UPE Links

UPE Tutoring:

CS 31 Midterm 1 Review

Sign-in https://tinyurl.com/cs31m1fall21form

Slides link available upon sign-in

Table of Contents

Libraries

- `#include` allows us to use a library
- `#include <iostream>` allows us to use things like:
  - `cin`
  - `cout`
  - `endl`
- *Note: `iostream` stands for input/output stream*

Namespaces

- `using namespace std;`
- A namespace is a collection of classes and functions
- If we don't call `using namespace` *`ns_name`*, we will have to specify the namespace of the function we want to call.
- e.g. `std::cout`, `std::string`, `std::isdigit`

Namespaces (cont.)

```
#include <iostream>

int main() {
  int age;
  std::cin >> age;
  std::cout << age;
  std::cout << std::endl;
}
```

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

int main() {
  int age;
  cin >> age;
  cout << age;
  cout << endl;
}
```

Basic data types

- `int, double, char`
  - Declare variables to store values in memory
  - `int x; // Creates a variable x of type int`
  - `char y; // Creates a variable y of type char`
- Can initialize with value at declaration:
  - `int a = 5;`
  - `double z = 53.24324;`

Modifying variables

- The type of the variable must be specified only once, at the time of declaration

```
int x = 5;
x = x + 5;
x -= 6; // equivalent to x = x - 6;

double z;
z = 53.234;
z *= 5; // equivalent to z = z * 5;
```

Modifying variables (cont.)

- Integer division truncates after the decimal point
- The % (modulus) operator returns the remainder of integer division

```
int x = 5;
int integerQuotient = x / 3; // integerQuotient equals 1
int remainder = x % 3;       // remainder equals 2
x %= 4;                      // same as x = x % 4, x now equals 1
```

Modifying variables (cont.)

- Double division
  - If <u>at least one</u> of the operands is a double, floating point division occurs.
  - If both values are integers, integer division occurs instead.

```
int x = 5;
double unexpectedQuotient = x / 2;    // equals 2.0
double expectedQuotient = x / 2.0;    // equals 2.5
```

Input/output

```
#include <iostream>
using namespace std;
int main() {
  int age;
  cout << "How old are you? " << endl;
  cin >> age;
  cout << "You are " << age <<
    " years old" << endl;
}
```

Input/output

```
#include <iostream>
using namespace std;
int main() {
  int age;
  cout << "How old are you? " << endl;
  cin >> age;
  cout << "You are " << age <<
    " years old" << endl;
}
```

Input/output

```
#include <iostream>
using namespace std;
int main() {
  int age;
  cout << "How old are you? " << endl;
  cin >> age;
  cout << "You are " << age <<
    " years old" << endl;
}
```

```
> How old are you?
```

age

Input/output

```
#include <iostream>
using namespace std;
int main() {
  int age;
  cout << "How old are you? " << endl;
  cin >> age;
  cout << "You are " << age <<
    " years old" << endl;
}
```

```
> How old are you? 20
```

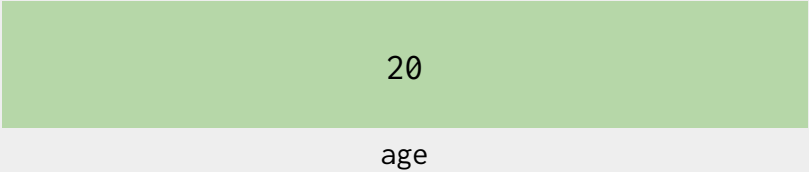
20

age

Input/output

```
#include <iostream>
using namespace std;
int main() {
  int age;
  cout << "How old are you? " << endl;
  cin >> age;
  cout << "You are " << age <<
    " years old" << endl;
}
```

```
> How old are you? 20
> You are 20 years old
```



Strings

- Used to store text
- Strings can be initialized
  - `string x = “hello”;`
- Individual characters can be accessed with `[ ]` or the .at(k) function
  - `char c = x[0]; // c == ‘h’`
  - `char ch = x.at(0); // ch == ‘h’`

String operations

```
string x = "hello there";
```

- The `size()` method returns the number of characters in a string.
  - `int length = x.size(); // length equals 11`
- The `substr(startIndex, length)` method returns a substring *including* `startIndex` of length `length`.
  - `string sub = x.substr(3, 2); // sub equals "lo"`
- *Note: `substr` is not in the scope of the midterm.*

String operations

```
// The + operator is overloaded:
// It appends to the end of strings.

int main() {
  string x = "hello there";
  x += ", my name is Mark";
  cout << x << endl;
}
```

hello there

x

String operations

