Welcome to Course

Programming Fundamentals

Week #3– Lecture 1 - 2

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 a multi-line
   comment
*/
```

- Can begin and end on the same line:

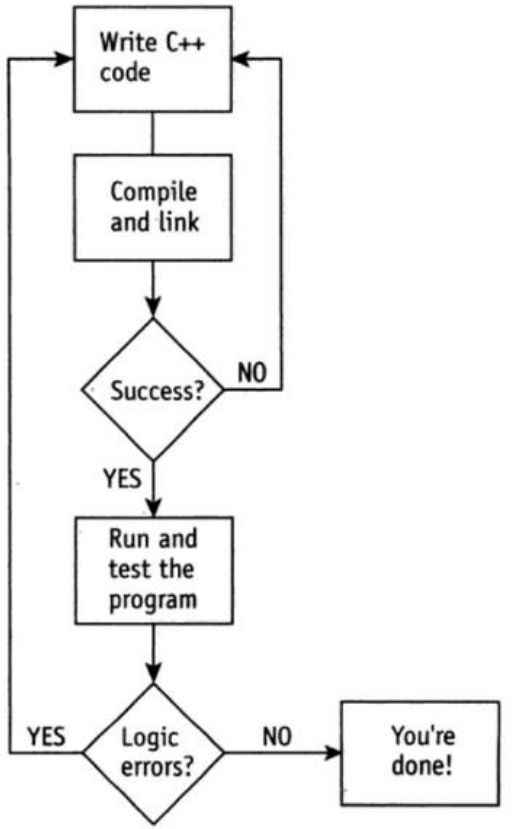
```
int area;   /* calculated area */
```

Bugs and Debugging

- Programming errors are called bugs
- The process of tracking bugs and correcting them is called debugging.
- Three kinds of errors can occur in a program:
  - Syntax errors
  - Logical errors
  - Runtime errors
- It is useful to distinguish between them in order to track them down more quickly

Program process

- Step1: Clearly define what the program is to do
- Step2: Visualize the program running on the computer
- Step3: Use design tools such as a hierarchy chart, flowcharts or pseudocode to create a model of the program
- Step4: Check the model for logical errors
- Step5: Type the code, save it, and compile it
- Step6: Correct any errors found during compilation and repeat steps 5 and 6 as many times as necessary
- Step7: Run the program with test data for input
- Step8: Correct any errors found while running the program and repeat steps 5 through 8 as many times as necessary
- Step9: Validate the results of the program

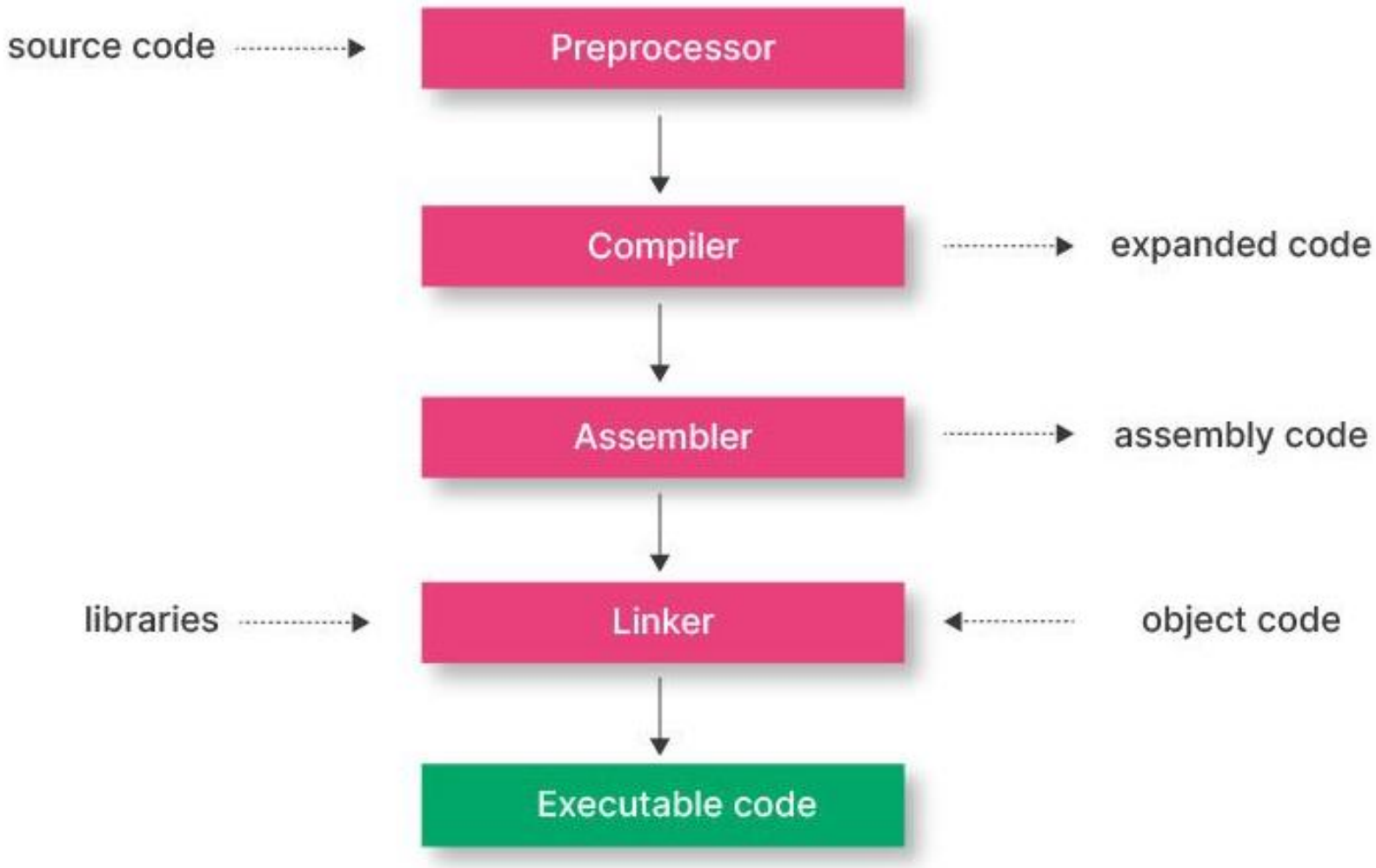


If the program run correctly, you are done. If it has program logic errors, you need to determine the source of error, make changes to the program, and rebuild it.

C++ code (first write the code and then run it)

- C++ is a general-purpose programming language
- Source code
  - Program files written with a text editor (e.g., program.cpp or .cc, cxx, .cp)
- Modified source code
  - Run pre-processor to convert source file directives to source code program statements (includes header files from standard or external libraries which contain declarations of features e.g., iostream or program.hpp files)
- Object code
  - Run compiler to convert source program into machine instructions
- Executable code
  - Run linker to connect hardware-specific code to machine instructions, producing an executable file

From a High-Level Program to an Executable File



From a High-Level Program to an Executable File

Step	Name	Description
