11.1 Binary

This section has been set as optional by your instructor.

Normally, a programmer can think in terms of base ten numbers. However, a compiler must allocate some finite quantity of bits (e.g., 32 bits) for a variable, and that quantity of bits limits the range of numbers that the variable can represent. Thus, some background on how the quantity of bits influences a variable's number range is helpful.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Because each memory location is composed of bits (0s and 1s), a processor stores a number using base 2, known as a ***binary number***.

For a number in the more familiar base 10, known as a ***decimal number***, each digit must be 0-9 and each digit's place is weighed by increasing powers of 10.

Table 11.1.1: Decimal numbers use weighed powers of 10.

Decimal number with 3 digits	Representation		
212	$= 2 \cdot 10^2$ $= 2 \cdot 100$ $= 200$ $= 212$	$+ 1 \cdot 10^1$ $+ 1 \cdot 10$ $+ 10$	$+ 2 \cdot 10^0$ $+ 2 \cdot 1$ $+ 2$

In ***base 2***, each digit must be 0-1 and each digit's place is weighed by increasing powers of 2.

Table 11.1.2: Binary numbers use weighed powers of 2.

Binary number with 4 bits	Representation			
1101	$= 1 \cdot 2^3$ $= 1 \cdot 8$ $= 8$ $= 13$	$+ 1 \cdot 2^2$ $+ 1 \cdot 4$ $+ 4$	$+ 0 \cdot 2^1$ $+ 0 \cdot 2$ $+ 0$	$+ 1 \cdot 2^0$ $+ 1 \cdot 1$ $+ 1$

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The compiler translates decimal numbers into binary numbers before storing the number into a memory location. The compiler would convert the decimal number 212 to the binary number 11010100, meaning 1*128 + 1*64 + 0*32 + 1*16 + 0*8 + 1*4 + 0*2 + 0*1 = 212, and then store that binary number in memory.

PARTICIPATION ACTIVITY 11.1.1: Understanding binary numbers.

Set each binary digit for the unsigned binary number below to 1 or 0 to obtain the binary equivalents of 9, then 50, then 212, then 255. Note also that 255 is the largest integer that the 8 bits can represent.

0	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

0

(decimal value)

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.1.2: Binary numbers.

1) Convert the binary number 00001111 to a decimal number.

Check Show answer

2) Convert the binary number 10001000 to a decimal number.

Check Show answer

3) Convert the decimal number 17 to an 8-bit binary number.

Check Show answer

4) Convert the decimal number 51 to an 8-bit binary number.

Check Show answer

11.2 Unsigned

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

This section has been set as optional by your instructor.

Sometimes a programmer knows that a variable's numbers will always be positive (0 or greater), such as when the variable stores a person's age or weight. The programmer can prepend the word "unsigned" to inform the compiler that the integers will always be positive. Because the integer's sign need not be stored, the integer range reaches slightly higher numbers, as follows:

Table 11.2.1: Unsigned integer data types.

Declaration	Size	Supported number range	Standard-defined minimum size
unsigned char myVar;	8 bits	0 to 255	8 bits
unsigned short myVar;	16 bits	0 to 65,535	16 bits
unsigned long myVar;	32 bits	0 to 4,294,967,295	32 bits
unsigned long long myVar;	64 bits	0 to 18,446,744,073,709,551,615	64 bits
unsigned int myVar;	32 bits	0 to 4,294,967,295	*16 bits*

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Signed numbers use the leftmost bit to store a number's sign, and thus the largest magnitude of a positive or negative integer is half the magnitude for an unsigned integer. Signed numbers actually use a more complicated representation called two's complement, but that's beyond our scope.

The following example demonstrates the use of unsigned long and unsigned long long variables to convert memory size.

Figure 11.2.1: Unsigned variables example: Memory size converter.

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

int main() {
   unsigned long memSizeGiB;
   unsigned long long memSizeBytes;
   unsigned long long memSizeBits;

   cout << "Enter memory size in GiBs: ";
   cin >> memSizeGiB;

   // 1 GiB = 1024 MiB, 1 MiB = 1024 KiB, 1 KiB =
1024 bytes
   memSizeBytes = memSizeGiB * (1024 * 1024 *
1024);
   // 1 byte = 8 bits
   memSizeBits = memSizeBytes * 8;

   cout << "Memory size in bytes: " << memSizeBytes
<< endl;
   cout << "Memory size in bits: " << memSizeBits
<< endl;

   return 0;
}
```

```
Enter memory size in GiBs: 1
Memory size in bytes:
1073741824
Memory size in bits:
8589934592

...

Enter memory size in GiBs: 4
Memory size in bytes:
4294967296
Memory size in bits:
34359738368
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.2.1: Unsigned variables.

1) Declare a 64-bit unsigned integer variable numMolecules.

Check Show answer

2) Declare a 16-bit unsigned integer variable named numAtoms.

Check Show answer

3) Initialize numAtoms to the smallest valid unsigned value.

```
unsigned short numAtoms =
;
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.3 Random numbers

This section has been set as optional by your instructor.

Generating a random number

Credit: Microsoft ClipArt gallery

Some programs need to use a random number. Ex: A game program may need to roll dice, or a website program may generate a random initial password.

The ***rand()*** function, in the C standard library, returns a random integer each time the function is called, in the range 0 to RAND_MAX.

Figure 11.3.1: Outputting three random integers.

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

int main() {
   cout << rand() << endl;
   cout << rand() << endl;
   cout << rand() << endl;

   cout << "(RAND_MAX: " << RAND_MAX << ")" <<
endl;

   return 0;
}
```

```
16807
282475249
1622650073
(RAND_MAX:
2147483647)
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

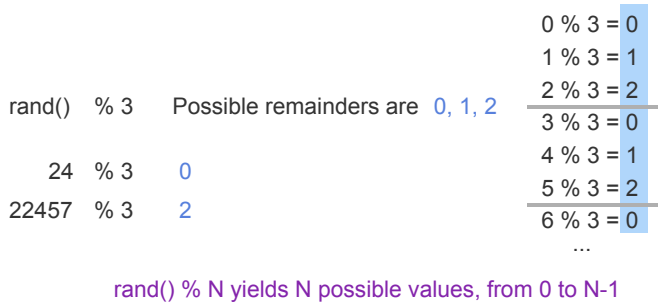
Line 2 includes the C standard library, which defines the rand() function and RAND_MAX.

RAND_MAX is a machine-dependent value, but is at least 32,767. Above, RAND_MAX is about 2 billion.

Usually, a programmer wants a random integer restricted to a specific number of possible values. The modulo operator % can be used. Ex: `integer % 10` has 10 possible remainders: 0, 1, 2, ..., 8, 9.

PARTICIPATION ACTIVITY | 11.3.1: Restricting random integers to a specific number of possible values.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024



Animation content:

Static figure:
Text on the bottom of the animation frame: "rand() % N yields N possible values, from 0 to N-1

Left of the frame are three rows of code, each of which has commentary to the side. The top down: "rand() % 3 Possible remainders are 0,1,2"; "24 % 3 0"; "22457 % 3 2".

To the right of the animation frame is a list that shows incrementing values mod 3. From the top down: "0 % 3 = 0"; "1 % 3 = 1"; "2 % 3 = 0"; "3 % 3 = 0"; "4 % 3 = 1"; "5 % 3 = 2"; "6 % 3 = 0"; "...". The answers to these modulus equations are all highlighted in a single column, showing the repeating pattern 0,1,2; 0,1,2; 0....

Step 1: Each call to rand() returns a random integer between 0 and a large number RAND_MAX. The animation frame is blank except for the text "rand()". Under "rand()" appears the number "24"; under "24" appears "22457".

Step 2: A programmer usually wants a smaller number of possible values, for which % can be used. % (modulo) means remainder. rand() % 3 has possible remainders of 0, 1, and 2. To the right of "rand()" appears the text "% 3". Next to this appears the text "Possible remainders are". To the right of the frame appears the list, wherein each row shows a different number modulo 3: "0 % 3 = 0"; "1 % 3 = 1"; "2 % 3 = 0"; "3 % 3 = 0"; "4 % 3 = 1"; "5 % 3 = 2"; "6 % 3 = 0"; "...". Next to "rand() % 3 Possible remainders are" appears the text "0, 1, 2".

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Step 3: Thus, rand() % 3 yields 3 possible values: 0, 1, and 2. Generally, rand() % N yields N possible values, from 0 to N-1. Next to the text "24" appears the text "% 3 0"; next to the text "22457" appears the text " % 3 2".

Animation captions:

1. Each call to rand() returns a random integer between 0 and a large number RAND_MAX.
2. A programmer usually wants a smaller number of possible values, for which % can be used. % (modulo) means remainder. rand() % 3 has possible remainders of 0, 1, and 2.

3. Thus, rand() % 3 yields 3 possible values: 0, 1, and 2. Generally, rand() % N yields N possible values, from 0 to N-1.

PARTICIPATION ACTIVITY 11.3.2: Random number basics.

1) What library must be included to use the rand() function?

- The C random numbers library
- The C standard library

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) The random integer returned by rand() will be in what range?

- 0 to 9
- -RAND_MAX to RAND_MAX
- 0 to RAND_MAX

3) Which expression's range is restricted from 0 to 7?

- rand() % 7
- rand() % 8

4) Which expression yields one of 5 possible values?

- rand() % 4
- rand() % 5
- rand() % 6

5) Which expression yields one of 100 possible values?

- rand() % 99
- rand() % 100
- rand() % 101

6) Which expression would best mimic the random outcome of flipping a coin?

- rand() % 1
- rand() % 2
- rand() % 3

7) What is the smallest *possible* value returned by rand() % 10?

- 0
- 1
- 10
- Unknown

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

8) What is the largest *possible* value returned by rand() % 10?

- 10
- 9
- 11

Specific ranges

The technique above generates random integers with N possible values ranging from 0 to N-1, like 6 values from 0 to 5. Commonly, a programmer wants a specific range that starts with some value x that isn't 0, like 10 to 15, or -20 to 20. The programmer should first determine the number of values in the range, generate a random integer with that number of possible values, and then add x to adjust the range to start with x.

zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY | 11.3.3: Generating random integers in a specific range not starting from 0.

Animation content:

Static figure:
There is a row of numbers: "10 11 12 13 14 15". Under this is the expression "15 - 10 + 1", to the side of which is the text "6 possible values". Below all this is an expression "(rand() % 6) + 10", under which is a row of numbers, "10 11 12 13 14 15". To the left is the expression "rand() % 6", under which is a row of numbers "0 1 2 3 4 5".

Step 1: A programmer wants random integers in the range 10 to 15. The number of possible values is 15 – 10 + 1. (People often forget the + 1.) The animation frame is blank, save a row of numbers: "10 11 12 13 14 15". Under this row appears the expression "15 - 10 + 1", to the side of which appears the text "6 possible values".

Step 2:The expression "rand() % 6" generates 6 possible values as desired, but with range 0 to 5. The expression "rand() % 6" appears, under which appears a row of numbers "0 1 2 3 4 5".

Step 3: Adding 10 to "rand() % 6" still generates 6 values, but now those values start at 10. The range thus becomes 10 to 15. On top of "rand % 6" appears the expression "(rand() % 6) + 10". This appeared expression moves to an unoccupied location of the animation frame. On top of "0 1 2 3 4 5" appears the row of numbers "10 11 12 13 14 15". This row of numbers moves to rest under "(rand() % 6) + 10".

zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Animation captions:

1. A programmer wants random integers in the range 10 to 15. The number of possible values is 15 – 10 + 1. (People often forget the + 1.)
2. rand() % 6 generates 6 possible values as desired, but with range 0 to 5.
3. Adding 10 still generates 6 values, but now those values start at 10. The range thus becomes 10 to 15.

PARTICIPATION ACTIVITY 11.3.4: Generating random integers in a specific range.

1) Goal: Random integer from the 6 possible values 0 to 5.
rand() % ___

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) Goal: Random integer from 0 to 4.
rand() % ___

Check Show answer

3) How many values exist in the range 10 to 15?

Check Show answer

4) How many values exist in the range 10 to 100?

Check Show answer

5) Goal: Random integer in the range 10 to 15.
(rand() % 6) + ___

Check Show answer

6) Goal: Random integer in the range 16 to 25.
(rand() % ___) + 16

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

7) How many values are in the range -5 to 5?

Check Show answer

8) Goal: Random integer in the range -20 to 20.
(rand() % 41) + ___

Check Show answer

PARTICIPATION ACTIVITY 11.3.5: Specific range.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1) Which generates a random integer in the range 18 ... 30?

- rand() % 30
- rand() % 31
- rand() % (30 - 18)
- (rand() % (30 - 18)) + 18
- (rand() % (30 - 18 + 1)) + 18

The following program randomly moves a student from one seat to another seat in a lecture hall, perhaps to randomly move students before an exam. The seats are in 20 rows numbered 1 to 20. Each row has 30 seats (columns) numbered 1 to 30. The student should be moved from the left side (columns 1 to 15) to the right side (columns 16 to 30).

Figure 11.3.2: Randomly moving a student from one seat to another.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

// Switch a student
// from a random seat on the left  (cols  1 to
15)
//   to a random seat on the right (cols 16 to
30)
// Seat rows are 1 to 20

int main() {
   int rowNumL;
   int colNumL;
   int rowNumR;
   int colNumR;

   rowNumL = (rand() % 20) + 1;  // 1 to 20
   colNumL = (rand() % 15) + 1;  // 1 to 15

   rowNumR = (rand() % 20) + 1;  // 1 to 20
   colNumR = (rand() % 15) + 16; // 16 to 30

   cout << "Move from ";
   cout << "row " << rowNumL << " col " <<
colNumL;
   cout << " to " ;
   cout << "row " << rowNumR << " col " <<
colNumR;
   cout << endl;

   return 0;
}
```

```
Move from row 8 col 5 to row 14
col 24
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.3.6: Random integer example: Moving seats.

Consider the above example.

1) The row is chosen using (rand() % 20) + 1. The 20 is because 20 rows exist. The + 1 is _____ .

- necessary
- optional

2) The column for the left is chosen using (rand() % 15) + 1. The 15 is used because the left half of the hall has _____ columns.

- 15
- 30

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) The column for the right could have been chosen using (rand() % 15) + 15.

- True
- False

Pseudo-random

The integers generated by rand() are known as pseudo-random. "Pseudo" means "not actually, but having the appearance of". The integers are pseudo-random because each time a program runs, calls to rand() yield the same sequence of values. Earlier in this section, a program called rand() three times and output 16807, 282475249, 1622650073. Every time the program is run, those same three integers will be printed. Such reproducibility is important for testing some programs. (Players of classic arcade games like Pac-man may notice that the seemingly-random actions of objects actually follow the same pattern every time the game is played, allowing players to master the game by repeating the same winning actions).

Internally, the rand() function has an equation to compute the next "random" integer from the previous one, (invisibly) keeping track of the previous one. For the first call to rand(), no previous random integer exists, so the function uses a built-in integer known as the ***seed***. By default, the seed is 1. A programmer can change the seed using the function srand(), as in srand(2) or srand(99).

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

If the seed is different for each program run, the program will get a unique sequence. One way to get a different seed for each program run is to use the current time as the seed. The function ***time()*** returns the number of seconds since Jan 1, 1970.

Note that the seeding should only be done once in a program, before the first call to rand().

Figure 11.3.3: Using a unique seed for each program run.

