      << endl;
```



2.17

A Programming Example

A Wage Problem

- The regular work hours per week is 40. Any hours worked over 40 are considered overtime. An employee earns $18.25 / hr. for regular hours, and $27.78 / hr. for overtime hours. The employee worked 50 hours this week. Write a program to calculate and display the employee's total wages for the week.

The Programming Process

1. Understand the problem
2. Develop an algorithm for solving the problem
3. Translate the algorithm into a computer language
4. Type the code, save it and compile it
5. Correct any syntax errors found during compilation. Repeat steps 4 and 5 as many times as needed
6. Run the program and check the result with test data
7. Correct any runtime and logical errors found while running the program. Repeat steps 4~7 as many times as needed

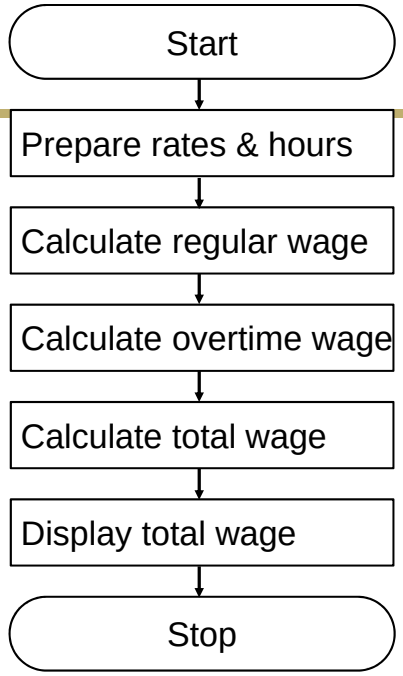
Algorithm

- An algorithm is a sequence of steps that shows how to solve a problem step by step.
- For example:
  - Driving direction
  - Cooking recipe
  - Directions for installing a device

Pseudocode Algorithm

- Pseudocode algorithm
  1. Prepare data: pay rates & work hours
  2. Regular wages = base pay rate × regular hours
  3. Overtime wages = overtime pay rate × overtime hours
  4. Total wages = regular wages + overtime wages
  5. Display the total wages
- The algorithm needs to be translated into C++
- In the beginning of program, define a variable for each value used in the algorithm (How many?)

Flowchart



Program 2-20

```
// This program calculates hourly wages, including overtime.
#include <iostream>
using namespace std;

int main()
{
   double regularWages,            // To hold regular wages
          basePayRate = 18.25,     // Base pay rate
          regularHours = 40.0,     // Hours worked less overtime
          overtimeWages,           // To hold overtime wages
          overtimePayRate = 27.78, // Overtime pay rate
          overtimeHours = 10,      // Overtime hours worked
          totalWages;              // To hold total wages

   // Calculate the regular wages.
   regularWages = basePayRate * regularHours;

   // Calculate the overtime wages.
   overtimeWages = overtimePayRate * overtimeHours;

   // Calculate the total wages.
   totalWages = regularWages + overtimeWages;

   // Display the total wages.
   cout << "Wages for this week are $" << totalWages << endl;
   return 0;
}
```

Program Output

```
Wages for this week are $1007.8
```