1	**Preprocessing**	The preprocessor handles directives like `#include <iostream>`. It expands header files, removes comments, and performs macro substitution. The output is a **pure C++ source file** without preprocessor directives.
2	**Compilation**	The compiler translates the preprocessed code into **assembly language** for the target machine (CPU architecture). Errors like syntax or type errors are caught here.
3	**Assembly**	The assembler converts the assembly code into **machine code (object file, e.g., `.o` or `.obj`)** containing binary instructions.
4	**Linking**	The linker combines the object file with required libraries (like the C++ Standard Library that defines `cout`). It resolves external references and produces an **executable file** (e.g., `a.exe` on Windows or `a.out` on Linux).
5	**Execution**	Finally, the OS loads the executable into memory, and the CPU executes instructions. The program prints "H , **World!**" on the screen.

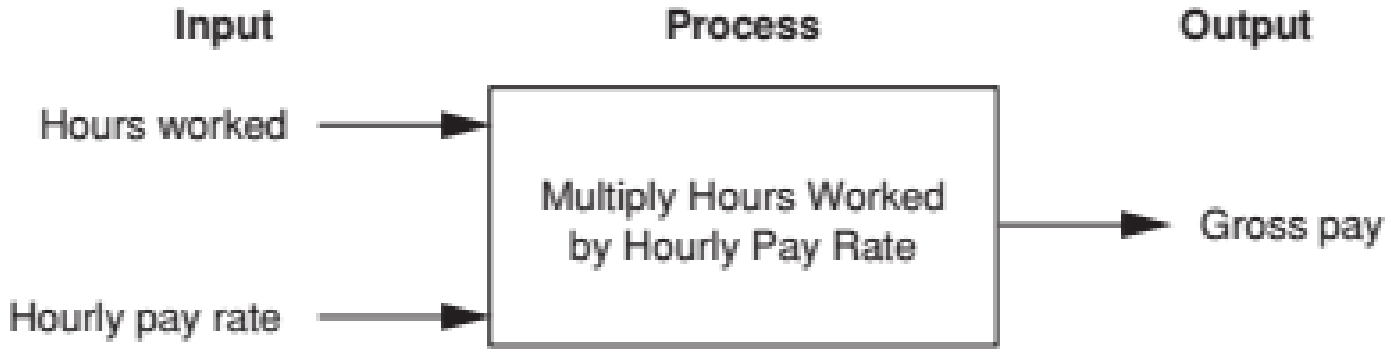
Integrated Development Environments (IDEs)

- An integrated development environment, or IDE, combine all the tools needed to write, compile, and debug a program into a single software application.
- Examples are Microsoft Visual C++, Dev C++, Turbo C++ Explorer, CodeWarrior, etc.

Basic Program Structure

- Any computer program usually has the following structure:
  1. Receive input
  2. Process input
  3. Produce output

Figure 2-3 The input, processing, and output of the pay calculating program



Control Structure of a Program

- In the 1960s, a group of mathematicians proved that only three program structures are needed to write any type of program.
- **Sequence Structure:**
  - The simplest structure of a program is to run instructions one after the other in the sequence that they are written
- **Decision structures**
  - A program may (or may not) run a few instructions in the sequence based on some condition
- **Repetition structures**
  - A program may repeat a set of instructions a number of times

Variables, String, Input, and Prompt

- A ***variable*** is a named memory location
- A ***string*** is a sequence of characters and is enclosed inside double quotes in a program
- An ***input*** command is used to read data from keyboard into a variable
- A ***prompt*** is a *display* command to tell the user what input is required