```
#include <iostream>
#include <cstdlib>
#include <ctime>    // Enables use of time()
function
using namespace std;

int main() {
   srand(time(0));  // Unique seed
   cout << rand() << endl;
   cout << rand() << endl;
   cout << rand() << endl;

   return 0;
}
```

```
636952311
51510682
304122633

(next run)

637053153
1746362176
1450088483
```

PARTICIPATION ACTIVITY 11.3.7: Using a unique seed for each program run.

1) The s in srand() most likely stands for _____ .

- sequence
- seed

2) By starting a program with srand(15), calls to rand() will yield a different integer sequence for each program run.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) By starting a program with srand(time(0)), calls to rand() will yield a different integer sequence for each successive program run.

- True
- False

4) rand() is known as generating a "pseudo-random" sequence of values because the sequence begins repeating itself after about 20 numbers.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Exploring further:

- C++ random number library

CHALLENGE ACTIVITY 11.3.1: Generate a random integer.

602398.4584902.qx3zqy7

Start

Generate a random integer between 0 and 4 (inclusive)

rand() % Ex: 20

1	2	3	4

Check Next

CHALLENGE ACTIVITY 11.3.2: Random numbers.

602398.4584902.qx3zqy7

Start

Integer seedVal is read from input. srand() is called with seedVal as the seed. Use rand() to assign variables dataPoint1, dataPoint2, dataPoint3, and dataPoint4 each with a random number generated between 0 and 26, both inclusive.

▶ **Click here for example**

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

int main() {
   int seedVal;
   int dataPoint1;
   int dataPoint2;
   int dataPoint3;
   int dataPoint4;
   int sum;

   cin >> seedVal;
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
srand(seedVal);
```

1 2 3

Check Next level

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.4 Character operations

This section has been set as optional by your instructor.

Including the ***cctype library*** via `#include <cctype>` provides access to several functions for working with characters. ctype stands for character type. The first c indicates the library is originally from the C language.

Table 11.4.1: Character functions return values.

isalpha(c)	true if alphabetic: a-z or A-Z	`isalpha('x')`
		`// true`
		`isalpha('6')`
		`// false`
		`isalpha('!')`
		`// false`
				toupper(c)	Uppercase version	`letter = toupper('a')`
						`// A`
						`letter = toupper('A')`
						`// A`
						`letter = toupper('3')`
						`// 3`

isdigit(c)	true if digit: 0-9.	`isdigit('x')`
		`// false`
		`isdigit('6')`
		`// true`
				tolower(c)	Lowercase version	`letter = tolower('A')`
						`// a`
						`letter = tolower('a')`
						`// a`
						`letter = tolower('3')`
						`// 3`

isspace(c)	true if whitespace.	`isspace(' ')`
		`// true`
		`isspace('\n')`
		`// true`
		`isspace('x')`
		`// false`				

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Note: Above, false is zero, and true is non-zero.

See http://www.cplusplus.com/reference/cctype/ for a more complete list (applies to both C and C++).

Figure 11.4.1: State abbreviation capitalization.

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

int main() {
   char let0;
   char let1;

   cout << "Enter a two-letter state
abbreviation: ";
   cin >> let0;
   cin >> let1;

   if ( ! (isalpha(let0) && isalpha(let1)) )
{
      cout << "Error: Both are not letters."
<< endl;
   }
   else {
      let0 = toupper(let0);
      let1 = toupper(let1);
      cout << "Capitalized: " << let0 << let1
<< endl;
   }

   return 0;
}
```

```
Enter a two-letter state
abbreviation: az
Capitalized: AZ
...
Enter a two-letter state
abbreviation: AZ
Capitalized: AZ
...
Enter a two-letter state
abbreviation: Mn
Capitalized: MN
...
Enter a two-letter state
abbreviation: 5x
Error: Both are not letters.
...
Enter a two-letter state
abbreviation: A@
Error: Both are not letters.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.4.1: Character functions.

To what value does each evaluate? userStr is "Hey #1?".

1) isalpha('7')

- True
- False

2) isalpha(userStr.at(0))

- True
- False

3) isspace(userStr.at(3))

- True
- False

4) isdigit(userStr.at(6))

- True
- False

5) toupper(userStr.at(1)) returns 'E'.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

6) tolower(userStr.at(2)) yields an error because 'y' is already lower case.

- True
- False

7) tolower(userStr.at(6)) yields an error because '?' is not alphabetic.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

8) After tolower(userStr.at(0)), userStr becomes "hey #1?"

- True
- False

CHALLENGE ACTIVITY 11.4.1: Character operations.

602398.4584902.qx3zqy7

Start

Variable userString is assigned with a 2-character string read from input. If userString's first character is alphabetic, output "The first character is alphabetic: " followed by the alphabetic character. Otherwise, output "The first character is not alphabe End each output with a newline.

Ex: If the input is `c@`, then the output is:

```
The first character is alphabetic: c
```

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

int main() {
   string userString;

   getline(cin, userString);

   /* Your code goes here */

   return 0;
}
```

1 2 3

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Check Next level

11.5 More string operations

This section has been set as optional by your instructor.

Finding in a string / Getting a substring

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The string library provides numerous useful functions, including functions for finding a character or string within a string, or getting a substring of a string.

Table 11.5.1: find() and substr() functions, invoked as myString.find().

find()	***find***(item) returns index of first item occurrence, else returns string::npos (a constant defined in the string library). Item may be char, string variable, string literal (or char array).
	
	find(item, indx) starts at index indx.
		`// userText is "Help me!"`
		`userText.find('p')  // Returns 3`
		`userText.find('e')  // Returns 1 (first occurrence of e only)`
		`userText.find('z')  // Returns string::npos`
		`userText.find("me")  // Returns 5`
		`userText.find('e', 2) // Returns 6 (starts at index 2)`

substr()	***substr***(index, length) returns substring starting at index and having length characters.	`// userText is "http://google.com"`
		`userText.substr(0, 7)  // Returns "http://"`
		`userText.substr(13, 4)  // Returns ".com"`
		`userText.substr(userText.size() - 4, 4)  // Last 4: ".com"`

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Figure 11.5.1: Example: Get username from email address.

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

int main() {
   string emailText;
   int atSymbolIndex;
   string emailUsername;

   cout << "Enter email address: ";
   cin  >> emailText;

   atSymbolIndex = emailText.find('@');
   if (atSymbolIndex == string::npos) {
      cout << "Address is missing @" <<
endl;
   }
   else {
      emailUsername = emailText.substr(0,
atSymbolIndex);
      cout << "Username: " << emailUsername
<< endl;
   }

   return 0;
}
```

```
Enter email address:
AbeLincoln@fakeemail.com
Username: AbeLincoln

...

Enter email address: swimming_is_fun
Address is missing @
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.5.1: find() and substr().

userText is "March 17, 2034".
Do not type quotes in answers.

1) What does userText.find(',') return?

Check Show answer

2) What does userText.find("April") return?

Check Show answer

3) What does userText.substr(0, 3) return?

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

4) What does userText.substr(userText.size() - 4, 4) return?

Check Show answer

5) In the above email username example, what was the second argument passed to substr()?

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Combining / Replacing

The string library has more functions for modifying strings.

Table 11.5.2: String modify functions, invoked as myString.push_back(c). Each increases/decreases string's length appropriately.

push_back()	***push_back***(c) appends character c to the end of a string.	`// userText is "Hello"`
		`userText.push_back('?');  // Now "Hello?"`
		`userText.size();  // Returns 6`

insert()	***insert***(indx, subStr) Inserts string subStr starting at index indx.	`// userText is "Goodbye"`
		`userText.insert(0, "Well "); // Now "Well Goodbye"`
		
		`// userText is "Goodbye"`
		`userText.insert(4, "---");  // Now "Good---bye"`

replace()	***replace***(indx, num, subStr) replaces characters at indices indx to indx+num-1 with a copy of subStr.	`// userText is "You have many gifts"`
		`userText.replace(9, 4, "a plethora of");`
		`// Now "You have a plethora of gifts"`

str1 + str2	Returns a new string that is a copy of str1 with str2 appended.
	
	If one of str1, str2 is a string, the other may be a character (or character array).
		`// userText is "A B"`
		`myString = userText + " C D";`
		`// myString is "A B C D"`
		`myString = myString + '!';`
		`// myString now "A B C D!"`
		`myString = myString + userText;`
		`// myString now "A B C D!A B"`

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Figure 11.5.2: String modify example: Greeting.

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

int main() {
   string userName;
   string greetingText;
   int itemIndex;

   itemIndex = 0;

   cout << "Enter name: ";
   getline(cin, userName);

   // Combine strings using +
   greetingText = "Hello " + userName;

   // Append a period (could have used +)
   greetingText.push_back('.'); // '' not ""
   cout << greetingText << endl;

   // Insert Mr/Ms before user's name
   greetingText.insert(6, "Mr/Ms ");
   cout << greetingText << endl;

   // Replace occurrence of "Darn" by "@$#"
   if (greetingText.find("Darn") != string::npos) { //
Found
      itemIndex = greetingText.find("Darn");
      greetingText.replace(itemIndex, 4, "@#$");
   }
   cout << greetingText << endl;

   return 0;
}
```

```
Enter name: Julia
Hello Julia.
Hello Mr/Ms Julia.
Hello Mr/Ms Julia.

...

Enter name: Darn
Rabbit
Hello Darn Rabbit.
Hello Mr/Ms Darn
Rabbit.
Hello Mr/Ms @#$
Rabbit.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.5.2: String modification functions.

str1 is "Main" and str2 is " Street" (note the leading space).

1) Use + to combine str1 and str2, so newStr should be "Main Street".

```
newStr = str1 ____;
```

Check Show answer

2) Use push_back to append period to str2, so str2 should be " Street."

```
str2.____;
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) Replace "ai" with "our" in str1, so str1 should be "Mourn". The first two arguments are just numbers.

```
str1.replace(
);
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Exploring further:

Numerous additional functions exist for strings.

- C++ string library

CHALLENGE ACTIVITY 11.5.1: Combining strings.

Assign secretID with firstName, a space, and lastName. Ex: If firstName is Barry and lastName is Allen, then output is:

```
Barry Allen
```

Learn how our autograder works

602398.4584902.qx3zqy7

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

int main() {
   string secretID;
   string firstName;
   string lastName;

   cin >> firstName;
   cin >> lastName;

   /* Your solution goes here  */

   cout << secretID << endl;
```

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.5.2: Basic string manipulation.

602398.4584902.qx3zqy7

Start

Given string userInput on one line and string toAdd on a second line, assign resultStr with the copy of userInput with toAdd appended.

Ex: If the input is:

```
Fuzzy bear
!!!
```

then the output is:

```
Fuzzy bear!!!
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main() {
   string userInput;
   string toAdd;
   string resultStr;

   getline(cin, userInput);
   getline(cin, toAdd);

   /* Your code goes here */

   cout << resultStr << endl;
```

1 2 3 4

Check Next level

11.6 Conditional expressions

This section has been set as optional by your instructor.

If-else statements with the form shown below are so common that the language supports the shorthand notation shown.

PARTICIPATION ACTIVITY 11.6.1: Conditional expression.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```

```
myVar = (condition) ? expr1 : expr2;
```

Animation content:

There are two code snippets. On the left the code snippet is:

```
if (condition) {
  myVar = expr1;
}
else {
  myVar = expr2;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The second code snippet is equivalent to the first and is on the right. It reads:
myVar = (condition) ? expr1:expr2

Animation captions:

1. An if-else statement can be written as a conditional expression.

A ***conditional expression*** has the form `condition ? exprWhenTrue : exprWhenFalse`.

All three operands are expressions. If the `condition` evaluates to true, then `exprWhenTrue` is evaluated. If the condition evaluates to false, then `exprWhenFalse` is evaluated. The conditional expression evaluates to whichever of those two expressions was evaluated. For example, if x is 2, then the conditional expression `(x == 2) ? 5 : 9 * x` evaluates to 5.

A conditional expression has three operands and thus the "?" and ":" together are sometimes referred to as a ***ternary operator***.

Good practice is to restrict usage of conditional expressions to an assignment statement, as in: y = (x == 2) ? 5 : 9 * x;.
Common practice is to put parentheses around the first expression of the conditional expression, to enhance readability.

PARTICIPATION ACTIVITY 11.6.2: Conditional expressions.

Convert each if-else statement to a single assignment statement using a conditional expression, using parentheses around the condition. Enter "Not possible" if appropriate.

1)
```
if (x > 50) {
   y = 50;
}
else {
   y = x;
}
```

y = ( ) ?
50 : x;

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2)
```
if (x < 20) {
   y = x;
}
else {
   y = 20;
}
```

y = (x < 20)

Check Show answer

3)
```
if (x < 100) {
   y = 0;
}
else {
   y = x;
}
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

4)
```
if (x < 0) {
   x = -x;
}
else {
   x = x;
}
```

Check Show answer

5)
```
if (x < 0) {
   y = -x;
}
else {
   z = x;
}
```

Check Show answer

CHALLENGE ACTIVITY | 11.6.1: Conditional expression: Print negative or positive. Full screen

602398.4584902.qx3zqy7

Drag and drop the code statement that assigns condStr with a conditional expression that evaluates to string "negative" if us is less than 0, and "non-negative" otherwise.

Ex: If the input is `-9`, then the output is:

```
-9 is negative.
```

Note: This activity includes distractors. Only one code statement on the left will be used in the final solution.

How to use this tool

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Unused

```
(userVal < 0) ? "negative" : "non-negative";
condStr = (userVal > 0) ? "negative" : "non-negative";
condStr = (userVal < 0) ? "negative" : "non-negative";
condStr = (userVal < 0) ? "non-negative" : "negative";
```

main.cpp Load default templ

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

int main() {
   string condStr;
   int userVal;
```

```
   cin >> userVal;


   cout << userVal << " is " << condStr << "." << e

   return 0;
}
```

Check

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.6.2: Conditional assignment.

Using a conditional expression, write a statement that increments numUsers if updateDirection is 1, otherwise decrements numUsers. Ex: if numUsers is 8 and updateDirection is 1, numUsers becomes 9; if updateDirection is 0, numUsers becomes 7.

Hint: Start with "numUsers = ...".

Learn how our autograder works

602398.4584902.qx3zqy7

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

int main() {
   int numUsers;
   int updateDirection;

   cin >> numUsers;
   cin >> updateDirection;

   /* Your solution goes here  */

   cout << "New value is: " << numUsers << endl;

   return 0;
```

Run

CHALLENGE ACTIVITY 11.6.3: Conditional expressions: Enter the output of the code.