```
// The + operator is overloaded:
// It appends to the end of strings.

int main() {
  string x = "hello there";
  x += ", my name is Mark";
  cout << x << endl;
}
```

hello there, my name is Mark

x

String operations

```
// The + operator is overloaded:
// It appends to the end of strings.

int main() {
  string x = "hello there";
  x += ", my name is Mark";
  cout << x << endl;
}
```

```
> hello there, my name is Mark
```

hello there, my name is Mark

x

String input

```
// getline(...) consumes characters from
// the input until it encounters a '\n'.

int main() {
  string x;
  getline(cin, x);
  cout << x << endl;
}
```

String input

```
// getline(...) consumes characters from
// the input until it encounters a '\n'.

int main() {
  string x;
  getline(cin, x);
  cout << x << endl;
}
```

```
> Why hello there!
```

String input

```
// getline(...) consumes characters from
// the input until it encounters a '\n'.

int main() {
  string x;
  getline(cin, x);
  cout << x << endl;
}
```

```
> Why hello there!
> Why hello there!
```

Ignoring characters

- Undesirable characters are often left in the input buffer after using cin.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.

Ignoring characters

- Undesirable characters are often left in the input buffer after using cin.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.

- **Common question**: does `getline` consume `'\n'`?

Ignoring characters

- Undesirable characters are often left in the input buffer after using cin.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.

- **Common question**: does `getline` consume `'\n'`?

  If a newline is found, it is extracted and discarded (i.e. it is not stored and the next input operation will begin after it).

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like?
```

Input

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like?
```

bigMacs

Input

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like? 1000
```

1000	\n
bigMacs	Input

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like? 1000
```

bigMacs	Input
1000	

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like? 1000
> What else would you like with your
  order?
```

bigMacs	Input
1000	

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like? 1000
> What else would you like with your
  order?
```

bigMacs	sides	Input
1000		

cin.ignore example

```
int main() {
  cout << "How many Big Macs would you " <<
      "like? ";
  int bigMacs;
  cin >> bigMacs;
  cin.ignore(10000, '\n'); // Important!

  cout << "What else would you like " <<
    "with your order?";
  string sides;
  getline(cin, sides);
}
```

```
> How many Big Macs would you like? 1000
> What else would you like with your
  order? Fries
```

1000	Fries	
bigMacs	sides	Input

cin.ignore(...) example

- What will be stored in the string “a” in this example?
- Assume that input is newline terminated.

```
int x; string a;
cout << "Enter an integer" << endl;
cin >> x; // Assume the user enters “7”
cout << "Enter a string" << endl;
getline(cin, a); // Assume user enters “500”
```

cctype

- Useful shortcut methods for characters
- #include<cctype> gives you...
  - `isalpha('M')  // true, since 'M' is a letter`
  - `isupper('M')  // true, since 'M' is an uppercase letter`
  - `islower('r')  // true, since 'r' is a lowercase letter`
  - `isdigit('5')  // true, since '5' is a digit character`
  - `islower('M')  // false, since 'M' is not a lowercase letter`
  - `isalpha(' ')  // false, since ' ' is not a letter`
  - `isalpha('5')  // false, since '5' is not a letter`

If statements

- `if` statements only run code if the condition is true
- *Note: any non-zero expression is considered true*

```
int age;
cin >> age;
if (age < 13) {
  cout << "You are not yet a teenager!" << endl;
}
```

If statements

- Without curly braces, only next statement is attached to the control statement.
- So, if you want multiple statements to be executed, use curly braces.
- *Note: this also applies to `else` and `else-if` statements*

```
if (cond1) {
  statement1;
  statement2;
}
```

If statements (cont.)

```
int main() {
  int x = 3;
  if (x == 5)
    cout << "x is 5" << endl;
    cout << "In if" << endl; // Incorrectly
                             // called!
}
```

```
int main() {
  int x = 5;
  if (x == 5) {
    cout << "x is 5" << endl;
    cout << "In if" << endl;
  }
}
```

Else statements

- Performed when all `if` and `else if` conditions fail

```
int number;
cin >> number;
if (number % 2 == 0)
    cout << "You gave an even number" << endl;
else
    cout << "You gave an odd number" << endl;
```

Else-if statements

- Allows us to check for more than the `if` condition and its complement

```
if (cond1)
    statement1;
else if (cond2)
    statement2;
else if (cond3)
    statement3;
else
    statement4;
```

Comparison pitfalls

- **Equals-equals (==) vs. Equals (=)**
- These operators are very different!