```
1 Display "What is your age?"
2 Input age
3 Display "Here is the value that you entered:"
4 Display age
```

Program Output (with Input Shown in Bold)

```
What is your age?
24 [Enter]
Here is the value that you entered:
24
```

Variable Assignment

- You can store a value in a variable with an assignment statement

```
1 Set dollars = 2.75
2 Display "I have ", dollars, " in my account."
```

Program Output

```
I have 2.75 in my account.
```

```
1 Set dollars = 2.75
2 Display "I have ", dollars, " in my account."
3 Set dollars = 99.95
4 Display "But now I have ", dollars, " in my account!"
```

Program Output

```
I have 2.75 in my account.
But now I have 99.95 in my account!
```

Variable Assignment

- You can store a value in a variable with an assignment statement
- The value can be the result of a calculation, which is created with math operators

Table 2-1 Common math operators

Symbol	Operator	Description
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the quotient
MOD	Modulus	Divides one number by another and gives the remainder
^	Exponent	Raises a number to a power

Calculations on a Variable

- You can store a value in a variable with an assignment statement
- The value can be the result of a calculation, which is created with math operators

```
1 Set price = 100
2 Set discount = 20
3 Set sale = price - discount
4 Display "The total cost is $", sale
```

Program Output

```
The total cost is $80
```

Variables and Data Types

- All variables are declared before they are used in a program.
- The declared variables need to know the type of the memory that they point to, in order to allow any operation on the data.
  - Integer, Real, String, Character, etc.

Program 2-13