602398.4584902.qx3zqy7

Start

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Type the program's output

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

int main() {
   int myNumber;
   int yourNumber;

   myNumber = 7;
   yourNumber = (myNumber >= 1) ? 5 : 6;

   cout << yourNumber << endl;

   return 0;
}
```

5

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1 2 3

Check Next

11.7 Enumerations

This section has been set as optional by your instructor.

Some variables only need to store a small set of named values. For example, a variable representing a traffic light need only store values named GREEN, YELLOW, or RED. An ***enumeration type*** (enum) declares a name for a new type and possible values for that type.

Construct 11.7.1: Enumeration type.

```
enum identifier {enumerator1, enumerator2,
...};
```

The items within the braces ("enumerators") are integer constants automatically assigned an integer value, with the first item being 0, the second 1, and so on. An enumeration declares a new data type that can be used like the built-in types int, char, etc.

Figure 11.7.1: Enumeration example.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

/* Manual controller for traffic light */
int main() {
   enum LightState {LS_RED, LS_GREEN, LS_YELLOW,
LS_DONE};
   LightState lightVal;
   char userCmd;

   lightVal = LS_RED;
   userCmd = '-';

   cout << "User commands: n (next), r (red), q
(quit)." << endl << endl;

   lightVal = LS_RED;
   while (lightVal != LS_DONE) {

      if (lightVal == LS_GREEN) {
         cout << "Green light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_YELLOW;
         }
      }
      else if (lightVal == LS_YELLOW) {
         cout << "Yellow light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_RED;
         }
      }
      else if (lightVal == LS_RED) {
         cout << "Red light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_GREEN;
         }
      }

      if (userCmd == 'r') { // Force immediate
red
         lightVal = LS_RED;
      }
      else if (userCmd == 'q') { // Quit
         lightVal = LS_DONE;
      }
   }

   cout << "Quit program." << endl;

   return 0;
}
```

```
User commands: n (next), r
(red), q (quit).

Red light  n
Green light  n
Yellow light  n
Red light  n
Green light  r
Red light  n
Green light  n
Yellow light  n
Red light  q
Quit program.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The program declares a new enumeration type named LightState. The program then declares a new variable lightVal of that type. The loop updates lightVal based on the user's input.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The example illustrates the idea of a ***state machine*** that is sometimes used in programs, especially programs that interact with physical objects, wherein the program moves among particular situations ("states") depending on input; see What is: State machine.

Because different enumerated types might use some of the same names, e.g., `enum Colors {RED, PURPLE, BLUE, GREEN};` might also appear in the same program, the program above follows the practice of prepending a distinguishing prefix, in this case "LS" (for Light State).

One might ask why the light variable wasn't simply declared as a string, and then compared with strings "GREEN", "RED", and "YELLOW". Enumerations are safer. If using a string, an assignment like `light = "ORANGE"` would not yield a compiler error, even though ORANGE is not a valid light color. Likewise, `light == "YELOW"` would not yield a compiler error, even though YELLOW is misspelled.

One could instead declare constant strings like `const string LS_GREEN = "GREEN";` or even integer values like `const int LS_GREEN = 0;` and then use those constants in the code, but an enumeration is clearer, requires less code, and is less prone to error.

Note: Each enumerator by default is assigned an integer value of 0, 1, 2, etc. However, a programmer can assign a specific value to any enumerator. Ex: `enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.7.1: Enumeration syntax.

1) Which of the following declares a new enumeration type named CarGear, with PARK, REVERSE, and DRIVE?

- `enum CarGear (PARK, REVERSE, DRIVE);`
- `enum CarGear {PARK, REVERSE, DRIVE}`
- `enum CarGear {PARK, REVERSE, DRIVE};`
- `CarGear {PARK, REVERSE, DRIVE};`

PARTICIPATION ACTIVITY 11.7.2: Enumerations.

1) Declare a new enumeration type named HvacStatus with three named values HVAC_OFF, AC_ON, FURNACE_ON, in that order.

Check Show answer

2) Declare a variable of the enumeration type HvacStatus named systemStatus.

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) Assign AC_ON to the variable systemStatus.

Check Show answer

4) What is the integer value of systemStatus after the following?

```
systemStatus = FURNACE_ON;
```

Check Show answer

5) Given `enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`, what does `cout << TC_ABC;` output?

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.7.1: Enumerations: Grocery items. Full screen

602398.4584902.qx3zqy7

Organize the lines of code to output "Fruit" if the value of userItem is a type of fruit. Otherwise, if the value of userItem is a ty drink, output "Drink". Otherwise, output "Unknown".

Ex: If the input is `GR_APPLES`, then the output is:

```
Fruit
```

How to use this tool

Unused

```
else if (userItem == GR_JUICE || userItem == GR_WATER) {
cout << "Unknown" << endl;
}
else {
if (userItem == GR_APPLES || userItem == GR_BANANAS) {
}
cout << "Drink" << endl;
cout << "Fruit" << endl;
}
```

main.cpp Load default templ

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

int main() {
   enum GroceryItem {GR_APPLES, GR_BANANAS, GR_JUIC
   int userItem;

   cin >> userItem;


   return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Check

CHALLENGE ACTIVITY 11.7.2: Soda machine with enums. Full screen

602398.4584902.qx3zqy7

The following program reads a number from input to indicate the coin inserted into a soda machine. Organize the lines of co add 10 to totalDeposit if the input is a dime. Otherwise, if the input is a nickel, add 5 to totalDeposit.

Ex: If the input is `ADD_DIME`, then the output is:

```
totalDeposit: 10
```

Note: This activity includes distractors. Not all lines of code on the left will be used in the final solution.

How to use this tool

Unused

```
}
}
totalDeposit + 10;
totalDeposit = totalDeposit + 10;
else {
else if (userInput == ADD_DIME) {
else if (userInput == ADD_NICKEL) {
totalDeposit = totalDeposit + 5;
totalDeposit + 5;
```

main.cpp Load default templ

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main() {
   enum AcceptedCoins {ADD_QUARTER, ADD_DIME, ADD_N
   int totalDeposit;
   int userInput;

   totalDeposit = 0;
   cin >> userInput;

   if (userInput == ADD_QUARTER) {
      totalDeposit = totalDeposit + 25;
   }

   else {
      cout << "Invalid coin selection." << endl;
   }
```

Check

11.8 Engineering examples

This section has been set as optional by your instructor.

Arrays can be useful in solving various engineering problems. One problem is computing the voltage drop across a series of resistors. If the total voltage across the resistors is V, then the current through the resistors will be I = V/R, where R is the sum of the resistances. The voltage drop Vx across resistor x is then Vx = I · Rx. The following program uses an array to store a user-entered set of resistance values, computes I, then computes the voltage drop across each resistor and stores each in another array, and finally prints the results.

Figure 11.8.1: Calculate voltage drops across series of resistors.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main() {
   const int NUM_RES = 5;      // Number of
resistors
   double resVals[NUM_RES];    // Ohms
   double circVolt;            // Volts
   double vDrop[NUM_RES];      // Volts
   double currentVal;          // Amps
   double sumRes;              // Ohms
   int i;                      // Loop index

   cout << "5 resistors are in series." <<
endl;
   cout << "Program calculates voltage drop
"
        << "across each resistor." << endl;

   cout << "Input voltage applied to
circuit: ";
   cin >> circVolt;

   cout << "Input ohms of " << NUM_RES << "
resistors:";
   cout << endl;
   for (i = 0; i < NUM_RES; ++i) {
      cout << (i + 1) << ") ";
      cin >> resVals[i];
   }

   // Calculate current
   sumRes = 0;
   for (i = 0; i < NUM_RES; ++i) {
      sumRes = sumRes + resVals[i];
   }
   currentVal = circVolt / sumRes;    // I
= V/R

   for (i = 0; i < NUM_RES; ++i) {
      vDrop[i] = currentVal * resVals[i];
// V = IR
   }

   cout << endl << "Voltage drop per
resistor is:" << endl;
   for (i = 0; i < NUM_RES; ++i) {
      cout << (i + 1) << ") " << fixed <<
setprecision(1)
           << vDrop[i] << " V" << endl;
   }

   return 0;
}
```

```
5 resistors are in series.
Program calculates voltage drop
across each resistor.
Input voltage applied to circuit: 12
Input ohms of 5 resistors:
1) 3.3
2) 1.5
3) 2
4) 4
5) 2.2

Voltage drop per resistor is:
1) 3.0 V
2) 1.4 V
3) 1.8 V
4) 3.7 V
5) 2.0 V
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.8.1: Voltage drop program.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1) What does variable circVolt store?

- Multiple voltages, one for each resistor.
- The resistance of each resistor.

- The total voltage across the series of resistors.

2) What does the first for loop do?

- Gets the voltage of each resistor and stores each in an array.
- Gets the resistance of each resistor and stores each in an array.
- Adds the resistances into a total value.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) What does the second for loop do?

- Adds the resistances into a single value, so that I = V/R can be computed.
- Computes the voltage across each resistor.

4) What does the third for loop do?

- Update the resistances array with new resistor values.
- Sum the voltages across each resistor into a total voltage.
- Determines the voltage drop across each resistor and stores each voltage in another array.

5) Could the fourth loop's statement have been incorporated into the third loop, thus eliminating the fourth loop?

- No, a resistor's voltage drop isn't known until the entire loop has finished.
- Yes, but keeping the loops separate is better style.

Engineering problems commonly involve matrix representation and manipulation. A matrix can be captured using a two-dimensional array. Then matrix operations can be defined on such arrays. The following illustrates matrix multiplication for 4x2 and 2x3 matrices captured as two-dimensional arrays.

Figure 11.8.2: Matrix multiplication of 4x2 and 2x3 matrices.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main() {
   const int M1_ROWS = 4;        // Matrix 1 rows
   const int M1_COLS = 2;        // Matrix 1 cols
   const int M2_ROWS = M1_COLS; // Matrix 2 rows (must have same
value)
   const int M2_COLS = 3;        // Matrix 2 cols
   int i;                        // Loop index
   int j;                        // Loop index
   int k;                        // Loop index
   int dotProd;                  // Dot product

   // Populate matrices
   int mtrx1[M1_ROWS][M1_COLS] = {{3, 4},
                                  {2, 3},
                                  {1, 5},
                                  {0, 2}};

   int mtrx2[M2_ROWS][M2_COLS] = {{5, 4, 4},
                                  {0, 2, 3}};

   int mtrx3[M1_ROWS][M2_COLS] = {{0, 0, 0},
                                  {0, 0, 0},
                                  {0, 0, 0},
                                  {0, 0, 0}};

   // mtrx3 = mtrx1 * mtrx2
   for (i = 0; i < M1_ROWS; ++i) {
      for (j = 0; j < M2_COLS; ++j) {
         // Compute dot product
         dotProd = 0;
         for (k = 0; k < M2_ROWS; ++k) {
            dotProd = dotProd + (mtrx1[i][k] * mtrx2[k][j]);
         }

         mtrx3[i][j] = dotProd;
      }
   }

   // Print m3 result
   for (i = 0; i < M1_ROWS; ++i) {
      for (j = 0; j < M2_COLS; ++j) {
         cout << setw(2) << mtrx3[i][j] << " ";
      }
      cout << endl;
   }

   return 0;
}
```

```
15 20 24
10 14 17
 5 14 19
 0  4  6
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Programmers commonly use arrays rather than vectors for problems involving two or more dimensions, because arrays are somewhat easier to declare and use, even though arrays are less safe.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.8.2: Matrix multiplication program.

1) For the first set of for loops, how many dot products are computed? (In other words, how many iterations are due to the outer two for loops?)

Check Show answer

2) For the first set of for loops, the inner-most loop computes a dot product. Each time the inner-most loop is reached, how many times will the inner-most loop iterate?

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Check Show answer

11.9 Engineering examples using functions

This section has been set as optional by your instructor.

This section contains examples of functions for various engineering calculations.

Gas equation

An equation used in physics and chemistry that relates pressure, volume, and temperature of a gas is PV = nRT. P is the pressure, V the volume, T the temperature, n the number of moles, and R a constant. The function below outputs the temperature of a gas given the other values.

Figure 11.9.1: PV = nRT. Compute the temperature of a gas.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

const double GAS_CONSTANT = 8.3144621;  //  J /
(mol * K)

/* Converts a pressure, volume, and number of
moles
 of a gas into a temperature. */
double PVnToTemp(double gasPressure, double
gasVolume, double numMoles) {
   return (gasPressure * gasVolume) / (numMoles *
GAS_CONSTANT);
}

int main() {
   double gasPress;        // User defined
pressure
   double gasVol;          // User defined volume
   double gasMoles;        // User defined moles

   // Prompt user for input parameters
   cout << "Enter gas pressure (in Pascals): ";
   cin >> gasPress;

   cout << "Enter gas volume (in cubic meters): ";
   cin >> gasVol;

   cout << "Enter number of moles: ";
   cin >> gasMoles;

   // Call function to calculate temperature
   cout << "Temperature = ";
   cout << fixed << setprecision(2)
        << PVnToTemp(gasPress, gasVol, gasMoles)
<< " K"
        << endl;

   return 0;
}
```

```
Enter gas pressure (in
Pascals): 2500
Enter gas volume (in cubic
meters): 35.5
Enter number of moles: 18
Temperature = 593.01 K
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.9.1: PV = nRT calculation.

Questions refer to PVnToTemp() above.

1) PVnToTemp() uses a rewritten form of PV = nRT to solve for T, namely T = PV/nR.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) PVnToTemp() uses a constant variable for the gas constant R.

- True
- False

3) TempVolMolesToPressure() would likely return (temp * vlm) / (mols * GAS_CONSTANT).

- True
- False

Projectile location

Common physics equations determine the x and y coordinates of a projectile object at any time, given the object's initial velocity and angle at time 0 with initial position x = 0 and y = 0. The equation for x is v * t * cos(a). The equation for y is v * t * sin(a) - 0.5 * g * t * t. The following provides a single function to compute an object's position; because position consists of two values (x and y), the function uses pass by reference parameters to return values for x and y. The program's main function asks the user for the object's initial velocity, angle, and height (y position), and then prints the object's position for every second until the object's y position is no longer greater than 0 (meaning the object fell back to earth).