```
1  Declare Real test1
2  Declare Real test2
3  Declare Real test3
4  Declare Real average
5
6  Set test1 = 88.0
7  Set test2 = 92.5
8  Set test3 = 97.0
9  Set average = (test1 + test2 + test3) / 3
10 Display "Your average test score is ", average
```

Program Output

```
Your average test score is 92.5
```

Hand Tracing a Program

- A simple debugging technique that helps in understanding the logic of a program.

Figure 2-15 Program with the hand trace chart completed

```
1  Declare Real test1
2  Declare Real test2
3  Declare Real test3
4  Declare Real average
5
6  Set test1 = 88.0
7  Set test2 = 92.5
8  Set average = (test1 + test2 + test3) / 3
9  Display "Your average test score is ", average
```

	test1	test2	test3	average
1	?	?	?	?
2	?	?	?	?
3	?	?	?	?
4	?	?	?	?
5	?	?	?	?
6	88	?	?	?
7	88	92.5	?	?
8	88	92.5	?	undefined
9	88	92.5	?	undefined

What is a Program Made of?

- Keywords or reserved words: Words that have a special meaning
- Programmer-defined identifiers: Words or names defined by the programmer (refer to variables or programming routines)
- Operators: Operators perform operations on one or more operands
- Punctuation: Punctuation characters that mark the beginning or ending of a statement, or separate items in a list
- Syntax: Rules that must be followed when constructing a program (how keywords and operators may be used, and where punctuation symbols must appear)
- Line: A single line appears in the body of a program
- Statement: an instruction that cause computer to perform some action

What Is a program made of?