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Figure 11.9.2: Trajectory of object on Earth.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

// Note: 1-letter variable names are typically avoided,
// but used below where standard in physics.

const double PI_CONST = 3.14159265;

// Given time, angle, velocity, and gravity
// Update x and y values
void ObjectTrajectory(double t, double a, double v,
double g,
                 double& x, double& y) {
   x = v * t * cos(a);
   y = v * t * sin(a) - 0.5 * g * t * t;
}

// Convert degree value to radians
double DegToRad(double inDeg) {
   return ((inDeg * PI_CONST) / 180.0);
}

int main() {
   const double GRAVITY  = 9.8; // Earth gravity (m/s^2)
   double launchAngle;          // Angle of launch (rad)
   double launchVelocity;       // Velocity (m/s)
   double elapsedTime;          // Time (s)
   double yLoc;                 // Obj's location above
ground (m)
   double xLoc;                 // Obj's horiz. loc.
from start (m)

   elapsedTime = 1.0;
   xLoc = 0.0;

   cout << "Launch angle (deg): ";
   cin >> launchAngle;
   launchAngle = DegToRad(launchAngle); // To radians

   cout << "Launch velocity (m/s): ";
   cin >> launchVelocity;

   cout << "Initial height (m): ";
   cin >> yLoc;

   while ( yLoc > 0.0 ) { // While above ground
      cout << "Time " << fixed << setprecision(0)
      << setw(3) << elapsedTime
      << "   x = " << setw(3) << xLoc
      << "   y = " << setw(3) << yLoc
      << endl;
      ObjectTrajectory(elapsedTime, launchAngle,
launchVelocity,
                       GRAVITY, xLoc, yLoc);
      elapsedTime = elapsedTime + 1.0;
   }

   return 0;
}
```

```
Launch angle (deg): 45
Launch velocity (m/s):
100
Initial height (m): 3
Time   1   x =   0   y
=   3
Time   2   x =  71   y
=  66
Time   3   x = 141   y
= 122
Time   4   x = 212   y
= 168
Time   5   x = 283   y
= 204
Time   6   x = 354   y
= 231
Time   7   x = 424   y
= 248
Time   8   x = 495   y
= 255
Time   9   x = 566   y
= 252
Time  10   x = 636   y
= 239
Time  11   x = 707   y
= 217
Time  12   x = 778   y
= 185
Time  13   x = 849   y
= 143
Time  14   x = 919   y
=  91
Time  15   x = 990   y
=  30
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.9.2: Projectile location.

Questions refer to ObjectTrajectory() above.

1) ObjectTrajectory() cannot return two values (for x and y), so instead takes x and y as modifiable parameters and changes their values.

- True
- False

2) ObjectTrajectory() could replace double types by int types without causing much change in computed values.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

- True
- False

3) Each iteration of the loop will see yLoc increase.

- True
- False

4) Assuming the launch angle is less than 90 degrees, each iteration of the loop will see xLoc increase.

- True
- False

CHALLENGE ACTIVITY 11.9.1: Function to compute gas volume.

Define a function ComputeGasVolume that returns the volume of a gas given parameters pressure, temperature, and moles. Use the gas equation PV = nRT, where P is pressure in Pascals, V is volume in cubic meters, n is number of moles, R is the gas constant 8.3144621 (J / (mol*K)), and T is temperature in Kelvin. All parameter types and the return type are double.

Learn how our autograder works

602398.4584902.qx3zqy7

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

const double GAS_CONST = 8.3144621;

/* Your solution goes here  */

int main() {
   double gasPressure;
   double gasMoles;
   double gasTemperature;
   double gasVolume;

   cin >> gasPressure;
   cin >> gasMoles;
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Run

11.10 Structs and vectors

This section has been set as optional by your instructor.

The power of structs becomes even more evident when used in conjunction with vectors. Consider a TV watching-time program where a user can enter a country name, and the program outputs the average daily TV-watching hours for a person in that country. One approach is to use two same-sized vectors, one to hold names, and the other to hold numbers corresponding to each name. Instead of those two vectors, a struct allows for the declaration of just one vector that stores items that each have a name and number data member.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Figure 11.10.1: A vector of struct items rather than two vectors of more basic types.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

struct CountryTvWatch {
   string countryName;
   int tvMinutes;
};

int main() {
   // Source: www.statista.com, 2010
   const int NUM_COUNTRIES = 4;

   vector<CountryTvWatch> countryList(NUM_COUNTRIES);
   string countryToFind;
   bool countryFound;
   int i;

   countryFound = false;

   countryList.at(0).countryName = "Brazil";
   countryList.at(0).tvMinutes = 222;
   countryList.at(1).countryName = "India";
   countryList.at(1).tvMinutes = 119;
   countryList.at(2).countryName = "U.K.";
   countryList.at(2).tvMinutes = 242;
   countryList.at(3).countryName = "U.S.A.";
   countryList.at(3).tvMinutes = 283;

   cout << "Enter country name: ";
   cin >> countryToFind;

   for (i = 0; i < NUM_COUNTRIES && !countryFound; ++i) {
// Find country's index
      if (countryList.at(i).countryName == countryToFind)
{
         countryFound = true;
         cout << "People in " << countryToFind << endl;
         cout << "watch " << countryList.at(i).tvMinutes;
         cout << " minutes of TV daily." << endl;
      }
   }
   if (!countryFound) {
      cout << "Country not found, try again." << endl;
   }

   return 0;
}
```

```
Enter country name:
U.S.A.
People in U.S.A.
watch 283 minutes
of TV daily.
...
Enter country name:
UK
Country not found,
try again.
...
Enter country name:
U.K.
People in U.K.
watch 242 minutes
of TV daily.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The countryList variable is declared as `vector<CountryTvWatch> countryList(NUM_COUNTRIES)`, meaning a vector of items of type CountryTvWatch. Thus, each vector element will have memory allocated for the struct's two data members, countryName and tvMinutes.

The notation countryList.at(i).countryName is equivalent to (countryList.at(i)).countryName, because the member access operator is evaluated left-to-right (as are any equal precedence operators). The left-to-right member access operator evaluation is well-known among programmers so parentheses are typically omitted.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.10.1: Using structs with vectors.

Use .at() notation for vector element access.

1) Declare a vector countryList of 5 CountryTvWatch elements

Check Show answer

2) Given a vector countryList consisting of 5 CountryTvWatch struct elements, write a statement that assigns the value of the 0th element's tvMinutes data member to the variable countryMin.

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) Given a vector countryList consisting of 5 CountryTvWatch struct elements, write one statement that copies element 4's struct values to element 0's.

Check Show answer

zyDE 11.10.1: Modify the TV watch program.

Finish the PrintCountryNames() function to print all country names in the list.

Load default template...

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

struct CountryTvWatch {
   string countryName;
   int tvMinutes;
};

void PrintCountryNames(vec
{
   cout << "FIXME: Finish
}
```

USA

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.10.1: Enter the output of the struct and vector.

602398.4584902.qx3zqy7

Start

Type the program's output

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

struct Car {
   string type;
   string color;
};

int main() {
   vector<Car> carList(3);

   carList.at(0).type = "sedan";
   carList.at(0).color = "blue";
   carList.at(1).type = "SUV";
   carList.at(1).color = "pink";
   carList.at(2).type = "truck";
   carList.at(2).color = "brown";

   cout << carList.at(0).color << " " << carList.at(0).type << endl;
   cout << carList.at(2).color << " " << carList.at(2).type << endl;

   return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
blue sedan
brown truck
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 11.10.2: Structs and vectors.

602398.4584902.qx3zqy7

Start

Declare a vector named pizzasInStore that stores 3 items of type PizzaIngredients.

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

struct PizzaIngredients {
   string pizzaName;
   string ingredients;
};

int main() {

   /* Your code goes here */

   pizzasInStore.at(0).pizzaName = "Barbecue";
   pizzasInStore.at(1).pizzaName = "Carbonara";
   pizzasInStore.at(2).pizzaName = "Ham and Cheese";
   pizzasInStore.at(0).ingredients = "Beef, chicken, bacon, barbecue sau
   pizzasInStore.at(1).ingredients = "Mushrooms, onion, creamy sauce";
   pizzasInStore.at(2).ingredients = "Ham, cheese, bacon";

   cout << pizzasInStore.at(0).pizzaName << ": " << pizzasInStore.at(0).
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
   cout << pizzasInStore.at(1).pizzaName << ": " << pizzasInStore.at(1).
   cout << pizzasInStore.at(2).pizzaName << ": " << pizzasInStore.at(2).

   return 0;
}
```

1 2 3

Check Next

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.11 Structs, vectors, and functions: A seat reservation example

This section has been set as optional by your instructor.

A programmer commonly uses structs, vectors, and functions together. Consider a program that allows a reservations agent to reserve seats for people, useful for a theater, an airplane, etc. The below program defines a Seat struct whose data members are a person's first name, last name, and the amount paid for the seat. The program declares a vector of 5 seats to represent the theater, airplane, etc., initializes all seats to being empty (indicated by a first name of "empty"), and then allows a user to enter commands to print all seats, reserve a seat, or quit.

Figure 11.11.1: A seat reservation system involving a struct, vector, and functions.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: empty empty, Paid:
0
3: empty empty, Paid:
0
4: empty empty, Paid:
0

Enter command (p/r/q):
r
Enter seat num: 2
Enter first name: John
Enter last name: Smith
Enter amount paid: 500
Completed.

Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: John Smith, Paid:
500
3: empty empty, Paid:
0
4: empty empty, Paid:
0

Enter command (p/r/q):
r
Enter seat num: 2
Seat not empty.

Enter command (p/r/q):
r
Enter seat num: 3
Enter first name: Mary
Enter last name: Jones
Enter amount paid: 198
Completed.

Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: John Smith, Paid:
500
3: Mary Jones, Paid:
198
4: empty empty, Paid:
0

Enter command (p/r/q):
q
Quitting.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

struct Seat {
   string firstName;
   string lastName;
   int amountPaid;
};

/*** Functions for Seat ***/
void SeatMakeEmpty(Seat& seat) {
   seat.firstName   = "empty";
   seat.lastName    = "empty";
   seat.amountPaid  = 0;
}

bool SeatIsEmpty(Seat seat) {
   return(seat.firstName == "empty");
}

void SeatPrint(Seat seat) {
   cout << seat.firstName << " ";
   cout << seat.lastName << ", ";
   cout << "Paid: " << seat.amountPaid << endl;
}
/*** End functions for Seat ***/

/*** Functions for vector of Seat ***/
void SeatsMakeEmpty(vector<Seat>& seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      SeatMakeEmpty(seats.at(i));
   }
}

void SeatsPrint(vector<Seat> seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      cout << i << ": ";
      SeatPrint(seats.at(i));
   }
}
/*** End functions for vector of Seat ***/

int main() {
   char userKey;
   int  seatNum;
   vector<Seat> allSeats(5);
   Seat currSeat;

   userKey = '-';

   SeatsMakeEmpty(allSeats);

   while (userKey != 'q') {

      cout << endl << "Enter command (p/r/q): ";
      cin >> userKey;

      if (userKey == 'p') { // Print seats
         SeatsPrint(allSeats);
      }
      else if (userKey == 'r') { // Reserve seat
         cout << "Enter seat num: ";
         cin  >> seatNum;

         if (!SeatIsEmpty(allSeats.at(seatNum)))
{
            cout << "Seat not empty." << endl;
         }
         else {
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
         else {
            cout << "Enter first name: ";
            cin >> currSeat.firstName;
            cout << "Enter last name: ";
            cin >> currSeat.lastName;
            cout << "Enter amount paid: ";
            cin >> currSeat.amountPaid;

            allSeats.at(seatNum) = currSeat;
            cout << "Completed." << endl;
         }
      }
      // FIXME: Add option to delete
reservations
      else if (userKey == 'q') {
         cout << "Quitting." << endl;
      }
      else {
         cout << "Invalid command." << endl;
      }
   }

   return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The programmer first defined several functions related to the Seat struct, such as checking if a seat is empty or printing a seat. The programmer then defined some functions related to a *vector* of seat items. To distinguish, the programmer named the former starting with Seat and the latter starting with Seats.

The programmer left a "FIXME" comment indicating that the program also requires the ability to delete a reservation. That functionality is straightforward to introduce, just requiring the user to enter a seat number and then making use of the existing SeatMakeEmpty() function.

Notice how main() is relatively clean, dealing mostly with the user commands, and then using functions to carry out the appropriate work. Actually, the "reserve seat" command could be improved; main() currently fills the reservation information (e.g., "Enter first name..."), but main() would be cleaner if it just called a function such as `SeatFillReservationInfo(currSeat)`.

The seat reservation program loses all its information when exited. An improvement is to save all reservation information in a file. Commands 's' and 'g' would save and get information to/from a file, respectively.

PARTICIPATION ACTIVITY 11.11.1: Seat reservation example with struct, vector, and functions.

Refer to the above example.

1) The number of seats is 5.

- True
- False

2) SeatsMakeEmpty(seats) has a loop that sets each seat in the seats vector to have a first name of "empty".

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) SeatIsEmpty() checks if all the seats in the vector are empty.

- True

False

4) Deleting a reservation would reduce the vector size from 5 down to 4.

True

False

zyDE 11.11.1: Introduce delete behavior to the reservation program.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Modify main() to allow the user to enter command 'd', followed by the user entering a seat number. Call SeatMakeEmpty() to delete the seat.

Load default template...

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

struct Seat {
   string firstName;
   string lastName;
   int amountPaid;
};

/*** Functions for Seat *
void SeatMakeEmpty(Seat& s
   seat.firstName   = "emp
   seat.lastName    = "emp
```

```
p
r 2 John Smith 500
p
```

Run

11.12 Command-line arguments

This section has been set as optional by your instructor.

Command-line arguments are values entered by a user when running a program from a command line. A *command line* exists in some program execution environments, wherein a user types a program's name and any arguments at a command prompt. To access those arguments, main() can be defined with two special parameters argc and argv, as shown below. The program prints provided command-line arguments. (The "for" loop is not critical to understanding the point, in case you haven't studied for loops yet). The program's executable is named argtest.

Figure 11.12.1: Printing command-line arguments.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main(int argc, char* argv[]) {
   int i;

   // Prints argc and argv values
   cout << "argc: " << argc << endl;
   for (i = 0; i < argc; ++i) {
      cout << "argv[" << i << "]: " << argv[i] << endl;
   }

   return 0;
}
```

```
> ./argtest
argc: 1
argv[0]: ./argtest

> ./argtest Hello
argc: 2
argv[0]: ./argtest
argv[1]: Hello

> ./argtest Hey ABC 99 -5
argc: 5
argv[0]: ./argtest
argv[1]: Hey
argv[2]: ABC
argv[3]: 99
argv[4]: -5
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

When a program is run, the system passes an int parameter ***argc*** to main(), indicating the number of command-line arguments (argc is short for argument count). The number includes the program name itself, so argc is 2 for the command line: `myprog.exe myfile.txt.`

When a program is run, the system passes a second parameter ***argv*** to main() (argv is short for argument vector), defined as an array of strings: char* argv[]. argv[] consists of one string for each command-line argument, with argv[0] being the program name.

PARTICIPATION ACTIVITY 11.12.1: Command-line arguments.

argc = 3
argv[0] = myprog.exe
argv[1] = userArg1
argv[2] = userArg2

Animation content:

Static figure: A computer keyboard and a command-line are displayed.
Step 1: The system passes an int parameter argc to main(), which indicates the number of command-line arguments. The text, > myprog.exe userArg1 userArg2, is displayed in the command-line. Four lines of code are displayed below the command-line:
argc =
argv[0] =
argv[1] =
argv[2] =
A 3 appears next to the command-line and moves to the line of code, argc =, which is now argc = 3.
Step 2: The system passes a second parameter argv to main(), which is an array of strings consisting

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

of one string for each command-line argument. The command-line argument, myprog.exe, is highlighted and a copy of the text moves to the line of code, argv[0] =, which is now argv[0] = myprog.exe. The command-line argument, userArg1, is highlighted and a copy of the text moves to the line of code, argv[1] =, which is now argv[1] = userArg1. The command-line argument, userArg2, is highlighted and a copy of the text moves to the line of code, argv[2] =, which is now argv[2] = userArg2.

Animation captions:

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1. The system passes an int parameter argc to main(), which indicates the number of command-line arguments.
2. The system passes a second parameter argv to main(), which is an array of strings consisting of one string for each command-line argument.

PARTICIPATION ACTIVITY 11.12.2: Command-line arguments.

1) What is argc for: :

```
myprog.exe 13 14 smith
```

Check Show answer

2) What is argc for:

```
a.out 12:55 PM
```

Check Show answer

3) What is the string in argv[2] for:

```
a.out Jan Feb Mar
```

Check Show answer

The following program, named myprog, expects two command-line arguments.

Figure 11.12.2: Simple use of command-line arguments.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

/* Usage: program username userage */
int main(int argc, char* argv[]) {
   string nameStr;  // User name
   string ageStr;   // User age

   // Get inputs from command line
   nameStr = argv[1];
   ageStr  = argv[2];

   // Output result
   cout << "Hello " << nameStr << ". ";
   cout << ageStr << " is a great age." <<
endl;

   return 0;
}
```

```
> myprog.exe Amy 12
Hello Amy. 12 is a great age.

...

> myprog.exe Rajeev 44 HEY
Hello Rajeev. 44 is a great
age.

...

> myprog.exe Denming
Segmentation fault
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

However, there is no guarantee a user will type two command-line arguments. Extra arguments, like "HEY" above, are ignored. Conversely, too few arguments can cause a problem. In particular, *a common error is to access elements in argv without first checking argc to ensure the user entered enough arguments, resulting in an out-of-range array access.* In the last run above, the user typed too few arguments, causing an out-of-range array access.

When a program uses command-line arguments, good practice is to check argc for the correct number of arguments. If the number of command-line arguments is incorrect, good practice is to print a usage message. A ***usage message*** lists a program's expected command-line arguments. The program should then return 1, indicating to the system that an error occurred (whereas 0 means no error).

Figure 11.12.3: Checking for proper number of command-line arguments.

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

/* Usage: program username userage */
int main(int argc, char* argv[]) {
   string nameStr;  // User name
   string ageStr;   // User age

   // Check if correct number of arguments
provided
   if (argc != 3) {
      cout << "Usage: myprog.exe name age" <<
endl;
      return 1; // 1 indicates error
   }

   // Get inputs from command line
   nameStr = argv[1];
   ageStr = argv[2];

   // Output result
   cout << "Hello " << nameStr << ". ";
   cout << ageStr << " is a great age." << endl;

   return 0;
}
```

```
> myprog.exe Amy 12
Hello Amy. 12 is a great
age.

...

> myprog.exe Denming
Usage: myprog.exe name age

...

> myprog.exe Alex 26 pizza
Usage: myprog.exe name age
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.12.3: Checking the number of command-line arguments.

1) If a user types the wrong number of command-line arguments, good practice is to print a usage message.

- True
- False

2) If a user types too many arguments but a program doesn't check for that, the program typically crashes.

- True
- False

3) If a user types too few arguments but a program doesn't check for that, the program typically crashes.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Command-line arguments are C strings. The (rather cumbersome) statement `age = atoi(ageStr.c_str());` converts the ageStr string into an integer, assigning the result into int variable age. So string "12" becomes integer 12. c_str() gets a C string version of ageStr. atoi() converts a C string to an integer, and is made available via `#include <cstdlib>`.

Putting quotes around an argument allows an argument's string to have any number of spaces.

Figure 11.12.4: Quotes surround the single argument 'Mary Jo'.

```
myprog.exe "Mary Jo"
50
```

PARTICIPATION ACTIVITY 11.12.4: String and integer command-line arguments.

1) What is the string in argv[1] for the following:

`a.out Amy Smith 19`

Check Show answer

2) What is the string in argv[1] for the following:

`a.out "Amy Smith" 19`

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) Given the following code snippet, complete the assignment of userNum with argv[1].

```
int main(int argc, char*
argv[]) {
   int userNum;
   userNum = /* COMPLETE
ASSIGNMENT */;
}
```

```
userNum =
                    ;
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Exploring further:

- Command-line arguments from cplusplus.com

11.13 Command-line arguments and files

This section has been set as optional by your instructor.

The location of an input file or output file may not be known before writing a program. Instead, a program can use command-line arguments to allow the user to specify the location of an input or output file as shown in the following program. Assume two text files exist in the same directory as the source code named words1.txt and words2.txt with the contents shown. The sample output shows the results when executing the program for each input file and for an input file that does not exist.

Figure 11.13.1: Using command-line arguments to specify the name of an input file.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main(int argc, char* argv[]) {
   ifstream inFS;
   string userWord;
   int wordFreq = 0;
   string currWord;

   // Check number of arguments
   if (argc != 2) {
      cout << endl << "Usage: myprog.exe
inputFileName" << endl;
      return 1;   // 1 indicates error
   }

   // Try to open file
   cout << "Opening file " << argv[1] << "." <<
endl;
   inFS.open(argv[1]);

   if (!inFS.is_open()) {
      cout << "Could not open file " << argv[1]
<< "." << endl;
      return 1;
   }

   // Word to be found
   cout << "Enter a word: ";
   cin >> userWord;

   // Identify when a word matches the userWord
   // and increase wordFreq
   while (!inFS.eof()) {
      inFS >> currWord;
      if (!inFS.fail()) {
         if (currWord == userWord) {
            ++wordFreq;
         }
      }
   }

   cout << userWord << " appears in " <<
argv[1] << " "
      << wordFreq << " times." << endl;

   // Done with file, so close it
   inFS.close();

   return 0;
}
```

words1.txt:

```
twenty
associable
twenty
unredacted
associable
folksay
twenty
```

words2.txt:

```
apples
bananas
oranges
```

```
> myprog.exe words1.txt
Opening file words1.txt.
Enter a word: twenty
twenty appears in
words1.txt 3 times.

...

> myprog.exe words2.txt
Opening file words2.txt.
Enter a word: twenty
twenty appears in
words2.txt 0 times.

...

> myprog.exe words3.txt
Opening file words3.txt.
Could not open file
words3.txt.
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.13.1: Filename command-line arguments.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1) A program "myprog.exe" has two command-line arguments, one for an input file and a second for an output file. Type a command to run the program with input file "infile.txt" and output file "out".

Check Show answer

2) For a program run as `progname data.txt`, what is argv[1]? Don't use quotes in your answer.

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.14 The #define directive

This section has been set as optional by your instructor.

The ***#define*** directive, of the form #define MACROIDENTIFIER replacement, instructs the processor to replace any occurrence of MACROIDENTIFIER in the subsequent program code by the replacement text.

Construct 11.14.1: #define directive.

```
#define MACROIDENTIFIER
replacement
```

#define is sometimes called a ***macro***. The #define line does *not* end with a semicolon.

Most uses of #define are strongly discouraged by experienced programmers. However, for legacy reasons, #define appears in much existing code, so a programmer still benefits from understanding #define.

One (discouraged) use of #define is for a constant. The directive `#define MAXNUM 99` causes the preprocessor to replace every occurrence of identifier MAXNUM by 99, before continuing with compilation. So `if (x < MAXNUM)` will be replaced by if `(x < 99)`. In contrast, declaring a constant variable `const int MAXNUM = 99` has several advantages over a macro, such as type checking, syntax errors for certain incorrect usages, and more.

Another (discouraged) use of #define is for a type-neutral function. #define may specify arguments, as in `#define FCT1(a, b) ((a + b)/(a * b))`. A program may then have statements like `numInt = FCT1(1,2)` or like `numFloat = FCT1(1.2, 0.7)`, which the preprocessor would replace by `numInt = ((1 + 2) / (1 * 2))` or `numFloat = ((1.2 + 0.7) / (1.2 * 0.7))`, respectively. However, defining functions for each type, or overloading a function name (discussed elsewhere), is better practice.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit

Some advanced techniques make good use of macros, and are mentioned in other sections, such as a section introducing the assert macro.

PARTICIPATION ACTIVITY 11.14.1: #define.

Given:

```
#define PI_CONST  3.14159

double CalcCircleArea(double radius) {
   return PI_CONST * radius * radius;
}
```

1) The function call CalcCircleArea(1.0) returns 3.14159.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) Replacing the return expression by `PI_CONST * PI_CONST * radius` yields a compiler error since only one replacement is allowed.

- True
- False

3) Replacing the return expression by `PI_CONSTPI_CONST` would yield 3.141593.14159, which would thus yield a compiler error complaining about the two decimal points.

- True
- False

4) The call CalcCircleArea(PI_CONST) would yield a compiler error complaining that the macro cannot be an argument.

- True
- False

5) Placing a semicolon at the end of the #define line will yield a compiler error at that line.

- True
- False

Exploring further:

- Other preprocessor directives include ***#undef***, ***#ifdef***, ***#if***, ***#else***, ***#elif***, ***#pragma***, ***#line***, and ***#error***.
- Preprocessor tutorial on cplusplus.com
- Preprocessor directives on MSDN

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

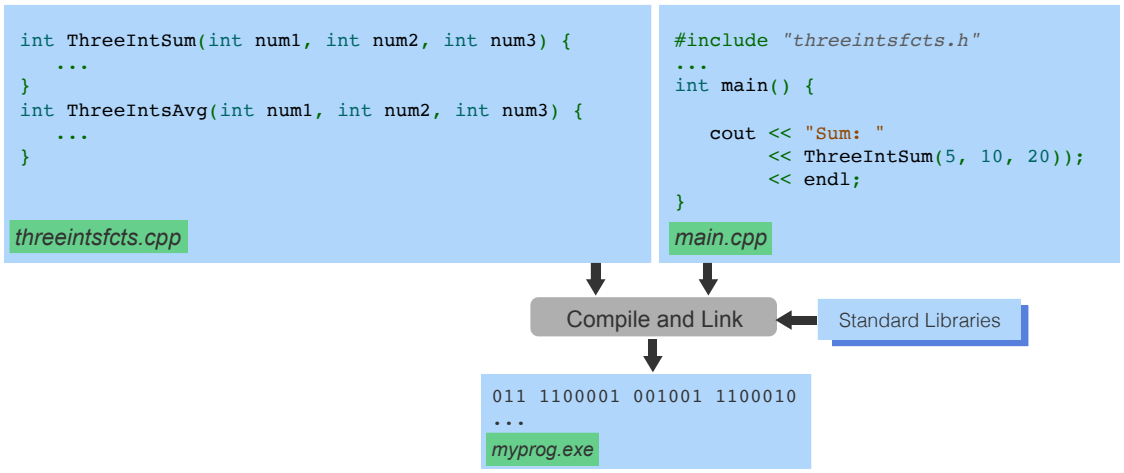
11.15 Modular compilation

This section has been set as optional by your instructor.

Using separate files to organize code can also help to make the compilation process more efficient. So far, we used a ***single-step compilation*** approach in which all source files are compiled at the same time to create the executable, e.g. `g++ main.cpp threenumsfcts.cpp`. This approach has a significant drawback. Anytime one of the source files is modified, all files must be recompiled. Even a small change, such as modifying a cout statement to fix a spelling error, would require all files be recompiled. The following animation illustrates the single-step compilation.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.15.1: Single-step compilation process.



Animation content:

Static figure: A code block labeled threeintsfcts.cpp and a code black labeled main.cpp are displayed.
threeintsfcts.cpp code block:
Begin C++ code:

```
int ThreeIntSum(int num1, int num2, int num3) {
   ...
}
int ThreeIntsAvg(int num1, int num2, int num3) {
   ...
}
```

End C++ code.
main.cpp code block:
Begin C++ code:

```
#include "threeintsfcts.h"
...
int main() {

  cout << ThreeIntSum(5, 10, 20));
       << endl;
}
```

End C++ code.
Step 1: Single-step compilation compiles and links all source files and libraries. The text, Compile and Link, is displayed below the code blocks. The text, Standard Libraries, is displayed with an arrow

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

pointing to Compile and Link. Two arrows appear pointing threeintsfcts.cpp and main.cpp to the text, Compile and Link. An arrow appears pointing the text, Compile and Link, to a new code block labeled myprog.exe which contains two lines:
011 1100001 001001 1100010
...
Step 2: Change made to main.cpp requires that main.cpp be recompiled. But, single-step compilation will recompile all source files. Everything except the code blocks, threeintsfcts.cpp and main.cpp, disappears. The lines of code in main.cpp, cout << ThreeIntSum(5, 10, 20));, << endl;, are highlighted and replaced with the lines of code, cout << "Sum: ", << ThreeIntSum(5, 10, 20));, << endl;. The text, Compile and Link, is displayed below the code blocks. The text, Standard Libraries, is displayed with an arrow pointing to Compile and Link. Two arrows appear pointing threeintsfcts.cpp and main.cpp to the text, Compile and Link. An arrow appears pointing the text, Compile and Link, to a new code block labeled myprog.exe which contains two lines:
011 1100001 001001 1100010
...

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Animation captions:

1. Single-step compilation compiles and links all source files and libraries.
2. Change made to main.cpp requires that main.cpp be recompiled. But, single-step compilation will recompile all source files.

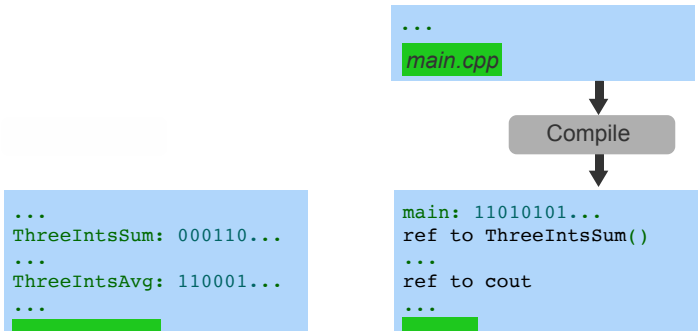
As programs become larger and the number of source files increases, the time required to recompile and link all source files can become very long - often requiring minutes to hours. Instead of compiling an executable using a single step, a ***modular compilation*** approach can be used that separates the compiling and linking steps within the compilation process. In this approach, each source file is independently compiled into an object file. An ***object file*** contains machine instructions for the compiled code along with placeholders, often referred to as references, for calls to functions or accesses to variables or classes defined in other source files or libraries.

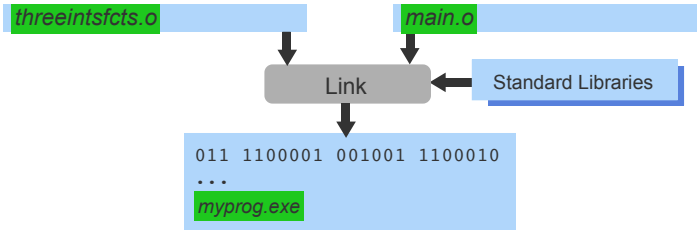
For a program involving two files main.cpp and threenumsfcts.cpp, the files can be compiled separately into two object files named main.o and threenumsfcts.o respectively. The resulting object files will include several placeholders for functions that are defined in other files. For example, the main.o object file may contain placeholders for calls to any functions in threenumsfcts.o.