```
1. // This program calculates the user's pay.
2. #include <iostream>
3. using namespace std;
4.
5. int main()
6.  {
7.    double hours, rate, pay;
8.
9.    // Get the number of hours worked.
10.    cout << "How many hours did you work? ";
11.    cin >> hours;
12.
13.    // Get the hourly pay rate.
14.    cout << "How much do you get paid per hour? ";
15.    cin >> rate;
16.
17.    // Calculate the pay.
18.    pay = hours * rate;
19.
20.    // Display the pay.
21.    cout << "You have earned $" << pay << endl;
22.    return 0;
23. }
```

Identify keywords?
lines (3 and 5)
using, namespace, int
Identify punctuations?
lines (3, 7, 10, 11, 14, 15, 18, 21, 22)
; ,
Identify operators?
lines (18)
*, =
Identify programmer defined identifiers?
lines (7),
hour, rate, pay

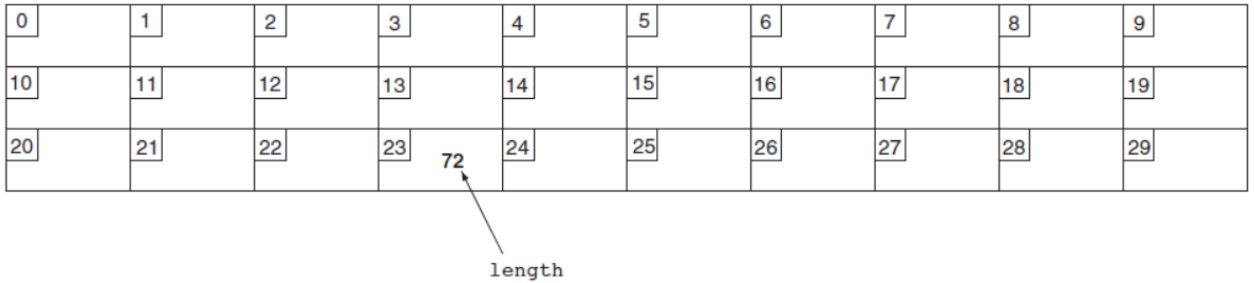
Any syntax?

Line ? Lines from 1-23

Statement? Line 14, 15 etc.,

Variables and identifier

- A variable is a named storage location in the computer's memory for holding a piece of information
  - Or a quantity whose value may change during execution of the program is called variable
  - It represented by symbol or a name (e.g., length = 72)
  - Data is stored into the memory location on a specific address
  - Variable name remains fixed during the execution of the program, but the data stored in that location may change from time to time
  - A variable name consists of alphabets and digits
  - An identifier is a programmer-defined name for some part of a program: variables, functions, etc.



Rules for naming a variable or identifiers

- First character of variable name must be an alphabetic character or underscore e.g., Length or _length
- After first character you may use the digits from 0 to 9, e.g., value1
- Blank spaces are not allowed in a variable name e.g., first name should be first_name or firstName etc.,
- Special characters (arithmetic operators, #, ^) cannot be used in a variable name
- Reserved words cannot be used as a variable names
- A variable name declared for one data-type cannot be used to declare another data-type