After each source file has been compiled, the ***linker*** will create the final executable by ***linking*** together the object files and libraries. For each placeholder found within an object file, the linker searches the other object files and libraries to find the referenced function or variable. When linking the main.o and threenumsfcts.o files, the placeholder for a call to a function in main.o will be replaced with a jump to the first instruction of that function within the threenumsfcts.o object file. This creates a link between the code within the main.o and threenumsfcts.o object files. Once all placeholders have been linked together, the final executable can be created. The following animation illustrates.

PARTICIPATION ACTIVITY 11.15.2: Modular compilation process.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024





©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Animation content:

Static figure: A code block labeled threeintsfcts.cpp and a code black labeled main.cpp are displayed.
threeintsfcts.cpp code block:
Begin C++ code:

```
int ThreeIntSum(int num1, int num2, int num3) {
   ...
}
int ThreeIntsAvg(int num1, int num2, int num3) {
   ...
}
```

End C++ code.
main.cpp code block:
Begin C++ code:

```
#include "threeintsfcts.h"
...
int main() {

   cout << ThreeIntSum(5, 10, 20));
          << ends;
   …
}
```

End C++ code.
Step 1: Modular compilation separates the compiling and linking. Source files are first compiled to object files. An arrow appears pointing threeintsfcts.cpp to the text, Compile. An arrow appears pointing the text, Compile, to a new code block labeled threeintsfcts.o which contains five lines:
...
ThreeIntsSum: 000110...
...
ThreeIntsAvg: 110001...
...
An arrow appears pointing main.cpp to the text, Compile. An arrow appears pointing the text, Compile, to a new code block labeled main.o which contains five lines:
main: 11010101...
ref to ThreeIntsSum()
...
ref to cout
...
Step 2: Linking process resolves references in object files and libraries to create a final executable. The text, Link, is displayed below the code blocks. The text, Standard Libraries, is displayed with an arrow pointing to Link. Two arrows appear pointing threeintsfcts.o and main.o to the text, Link.
Step 3: Linker will search object files and libraries to find the referenced functions or variables. The line of code in main.o, ref to ThreeIntsSum(), and the line of code in threeintsfcts.o, ThreeIntsSum:

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

000110..., are both highlighted. The line of code in main.o, ref to cout, and the text, Standard Libraries, are both highlighted.
Step 4: Once the object files have been linked together, the final executable is created. An arrow appears pointing the text, Link, to a new code block labeled myprog.exe which contains two lines:
011 1100001 001001 1100010
...
Step 5: Changes to main.cpp only require that main.cpp be recompiled. threeintsfcts.cpp does not need to be recompiled. Executable is re-linked using new main.o object file and existing threeintsfcts.o object file. Everything except the code block, main.cpp, disappears. The lines of code in main.cpp, cout << ThreeIntSum(5, 10, 20));, << endl;, are highlighted and replaced with the lines of code, cout << "Sum: ", << ThreeIntSum(5, 10, 20));, << endl;. An arrow appears pointing main.cpp to the text, Compile. An arrow appears pointing the text, Compile, to a new code block labeled main.o which contains five lines:
main: 11010101...
ref to ThreeIntsSum()
...
ref to cout
...
A new code block labeled threeintsfcts.o is displayed and contains five lines:
...
ThreeIntsSum: 000110...
...
ThreeIntsAvg: 110001...
...
The text, Link, is displayed below the code blocks. The text, Standard Libraries, is displayed with an arrow pointing to Link. Two arrows appear pointing threeintsfcts.o and main.o to the text, Link. An arrow appears pointing the text, Link, to a new code block labeled myprog.exe which contains two lines:
011 1100001 001001 1100010
...

Animation captions:

1. Modular compilation separates the compiling and linking. Source files are first compiled to object files.
2. Linking process resolves references in object files and libraries to create a final executable.
3. Linker will search object files and libraries to find the referenced functions or variables.
4. Once the object files have been linked together, the final executable is created.
5. Changes to main.cpp only require that main.cpp be recompiled. threeintsfcts.cpp does not need to be recompiled. Executable is re-linked using new main.o object file and exisiting threeintsfcts.o object file.

zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Using a modular compilation approach has the benefit of reducing the time required to recompile and link the program executable. Instead of recompiling all source files, only the source files that have been modified need to be recompiled to create an updated object file. The linker can then use these newly recompiled object files along with the previously compiled object files for any unmodified source files to create the updated executable.

zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.15.3: Modular compilation.

1) A single-step compilation approach is faster than a modular compilation approach.

- True
- False

2) An object file contains machine instructions for the compiled code, along with references to functions or access to variables or classes in other files or libraries.

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) A linker is responsible for creating object files.

- True
- False

11.16 Makefiles

This section has been set as optional by your instructor.

While modular compilation offers many advantages, manually compiling and linking a program with numerous source and header files can be challenging. For example, a change within a single header file will require recompiling all source files that include that header file. Keeping track of these dependencies to determine which files need to be recompiled is not trivial and can require considerable effort, if done manually. For large programs, programmers often utilize ***project management tools*** to automate the compilation and linking process. ***make*** is one project management tool that is commonly used on Unix and Linux computer systems.

The make utility uses a ***makefile*** to recompile and link a program whenever changes are made to the source or header files. The makefile consists of a set of rules and commands. ***Make rules*** are used to specify dependencies between a target file (e.g., object files and executable) and a set of prerequisite files that are needed to generate the target file (e.g., source and header files). A make rule can include one or more commands -- referred to as ***make recipe*** -- that will be executed in order to generate the target file. The following shows the general form for a make rule. The make rule's target and prerequisites are defined on a single line. The commands for the make rule should be specified one per line starting with a single tab character.

Construct 11.16.1: Makefile rules and commands.

```
target : prerequisite1 prerequisite2 ...
prerequisiteN
    command1
    command2
    ...
    commandN
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

A common error is to use spaces instead of a single tab character for specifying the commands. Another common error is including a tab character on an empty line, which will be reported by make as an error. As distinguishing between tabs and spaces in some text editors can be difficult, these errors are often challenging to find.

The following provides an example makefile for a program consisting of three files main.cpp, threeintsfcts.cpp, and threeintsfcts.h. The executable named myprog.exe is dependent on the object files main.o and threeintsfcts.o. The main.o object file is dependent on the main.cpp source file and the threeintsfcts.h header file. The threeintsfcts.o object file is dependent on the threeintsfcts.cpp and threeintsfcts.h files.

Figure 11.16.1: Makefile example for three integer functions program.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
myprog.exe : main.o threeintsfcts.o
    g++ main.o threeintsfcts.o -o myprog.exe

main.o : main.cpp threeintsfcts.h
    g++ -Wall -c main.cpp

threeintsfcts.o : threeintsfcts.cpp threeintsfcts.h
    g++ -Wall -c threeintsfcts.cpp

clean :
    rm *.o myprog.exe
```

Running make for first time:	`> make`
	`g++ -Wall -c main.cpp`
	`g++ -Wall -c threeintsfcts.cpp`
	`g++ main.o threeintsfcts.o -o myprog.exe`

Running make after modifying main.cpp:	`> make`
	`g++ -Wall -c main.cpp`
	`g++ main.o threeintsfcts.o -o myprog.exe`

When the make command is executed, if any of the prerequisites for the rule have been modified since the target was last created, the commands for the rule will be executed to create the target file. For example, in the above makefile, if main.cpp is modified, then make will first execute the command `g++ -Wall -c main.cpp` to create the main.o object file. The -c flag is used here to inform the compiler (e.g., g++) that the source file should only be compiled (and not linked) to create an object file. As main.o has now been modified, make will then execute the command `g++ main.o threeintsfcts.o -o myprog.exe`. The -o flag is used here to inform the linker (e.g., g++) to link the object files into the final executable using the name specified after the -o. In this case, the executable is named myprog.exe.

Make rules can also be used to define common operations used when managing larger programs. One common make rule is the `clean : rule` that is used to execute a command for deleting all generated files such as object files and the program executable. In the above example, this is accomplished by running the command `rm *.o myprog.exe`.

By default make assumes the makefile is named `makefile` or `Makefile`. The -f flag can be used to run make using a different filename. For example, `make -f MyMakefile` will run make using the file named `MyMakefile`.

PARTICIPATION ACTIVITY 11.16.1: Makefiles.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Answer the following using the Makefile provided above.

1) List the targets affected if main.cpp is changed. (Note: List the targets separated by spaces and in the order that their rules would execute.)

Check Show answer

2) What command will execute after threeintsfcts.o is changed? (Hint: The answer starts with g++.)

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

3) How many commands will execute when threeintsfcts.h is changed?

Check Show answer

Exploring further:

- Makefile tutorial from cprogramming.com
- Manual for make from gnu.org

11.17 Additional practice: Output art

This section has been set as optional by your instructor.

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following program prints a simple triangle.

zyDE 11.17.1: Create ASCII art.

Load default template...

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

int main() {

   cout << "  *  " << endl
   cout << " *** " << endl
   cout << "*****" << endl

   return 0;
}

```

Pre-enter any input for program, then press run.

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Create different versions of the program:

1. Print a tree by adding a base under a 4-level triangle:

```
   *
  ***
 *****
*******
  ***
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2. Print the following "cat":

```
^   ^
 o o
=   =
 ---
```

3. Allow a user to enter a number, and then print the original triangle using that number instead of asterisks, as in:

```
   9
  999
 99999
```

Pictures made from keyboard characters are known as ***ASCII art***. ASCII art can be quite intricate, and fun to make and view. ASCII Art Archive provides examples. Doing a web search for "ASCII art (some item)" can find ASCII art versions of an item. For example, searching for "ASCII art cat" turns up thousands of examples of cats, most much more clever than the cat above.

11.18 Additional practice: Grade calculation

This section has been set as optional by your instructor.

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

zyDE 11.18.1: Grade calculator.

The following incomplete program should compute a student's total course percentage based on scores on three items of different weights (%s):

- 20% Homeworks (out of 80 points)

- 30% Midterm exam (out of 40 points)
- 50% Final exam (out of 70 points)

Suggested (incremental) steps to finish the program:

1. First run it.
2. Next, complete the midterm exam calculation and run the program again. Use the constant variables where appropriate.
3. Then, complete the final exam calculation and run the program. Use the constant variables where appropriate.
4. Modify the program to include a quiz score out of 20 points. New weights: 10% homework, 15% quizzes, 30% midterm, 45% final. Run the program again.
5. To avoid having one large expression, introduce variables homeworkPart, quizPart, midtermPart, and finalPart. Compute each part first; each will be a number between 0 and 1. Then combine the parts using the weights into the course value. Run the program again.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Load default template...

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

int main() {
   const double HOMEWORK_MAX = 80.0;
   const double MIDTERM_MAX = 40.0;
   const double FINAL_MAX = 70.0;
   const double HOMEWORK_WEIGHT = 0.20; // 20%
   const double MIDTERM_WEIGHT = 0.30;
   const double FINAL_WEIGHT = 0.50;

   double homeworkScore;
   double midtermScore;
   double finalScore;
   double coursePercentage;
```

78 36 62

Run

11.19 Additional practice: Tweet decoder

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

This section has been set as optional by your instructor.

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

The following program decodes a few common abbreviations in online communication as communications in Twitter ("tweets") or email, and provides the corresponding English phrase.

zyDE 11.19.1: Tweet decoder.

Load default template...

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

int main() {
   string origTweet;

   cout << "Enter abbrevi
   cin >> origTweet;

   if (origTweet == "LOL"
      cout << "LOL = laug
   }
   else if (origTweet ==
      cout << "BFN = bye
```

LOL

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Create different versions of the program that:

1. Expands the number of abbreviations that can be decoded. Add support for abbreviations you commonly use or search the Internet to find a list of common abbreviations.
2. For abbreviations that do not match the supported abbreviations, check for common misspellings. Provide a suggestion for correct abbreviation along with the decoded meaning. For example, if the user enters "LLO", your program can output "Did you mean LOL? LOL = laughing out loud".
3. Allows the user to enter a complete tweet (140 characters or less) as a single line of text. Search the resulting string for those common abbreviations and print a list of each abbreviation along with its decoded meaning.
4. Convert the user's tweet to a decoded tweet, replacing the abbreviations directly within the tweet.

11.20 Additional practice: Dice statistics

This section has been set as optional by your instructor.

The following is a sample programming lab activity; not all classes using a zyBook require students to fully complete this activity. No auto-checking is performed. Users planning to fully complete this program may consider first developing their code in a separate programming environment.

Analyzing dice rolls is a common example in understanding probability and statistics. The following calculates the number of times the sum of two dice (randomly rolled) equals six or seven.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

zyDE 11.20.1: Dice rolls: Counting number of rolls that equals six or seven.

Load default template...

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;

int main(){
   int i;           // Loop counter iterates numRolls t
   int numRolls;    // User defined number of rolls
   int numSixes;    // Tracks number of 6s found
   int numSevens;   // Tracks number of 7s found
   int die1;        // Dice values
   int die2;        // Dice values
   int rollTotal;   // Sum of dice values

   numSixes = 0;
   numSevens = 0;
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

10

Run

Create different versions of the program that:

1. Calculates the number of times the sum of the randomly rolled dice equals each possible value from 2 to 12.
2. Repeatedly asks the user for the number of times to roll the dice, quitting only when the user-entered number is less than 1. Hint: Use a while loop that will execute as long as numRolls is greater than 1. Be sure to initialize numRolls correctly.
3. Prints a histogram in which the total number of times the dice rolls equals each possible value is displayed by printing a character like * that number of times, as shown below.

Figure 11.20.1: Histogram showing total number of dice rolls for each possible value.

```
Dice roll histogram:

2:  ******
3:  ****
4:  ***
5:  ********
6:
*******************
7:  *************
8:  *************
9:  **************
10: ***********
11: *****
12: ****
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

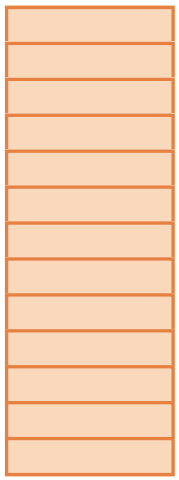
11.21 How functions work

This section has been set as optional by your instructor.

Each function call creates a new set of local variables, forming part of what is known as a ***stack frame***. A return causes those local variables to be discarded.

PARTICIPATION ACTIVITY 11.21.1: Function calls and returns.

```
int FtInToIn(int inFeet, int inInches) {
   int totInches;
   ...
   return totInches;
}

double FtInToCm(int inFeet, int inInches) {
   int totIn;
   double totCm;
   ...
   totIn = FtInToIn(inFeet, inInches);
   ...
   return totCm;
}

int main() {
   int userFt;
   int userIn;
   int userCm;
   ...
   userCm = FtInToCm(userFt, userIn);
   ...
   return 0;
}
```



©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Animation content:

Static Figure:
Being C++ code:

```
int FtInToIn(int inFeet, int inInches) {
  int totInches;
  ...
  return totInches;
}

double FtInToCm(int inFeet, int inInches) {
  int totIn;
  double totCm;
  ...
  totIn = FtInToIn(inFeet, inInches);
  ...
  return totCm;
}

int main() {
  int userFt;
  int userIn;
  int userCm;
  ...
  userCm = FtInToCm(userFt, userIn);
  ...
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
  return 0;
}
```

End C++ code.
A memory space containing 12 empty cells is shown.

Step 1: Each function call creates a new set of local variables.
The C++ code is highlighted into three separate section. The memory space is also separated into three separate sections, indicating the spaces needed for the local variable within the C++ code.
The first section highlighted is the main function and is highlighted in purple:

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
int main() {
  int userFt;
  int userIn;
  int userCm;
  ...
  userCm = FtInToCm(userFt, userIn);
  ...
  return 0;
}
```

The first three memory cells are highlighted in purple showing that local variables, “userFt”, “userIn”, and “userCm” are set to these three cells.
The second section highlighted is the FtInToCm() function and is highlighted in blue:

```
double FtInToCm(int inFeet, int inInches) {
  int totIn;
  double totCm;
  ...
  totIn = FtInToIn(inFeet, inInches);
  ...
  return totCm;
}
```

The fifth through the eighth memory cells are highlighted in blue showing that local variables, “InFeet”, “inInches”, “totIn”, and “totCm” are set to these four cells.
The third section highlighted is the FtInToIn() function and is highlighted in green:

```
int FtInToIn(int inFeet, int inInches) {
  int totInches;
  ...
  return totInches;
}
```

The eleventh, twelfth, and thirteenth memory cells are highlighted in green showing that local variables, “inFeet”, “inInches”, and “totInches” are set to these three cells.

Step 2: Each return causes those local variables to be discarded.
Once the functions execute the return line of code, the memory space is freed, removing the local variables.

Animation captions:

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1. Each function call creates a new set of local variables.
2. Each return causes those local variables to be discarded.

Some knowledge of how a function call and return works at the assembly level can not only satisfy curiosity, but can also lead to fewer mistakes when parameter and return items become more complex. The following animation illustrates by showing, for a function named CalcSum(), some sample high-level code, compiler-generated assembly instructions in

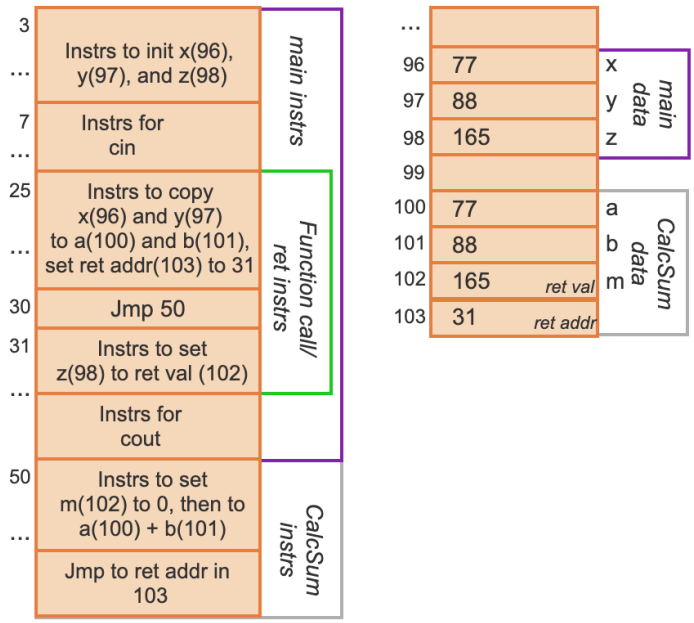
memory, and data in memory during runtime. This animation presents advanced material intended to provide insight and appreciation for how a function call and return works.

The compiler generates instructions to copy arguments to parameter local variables, and to store a return address. A jump instruction jumps from main to the function's instructions. The function executes and stores results in a designated return value location. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

PARTICIPATION ACTIVITY 11.21.2: How function call/return works.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int CalcSum(int a, int b) {
  int m;

  m = 0;
  m = a + b;
  return m;
}

int main() {
  int x;
  int y;
  int z;

  x = 0;
  y = 0;
  z = 0;
  cin >> x;
  cin >> y;
  z = CalcSum(x,y);

  cout << z << "\n";
  return 0;
}
```



Animation content:

Static Figure:
Being C++ code:
#include <iostream>
using namespace std;

int CalcSum(int a, int b) {
  int m;

  m = 0;
  m = a + b;
  return m;
}

int main() {
  int x;
  int y;
  int z;

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
  x = 0;
  y = 0;
  z = 0;
  cin >> x;
  cin >> y;
  z = CalcSum(x,y);

  cout << z << "\n";
  return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

End C++ code.

Two memory spaces are shown. The first memory space contains 8 memory cells/addresses. The first through sixth memory cells are designated for main assembly instructions and is highlighted in purple. The third through fifth cells are designated for Function call/ ret assembly instructions and is highlighted in green. The seventh and eight cells are designated for CalcSum assembly instructions and are highlighted in grey. The first cell contains "Instrs to init x(96), y(97), and z(98)". The second cell contains "Instrs for cin". The third cell contains, "Instrs to copy x(96) and y(97) to a(100) and b(101), set ret addr(103) to 31". The fourth memory space contains "Jmp 50". The fifth cell contains "Instrs to set z(98) to ret val (102)". The sixth cell contains "Instrs for cout". The seventh cell contains "Instrs to set m(102) to 0, then to a(100) + b(101)". The eighth cell contains "Jmp to ret addr in 103".

The second memory space contains 9 memory cells/addresses. The second through fourth cells are designated for main data and is highlighted in purple. The sixth through ninth cells are designated for CalcSum data and is highlighted in grey. The first and fifth cells are empty. The second cell contains the value of x, 77. The third cell contains the value of y, 88. The fourth cell contains the value of z, 165. The sixth cell contains the value of a, 77. The seventh cell contains the value of b, 88. The eighth cell contains the value of m, 165, which is the return value. The nineth cell contains the return address value, 31.

Step 1: The compiler converts high-level code into assembly instructions (instrs) in memory.

The C++ code is processed through a compiler and is converted into assembly code in memory. The memory space contains 8 memory cells/addresses. The first through sixth memory cells are designated for main assembly instructions and is highlighted in purple. The third through fifth cells are designated for Function call/ ret assembly instructions and is highlighted in green. The seventh and eight cells are designated for CalcSum assembly instructions and is highlighted in grey. The first cell contains "Instrs to init x(96), y(97), and z(98)". The second cell contains "Instrs for cin". The third cell contains, "Instrs to copy x(96) and y(97) to a(100) and b(101), set ret addr(103) to 31". The fourth memory space contains "Jmp 50". The fifth cell contains "Instrs to set z(98) to ret val (102)". The sixth cell contains "Instrs for cout". The seventh cell contains "Instrs to set m(102) to 0, then to a(100) + b(101)". The eighth cell contains "Jmp to ret addr in 103".

Step 2: Before executing the function, arguments are copied to parameter local variables and a return address is stored.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The main function executes adding the values of the local variables to memory. The C++ main function code lines, "int x; ", "int y; ", and "int z;" execute, designating 3 memory cells/addresses in memory for local variables x, y, and z. The code line, "x = 0; ", "y = 0; ", and "z = 0;" execute, adding 0 to each local memory address x, y, and z. The code lines "cin >> x;" and "cin >> y;", assigning value 77 to memory address for variable "x" and value 88 to memory address for variable "y". Next, code line, "z = CalcSum(x,y);" designating 3 memory cells/addresses in memory for local variables "a", "b", and the return address for function CalcSum. The memory address for "a" contains the value 77, the memory address for "b" contains the value 88, and the memory address for the return address contains the value 31.

Step 3: The function executes and stores the result in a designated return value location.
The CalcSum function executes adding the values of the local variables to memory. The lines of code "int m;", "m = 0;", "m = a + b;", and "return m;" executes, adding the value 165 to the memory address assigned to local variable m.

Step 4: When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.
The execution flow exits the CalcSum function and returns to the line "z = CalcSum(x,y);" in the main functions. The memory address for "z" now contains the value 165.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Animation captions:

1. The compiler converts high-level code into assembly instructions (instrs) in memory.
2. Before executing the function, arguments are copied to parameter local variables and a return address is stored.
3. The function executes and stores the result in a designated return value location.
4. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

PARTICIPATION ACTIVITY 11.21.3: How functions work.

1) After a function returns, its local variables keep their values, which serve as their initial values the next time the function is called.

- True
- False

2) A return address indicates the value returned by the function.

- True
- False

11.22 Using pass by reference to modify string/vector parameters

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

This section has been set as optional by your instructor.

Functions commonly modify a string or vector. The following function modifies a string by replacing spaces with hyphens.

Figure 11.22.1: Modifying a string parameter, which should be pass by reference.

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(string& modStr) {
   unsigned int i;   // Loop index

   for (i = 0; i < modStr.size(); ++i) {
      if (modStr.at(i) == ' ') {
         modStr.at(i) = '-';
      }
   }
}

int main() {
   string userStr;  // Input string from
user

   // Prompt user for input
   cout << "Enter string with spaces: " <<
endl;
   getline(cin, userStr);

   // Call function to modify user defined
string
   StrSpaceToHyphen(userStr);

   // Output modified string
   cout << "String with hyphens: ";
   cout << userStr << endl;

   return 0;
}
```

```
Enter string with spaces:
Hello there everyone.
String with hyphens: Hello-there-
everyone.

...

Enter string with spaces:
Good bye  now   !!!
String with hyphens: Good-bye--now-
--!!!
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The string serves as function input and output. The string parameter must be pass by reference, achieved using &, so that the function modifies the original string argument (userStr) and not a copy.

zyDE 11.22.1: Modifying a string parameter: Spaces to hyphens.

1. Run the program, noting correct output.
2. Remove the & and run again, noting the string is not modified, because the string is pass by value and thus the function modifies a copy. When done replace the &
3. Modify the function to also replace each '!' by a '?'.

Load default template...

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

// Function replaces spac
void StrSpaceToHyphen(stri
   unsigned int i;  // Lo

   for (i = 0; i < modStr
      if (modStr.at(i) ==
         modStr.at(i) = '-
      }
```

Hello there everyone!!!

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

```
    }
}

```

Sometimes a programmer defines a vector or string parameter as pass by reference even though the function does not modify the parameter, to prevent the performance and memory overhead of copying the argument that would otherwise occur.

The keyword ***const*** can be prepended to a function's vector or string parameter to prevent the function from modifying the parameter. Programmers commonly make a large vector or string input parameter pass by reference, to gain efficiency, while also making the parameter const, to prevent assignment.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The following illustrates. The first function modifies the vector so it defines a normal pass by reference. The second function does *not* modify the vector but for efficiency uses constant pass by reference.

Figure 11.22.2: Normal and constant pass by reference vector parameters in a vector reversal program.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

void ReverseVals(vector<int>& vctrVals) {
   unsigned int i;  // Loop index
   int tmpVal;      // Temp variable for swapping

   for (i = 0; i < (vctrVals.size() / 2); ++i) {
      tmpVal = vctrVals.at(i); // These statements
swap
      vctrVals.at(i) = vctrVals.at(vctrVals.size()
- 1 - i);
      vctrVals.at(vctrVals.size() - 1 - i) =
tmpVal;
   }
}

void PrintVals(const vector<int>& vctrVals) {
   unsigned int i;  // Loop index

   // Print updated vector
   cout << endl << "New values: ";
   for (i = 0; i < vctrVals.size(); ++i) {
      cout << " " << vctrVals.at(i);
   }
   cout << endl;
}

int main() {
   const int NUM_VALUES = 8;             // Vector
size
   vector<int> userValues(NUM_VALUES);  // User
values
   int i;                               // Loop
index

   // Prompt user to populate vector
   cout << "Enter " << NUM_VALUES << " values..."
<< endl;
   for (i = 0; i < NUM_VALUES; ++i) {
      cout << "Value: ";
      cin >> userValues.at(i);
   }

   // Call function to reverse vector values
   ReverseVals(userValues);

   // Print reversed values
   PrintVals(userValues);

   return 0;
}
```

```
Enter 8 values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  80 70 60 50 40
30 20 10
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

A reader might wonder why all input parameters are not defined as constant pass by reference parameters: Why make local copies at all? The reason is efficiency. For parameters involving just a few memory locations, making a local copy enables the compiler to generate more efficient code, in part because the compiler can place those copies inside a tiny-but-fast memory inside the processor called a register file—further details are beyond our scope.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

In summary:

- Define a function's output or input/output parameters as pass by reference.
  - But create output parameters sparingly, striving to use return values instead.
- Define input parameters as pass by value.
  - Except for large items (perhaps 10 or more elements); use constant pass by reference for those.

PARTICIPATION ACTIVITY 11.22.1: Constants and pass by reference.

How should a function's vector parameter `ages` be defined for the following situations?

1) ages will always be small (fewer than 10 elements) and the function will not modify the vector.

- Constant and pass by reference.
- Constant but not pass by reference.
- Pass by reference but not constant.
- Neither constant nor pass by reference.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) ages will always be small, and the function will modify the vector.

- Constant and pass by reference.
- Constant but not pass by reference.
- Pass by reference but not constant.
- Neither constant nor pass by reference.

3) ages may be very large, and the function will modify the vector.

- Constant and pass by reference.
- Constant but not pass by reference.
- Pass by reference but not constant.
- Neither constant nor pass by reference.

4) ages may be very large, and the function will not modify the vector.

- Constant and pass by reference.
- Constant but not pass by reference.
- Pass by reference but not constant.
- Neither constant nor pass by reference.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.22.2: Vector parameters.

Define a function's vector parameter `ages` for the following situations. Assume ages is a vector of integers. Example: ages will always be small (fewer than 10 elements) and the function will not modify the vector: `const vector<int> ages`.

1) ages will always be small, and the function will modify the vector.

```
void MyFct (
                    ) {
```

Check Show answer

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

2) ages may be very large, and the function will modify the vector

```
void MyFct (
                    ) {
```

Check Show answer

3) ages may be very large, and the function will not modify the vector.

```
void MyFct (
                    ) {
```

Check Show answer

CHALLENGE ACTIVITY 11.22.1: Use an existing function.