- C++ is a case sensitive language, so a length and LENGTH are treated as different
- A variable name should represent the purpose of the variable. For example: **`itemsOrdered`** The purpose of this variable is to hold the number of items ordered

C++ Keywords

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

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	

Legal and illegal variable names

Variable Name	Legal or Illegal?
`dayOfWeek`	
`3dGraph`	
`_employee_num`	
`June1997`	
`Mixture#3`	

Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`		
`total_Sales`		
`total.Sales`		
`4thQtrSales`		
`totalSale$`		

Constants

- Example: statement in a bank program: double amount = balance * 0.069;
- Is there any potential problems?
- Problems:
  1. It is not clear to anyone other than the original programmer that what is 0.069?
  2. A problem occurs if this number is used in other calculations throughout the program and must be changed periodically
- Both problems can be addressed by using named constants
  - It is like a variable, but its content is read-only and cannot be changed while the program is running e.g.,

    const double INTEREST_RATE = 0.069;
- can be changed to read the statement in bank program as

  double amount = balance * INTEREST_RATE;

const keyword and its advantages

- const is a qualifier that tells the compiler to make the variable read-only
  - Its value will remain constant throughout the program's execution
  - An initialization value must be given when defining a constant with the const qualifier
  - A compiler error will also result if there are any statements in the program that attempt to change the value of a named constant
- Advantages
  - make programs more self documenting
  - widespread changes can easily be made to the program
  - can also help prevent typographical errors in a program's code e.g., writing of the value of pi= 3.14159 in program repeatedly

Example program

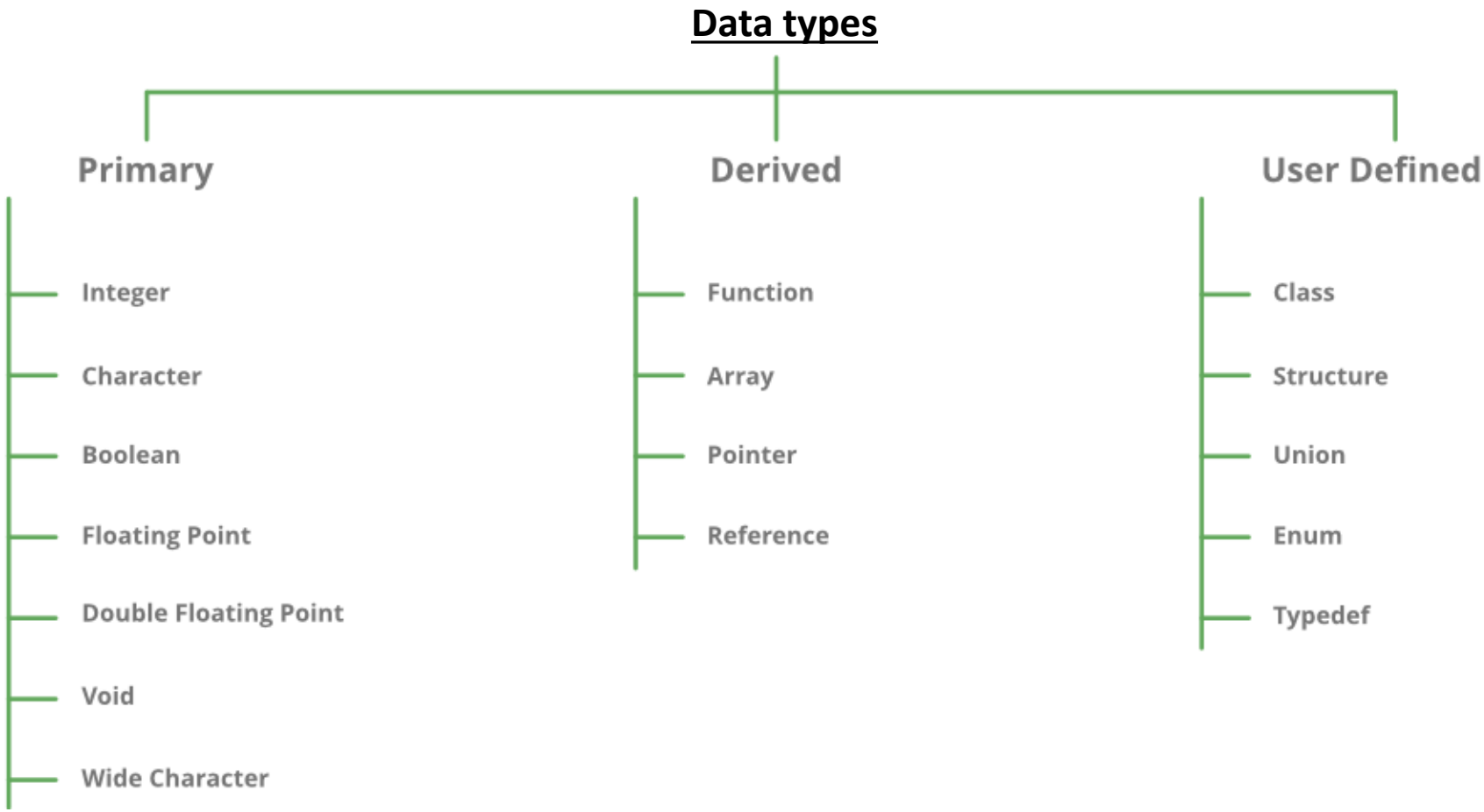
```
// This program calculates the circumference of a circle.
#include <iostream>
using namespace std;