Use function GetUserInfo to get a user's information. If user enters 20 and Holly, sample program output is:

```
Holly is 20 years old.
```

Learn how our autograder works

602398.4584902.qx3zqy7

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

void GetUserInfo(int& userAge, string& userName) {
   cout << "Enter your age: " << endl;
   cin >> userAge;
   cout << "Enter your name: " << endl;
   cin >> userName;
}

int main() {
   int userAge;
   string userName;

```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Run

CHALLENGE ACTIVITY 11.22.2: Modify a string parameter.

Complete the function to replace any period with an exclamation point. Ex: "Hello. I'm Miley. Nice to meet you." becomes:

```
"Hello! I'm Miley! Nice to meet you!"
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Learn how our autograder works

602398.4584902.qx3zqy7

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

void MakeSentenceExcited(string& sentenceText) {

   /* Your solution goes here  */

}

int main() {
   string testStr;

   getline(cin, testStr);
   MakeSentenceExcited(testStr);
```

Run

CHALLENGE ACTIVITY 11.22.3: Modify a vector parameter. Full screen

602398.4584902.qx3zqy7

Organize the lines of code to define a function SwapVectorEnds() that swaps the first and last elements of the function's vec parameter.

Ex: If the input is `10 20 30 40`, then the output is:

```
40 20 30 10
```

Note: This activity includes distractors. Not all lines of code on the left will be used in the final solution.

How to use this tool

Unused

```
inputVector.at(inputVector.size() - 1) = tempValue;
}
tempValue = inputVector.at(0);
int tempValue;
inputVector.at(0) = inputVector.at(inputVector.size() -
void SwapVectorEnds(vector<int> &inputVector) {
```

main.cpp Load default templ

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

int main() {
   vector<int> sortVector(4);
   unsigned int i;
   int userNum;
```

```
void SwapVectorEnds(vector<int> inputVector) {
```

```
for (i = 0; i < sortVector.size(); ++i) {
   cin >> userNum;
   sortVector.at(i) = userNum;
}

SwapVectorEnds(sortVector);

for (i = 0; i < sortVector.size(); ++i) {
   cout << sortVector.at(i) << " ";
```

Check

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.22.4: Functions with string parameters.

602398.4584902.qx3zqy7

Start

Complete the function FindAll() that has one string parameter and one character parameter. The function returns true if all th characters in the string are not equal to the character parameter. Otherwise, the function returns false.

Ex: If the input is `tmkv v`, then the output is:

```
False, at least one character is equal to v.
```

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

bool FindAll(string inputString, char x) {

   /* Your code goes here */

}

int main() {
   string inputString;
   char x;
   bool result;

   cin >> inputString;
```

1 2 3

Check Next level

CHALLENGE ACTIVITY 11.22.5: Functions with vector parameters.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

602398.4584902.qx3zqy7

Start

Define a function CheckSum() that takes one integer vector parameter and one integer parameter. The function computes th sum of the vector's elements. Then, the function returns true if the sum is equal to the integer parameter, and returns false otherwise.

Ex: If the input is:

```
4
2 -9 -5 0
2
```

then the vector has 4 elements {2, -9, -5, 0}, and the integer parameter is 2. The output is:

```
False, the sum of all the elements is not equal to 2.
```

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

/* Your code goes here */

int main() {
   vector<int> inVector;
   int size;
   int input;
   int i;
   int x;
   bool result;

   // Read the vector's size, and then the vector's elements
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1 2

Check Next level

11.23 Default parameter values

This section has been set as optional by your instructor.

Sometimes a function's last parameter (or last few) should be optional. A function call could then omit the last argument, and instead the program would use a default value for that parameter. A function can have a ***default parameter value*** for the last parameter(s), meaning a call can optionally omit a corresponding argument.

Figure 11.23.1: Parameter with a default value.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

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

// Function prints date in two styles (0: American (default), 1:
European)
void PrintDate(int currDay, int currMonth, int currYear, int
printStyle = 0) {

   if (printStyle == 0) {       // American
      cout << currMonth << "/" << currDay << "/" << currYear;
   }
   else if (printStyle == 1) { // European
      cout << currDay << "/" << currMonth << "/" << currYear;
   }
   else {
      cout << "(invalid style)";
   }
}

int main() {

   // Print dates given various style settings
   PrintDate(30, 7, 2012, 0);
   cout << endl;

   PrintDate(30, 7, 2012, 1);
   cout << endl;

   PrintDate(30, 7, 2012); // Uses default value for printStyle
   cout << endl;

   return 0;
}
```

```
7/30/2012
30/7/2012
7/30/2012
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The fourth (and last) parameter has a default value: `int printStyle = 0`. If a function call does not provide a fourth argument, then the style parameter is 0.

The same can be done for other parameters, as in:
`void PrintDate(int currDay = 1, int currMonth = 1, int currYear = 2000, int printStyle = 0)`
Because arguments are matched with parameters based on their ordering in the function call, only the last arguments can be omitted. The following are valid calls to this PrintDate() function having default values for all parameters:

Figure 11.23.2: Valid function calls with default parameter values.

```
PrintDate(30, 7, 2012, 0); // No defaults
PrintDate(30, 7, 2012);    // Defaults:                          style=0
PrintDate(30, 7);          // Defaults:                year=2000, style=0
PrintDate(30);             // Defaults:        month=1, year=2000, style=0
(strange, but valid)
PrintDate();               // Defaults: day=1, month=1, year=2000, style=0
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

If a parameter does not have a default value, then failing to provide an argument generates a compiler error. Ex: Given: void PrintDate(int currDay, int currMonth, int currYear, int printStyle = 0). Then the call PrintDate(30, 7) generates the following error message from g++.

Figure 11.23.3: Compiler error if parameters corresponding to omitted arguments don't have default values.

```
fct_defparm.cpp: In function int main():
fct_defparm.cpp:5: error: too few arguments to function void PrintDate(int, int, int,
int)
fct_defparm.cpp:22: error: at this point in file
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

PARTICIPATION ACTIVITY 11.23.1: Function parameter defaults.

Given:

```
void CalcStat(int num1, int num2, int num3 = 0, char usrMethod = 'a') { ...
}
```

1) A compiler error will occur because only an int parameter can have a default value.

- True
- False

2) The call CalcStat(44, 47, 42, 'b') uses usrMethod = 'a' because the parameter default value of 'a' overrides the argument 'b'.

- True
- False

3) The call CalcStat(44, 47, 42) uses usrMethod = 'a'.

- True
- False

4) The call CalcStat(44, 47, 'b') uses num3 = 0.

- True
- False

5) The following is a valid start of a function definition: `void myFct(int num1 = 0, int num2 = 0, char usrMethod) {`

- True
- False

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Exploring further:

- Default arguments from msdn.microsoft.com

CHALLENGE ACTIVITY 11.23.1: Functions with default parameters.

602398.4584902.qx3zqy7

Start

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Type the program's output

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

void PrintNums(int a, int b, int c = 8) {
   cout << a << ", " << b << ", " << c << endl;
}

int main() {
   PrintNums(2, 5);
   PrintNums(6, 3, 1);

   return 0;
}
```

```
2, 5, 8
6, 3, 1
```

1 2

Check Next

CHALLENGE ACTIVITY 11.23.2: Return number of pennies in total. Full screen

602398.4584902.qx3zqy7

Integer numDollars is read from input. Organize the lines of code to define a function NumberOfPennies() that returns the to number of pennies given the number of dollars and (optionally) a number of pennies.

Ex: If the input is `4`, then the output is:

```
Balance: 400 pennies
```

```
Balance with quarter: 425 pennies
```

Note: This activity includes distractors. Not all lines of code on the left will be used in the final solution.

How to use this tool

Unused

```
return (dollars * 10) + pennies;
return (dollars * 100) + pennies;
return (dollars * 100) + pennies
}
int NumberOfPennies(int dollars, int pennies) {
int NumberOfPennies(int dollars, int pennies = 0) {
int NumberOfPennies(int dollars = 0, int pennies) {
```

main.cpp Load default templ

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

int main() {
   int numDollars;

   cin >> numDollars;

   cout << "Balance: " << NumberOfPennies(numDollar
   cout << "Balance with quarter: " << NumberOfPenn

   return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Check

11.24 Function name overloading

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

This section has been set as optional by your instructor.

Sometimes a program has two functions with the same name but differing in the number or types of parameters, known as ***function name overloading*** or just ***function overloading***. The following two functions print a date given the day, month, and year. The first function has parameters of type int, int, and int, while the second has parameters of type int, string, and int.

Figure 11.24.1: Overloaded function name.

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

void PrintDate(int currDay, int currMonth, int currYear)
{
   cout << currMonth << "/" << currDay << "/" <<
currYear;
}

void PrintDate(int currDay, string currMonth, int
currYear) {
   cout << currMonth << " " << currDay << ", " <<
currYear;
}

int main() {

   PrintDate(30, 7, 2012);
   cout << endl;

   PrintDate(30, "July", 2012);
   cout << endl;

   return 0;
}
```

```
7/30/2012
July 30,
2012
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

The compiler determines which function to call based on the argument types. PrintDate(30, 7, 2012) has argument types int, int, int, so calls the first function. PrintDate(30, "July", 2012) has argument types int, string, int, so calls the second function.

More than two same-named functions is allowed as long as each has distinct parameter types. Thus, in the above program:

- PrintDate(int month, int day, int year, int style) can be added because the types int, int, int, int differ from int, int, int, and from int, string, int.
- PrintDate(int month, int day, int year) yields a compiler error, because two functions have types int, int, int (the parameter names are irrelevant).

A function's return type does not influence overloading. Thus, having two same-named function definitions with the same parameter types but different return types still yield a compiler error.

The use of overloading and of default parameter values may be combined as long as no ambiguity is introduced. Adding the function `void PrintDate(int month, int day, int year, int style = 0)` above would generate a compiler error because the compiler cannot determine if the function call `PrintDate(7, 30, 2012)` should go to the "int, int, int" function or to that new "int, int, int, int" function with a default value for the last parameter.

PARTICIPATION ACTIVITY 11.24.1: Function name overloading.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Given the following function definitions, select the number that each function call would print.
If the function call would not compile, choose Error.

```
void PrintDate(int day, int month, int year) {
   cout << "1" << endl;
}

void PrintDate(int day, string month, int year) {
   cout << "2" << endl;
}

void PrintDate(int month, int day) {
   cout << "3" << endl;
}
```

1) PrintDate(30, 7, 2012);

- 1
- 2
- 3
- Error

2) PrintDate(30, "July", 2012);

- 1
- 2
- 3
- Error

3) PrintDate(7, 2012);

- 1
- 2
- 3
- Error

4) PrintDate(30, 7);

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

- 1
- 2
- 3
- Error

5) PrintDate("July", 2012);

- 1
- 2

- 3
- Error

Exploring further:

- Overloaded functions from cplusplus.com.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

CHALLENGE ACTIVITY 11.24.1: Overload salutation printing.

Complete the second PrintSalutation function to print the following given personName "Holly" and customSalutation "Welcome":

```
Welcome, Holly
```

End with a newline.

Learn how our autograder works

602398.4584902.qx3zqy7

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

void PrintSalutation(string personName) {
   cout << "Hello, " << personName << endl;
}

// Define void  PrintSalutation(string personName, string customSalutation)..

/* Your solution goes here  */

int main() {
   PrintSalutation("Holly", "Welcome");
   PrintSalutation("Sanjiv");
```

Run

CHALLENGE ACTIVITY 11.24.2: Convert a height into inches. Full screen

602398.4584902.qx3zqy7

Double numFeet is read from input. Organize the lines of code to define a second ConvertToInches() with two double param numFeet and numInches, that returns the total number of inches.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Ex: If the input is `4.0`, then the output is:

```
4.0 feet is 48.0 inches

4.0 feet plus 6 inches is 54.0 inches
```

Note: This activity includes distractors. Not all lines of code on the left will be used in the final solution.

How to use this tool

Unused

```
totalInches = (numFeet * 12.0) * numInches;
totalInches = (numFeet * 12.0) + numInches;
return totalInches;
double ConvertToInches(double numFeet, double numInches)
double ConvertToInches(double numFeet, numInches) {
}
double totalInches;
int totalInches;
```

main.cpp Load default templ

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

double ConvertToInches(double numFeet) {
   return numFeet * 12.0;
}

int main() {
   double numFeet;
   double totalInches;

   cin >> numFeet;

   totalInches = ConvertToInches(numFeet);
   cout << fixed << setprecision(1) << numFeet << "

   totalInches = ConvertToInches(numFeet, 6);
   cout << fixed << setprecision(1) << numFeet << "
```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Check

CHALLENGE ACTIVITY 11.24.3: Function name overloading.

602398.4584902.qx3zqy7

Start

The function PrintWardrobeHeight() has a double parameter. Define a second PrintWardrobeHeight() function that has an int parameter. The second function outputs the following in order, all on one line:

- "Wardrobe height as a whole number: "
- the value of the integer parameter
- " inches"

End with a newline.

Ex: If the input is `41.25 41`, then the output is:

```
Wardrobe height to two decimal places: 41.25 inches
Wardrobe height as a whole number: 41 inches
```

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

void PrintWardrobeHeight(double wardrobeHeight) {
   cout << fixed << setprecision(2) << "Wardrobe height to two decimal places: ";
   cout << wardrobeHeight << " inches" << endl;
}

/* Your code goes here */

int main() {
   double wardrobeHeight1;
   int wardrobeHeight2;

```

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

1 2 3

Check Next level

11.25 zyLab training: Interleaved input / output

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

This section has been set as optional by your instructor.

LAB ACTIVITY | 11.25.1: zyLab training: Interleaved input / output 0 / 2

Auto-graded programming assignments have numerous advantages, but have some challenges too. Students commonly struggle with realizing that example input / output provided in an assignment's specification interleaves input and output, but the program *should only output the output parts*. If a program should double its input, an instructor might provide this example:

```
Enter x:
5
x doubled is: 10
```

Students often incorrectly create a program that outputs the 5. Instead, the program should only output the output parts:

```
Enter x:
x doubled is: 10
```

The instructor's example is showing both the output of the program, AND the user's input to that program, assuming the program is developed in an environment where a user is interacting with a program. But the program itself doesn't output the 5 (or the newline following the 5, which occurs when the user types 5 and presses enter).

Also, if the instructor configured the test cases to observe whitespace, then according to the above example, the program should output a newline after `Enter x:` (and possibly after the 10, if the instructor's test case expects that).

The program below *incorrectly* echoes the user's input to the output.

1. Try submitting it for grading (click "Submit mode", then "Submit for grading"). Notice that the test cases fail. The first test case's highlighting indicates that output 3 and newline were not expected. In the second test case, the -5 and newline were not expected.
2. Remove the code that echoes the user's input back to the output, and submit again. Now the test cases should all pass.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

Run History Tutorial

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

int main() {
   int x;

   cout << "Enter x: " << endl;
   cin  >> x;

   cout << x << endl;  // Student mistakenly is echoing the input to output to
   cout << "x doubled is: " << 2 * x << endl;

   return 0;
}
```

©zyBooks 09/29/24 11:48 2292451
Shivas Pandit
TXSTATECS1428Vargas-GarzonFall2024

DESKTOP CONSOLE

Submit for grading

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

Latest submission

No submissions yet

©zyBooks 09/29/24 11:48 2292451
Shivas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.26 zyBooks built-in programming window

This section has been set as optional by your instructor.

zyDE 11.26.1: Programming window.

Load default template...

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

int main() {
   // Enter your program h
   return 0;
}
```

Pre-enter any input for program, then press run.

Run

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024

11.27 zyLab training*: One large program

This section's content is not available for print.

©zyBooks 09/29/24 11:48 2292451
Diwas Pandit
TXSTATECS1428Vargas-GarzonFall2024