int main()
{
    // Constants
    const double PI = 3.14159;
    const double DIAMETER = 10.0;

    // Variable to hold the circumference

    double circumference;

    // Calculate the circumference.
    circumference = PI * DIAMETER;

    // Display the circumference.
    cout << "The circumference is: " << circumference << endl;
    return 0;
}
```

Output:

The circumference is: 31.4159

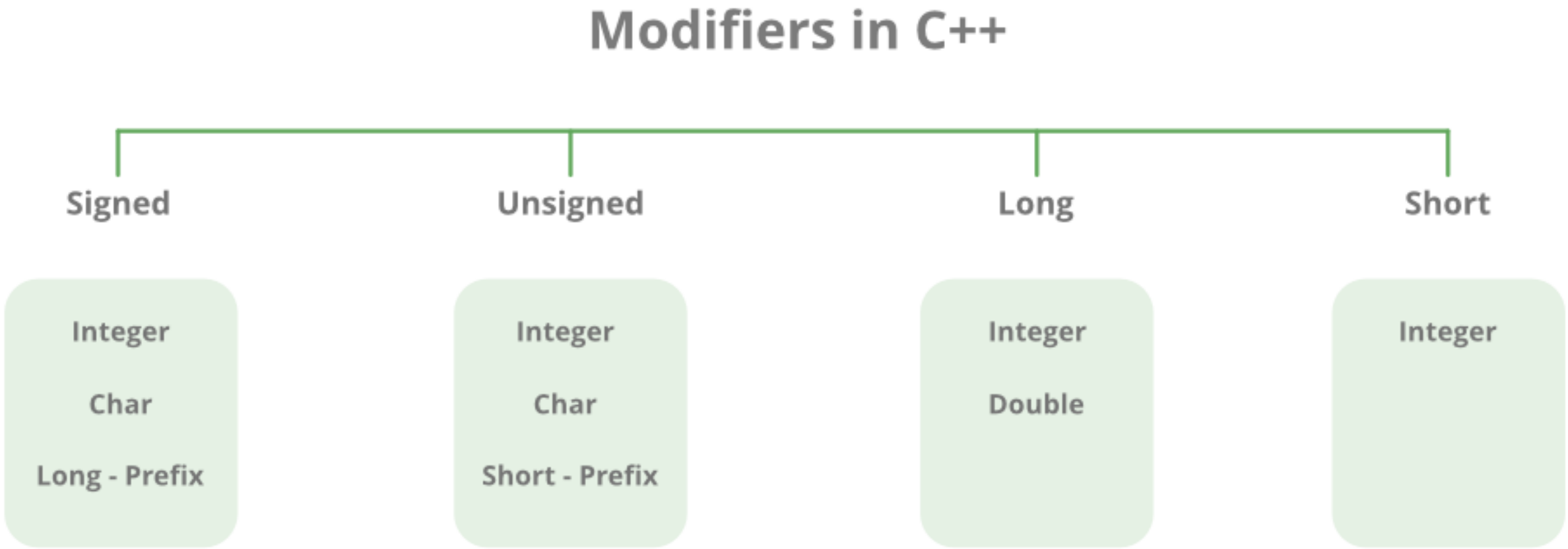
Data Types

Data types in C/C++



https://www.geeksforgeeks.org/c-data-types/

Datatype Modifiers



https://www.geeksforgeeks.org/c-data-types/

Data types and size (in bytes) ranges

Data Type	Size (in bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	8	-2,147,483,648 to 2,147,483,647
unsigned long int	8	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

Note: Above values may vary from compiler to compiler. In the above example, we have considered GCC 32 bit

2.6

Integer Data Types

Integer Data Types

- Integer variables can hold whole numbers such as 12, 7, and -99.

Table 2-6 Integer Data Types, Sizes, and Ranges

Data Type	Size	Range
`short`	2 bytes	–32,768 to +32,767
`unsigned short`	2 bytes	0 to +65,535
`int`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned int`	4 bytes	0 to 4,294,967,295
`long`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned long`	4 bytes	0 to 4,294,967,295

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

Integer Types in Program 2-10

Program 2-10

```
1  // This program has variables of several of the integer types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int checking;
8     unsigned int miles;
9     long days;
10
11    checking = -20;
12    miles = 4276;
13    days = 189000;
14    cout << "We have made a long journey of " << miles;
15    cout << " miles.\n";
16    cout << "Our checking account balance is " << checking;
17    cout << "\nAbout " << days << " days ago Columbus ";
18    cout << "stood on this spot.\n";
19    return 0;
20 }
```

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

Integer Literals

- An integer literal is an integer value that is typed into a program's code. For example:

```
itemsOrdered = 15;
```

  In this code, 15 is an integer literal.

Integer Literals in Program 2-10

Program 2-10

```
1  // This program has variables of several of the integer types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     int checking;
8     unsigned int miles;
9     long days;
10
11    checking = -20;
12    miles = 4276;
13    days = 189000;
14    cout << "We have made a long journey of " << miles;
15    cout << " miles.\n";
16    cout << "Our checking account balance is " << checking;
17    cout << "\nAbout " << days << " days ago Columbus ";
18    cout << "stood on this spot.\n";
19    return 0;
20 }
```

Integer Literals

- Integer literals are stored in memory as `int`s by default
- To store an integer constant in a long memory location, put '`L`' at the end of the number: `1234L`
- 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

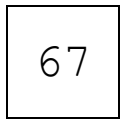
- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

CODE:

```
char letter;
letter = 'C';
```

MEMORY:

letter



Character Literals

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

```
'A'
```

Character Literals in Program 2-13

Program 2-13

```
1  // This program uses character literals.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     char letter;
8
9     letter = 'A';
10    cout << letter << endl;
11    letter = 'B';
12    cout << letter << endl;
13    return 0;
14 }
```

Program Output

```
A
B
```

Character Strings

- A series of characters in consecutive memory locations:

  `"Hello"`

- Stored with the null terminator, `\0`, at the end:

- Comprised of the characters between the `"  "`