```
if (x == y) // Returns true if x and y are equal

if (x = y)  // Assigns the value of y to x and returns the value
            // ASSIGNED to x.
```

Conditional confusion?

- Does this output anything?

```
int age = 17;
if (age) {
  cout << "You are not 0 years old!" << endl;
}
```

Conditional confusion?

- What does this output?

```
int age = 0;
if (age) {
  cout << "You are not 0 years old!" << endl;
} else {
  cout << “You are 0 years old!” << endl;
}
```

Switches

- Arguably a more compact alternative to long `if/else if/else` sequences
- The value tested must be an integral type or convertible to one
  - e.g. `int`, `char`, `short`, `long`, etc.
  - `string` is not a permitted type
- A `break` statement must be used to leave the switch. Otherwise execution will fall through to the next case.

Switches (cont.)

```
string value; int number;
cin >> number;
switch (number) {
  case 0:  // Fall-through to Case 2.
  case 2:
    value = "Good";
    break; // Remember to break!
  case 3:
    value = "Bad";
    break;
  default:
    value = "Ugly";
    break;
}
```

Common question 1: is a `break` statement required for the `default` case?

Switches (cont.)

```
string value; int number;
cin >> number;
switch (number) {
  case 0:  // Fall-through to Case 2.
  case 2:
    value = "Good";
    break; // Remember to break!
  case 3:
    value = "Bad";
    break;
  default:
    value = "Ugly";
    break;
}
```

Common question 1: is a `break` statement required for the `default` case?

No, if the default case is at the end. However, we recommend that you put one anyways. This allows the `default` case to appear in a different order, and not necessarily at the end of the switch statement.

Switches (cont.)

```
string value; int number;
cin >> number;
switch (number) {
  case 0:  // Fall-through to Case 2.
  case 2:
    value = "Good";
    break; // Remember to break!
  case 3:
    value = "Bad";
    break;
  default:
    value = "Ugly";
    break;
}
```

Common question 1: is a `break` statement required for the `default` case?

No if the default case is at the end. However, we recommend that you put one anyways. This allows the `default` case to appear in a different order, and not necessarily at the end of the switch statement.

Common question 2: do I need a default statement to compile?

Switches (cont.)

```
string value; int number;
cin >> number;
switch (number) {
  case 0:  // Fall-through to Case 2.
  case 2:
    value = "Good";
    break; // Remember to break!
  case 3:
    value = "Bad";
    break;
  default:
    value = "Ugly";
    break;
}
```

Common question 1: is a `break` statement required for the `default` case?

No if the default case is at the end. However, we recommend that you put one anyways. This allows the `default` case to appear in a different order, and not necessarily at the end of the switch statement.

Common question 2: do I need a default statement to compile?
No, but it is good to have to catch unexpected cases. You should leave a `//comment` if you don't have a `default` to explain why!

While loops

- `while` loops run code until the condition is false

```
int count;
cin >> count;
while (count >= 0) {
  cout << "Countdown: " << count << endl;
  count--;
}
```

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
```

0

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
```

1

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
```

1

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
> 1
```

1

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
> 1
```

2

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
> 1
```

2

x

While loops

```
int main() {
  int x = 0;
  while (x < 2) {
    cout << x << endl;
    x++;
  }
  cout << "Done!" << endl;
}
```

```
> 0
> 1
> Done!
```

2

x

For loops

- **Declaration** is run once before anything else
- **Condition** is evaluated before the **code block** is executed
- **Action** is run after the code block is executed

```
for (declaration; condition; action) {
  //code block
  statement1;
  statement2;
}
```

Note: all of the three sections of the for loop are ***optional***; all that is required is the semicolon. If `condition` is empty, it ***defaults to always `true`***. Example: `for(int i = 0;;i++) { //infinite loop} }`

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

0

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
```

0

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
```

1

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
```

1

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

1

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

2

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

2

i

For loops

```
int main() {
  for (int i = 0; i < 2; i++) {
    cout << "i is now equal to: " << i <<
        endl;
  }

  // Note that i is now out of scope.
  cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
> Done!
```

Nested loops

```
for (int i = 1; i <= 10; i++) {
  for (int j = 1; j <= 10; j++)  {
    cout << (i * j) << "\t";
  }
  cout << endl;
}
```

Quick Question - Breaking the Outer Loop

- What happens when you `break` inside nested loops?

Quick Question - Breaking the Outer Loop

- What happens when you `break` inside nested loops?
  - Only the loop that contains the `break` statement is broken out of.

Quick Question - Breaking the Outer Loop

- What happens when you `break` inside nested loops?
  - Only the loop that contains the `break` statement is broken out of.
- Solution: use a boolean variable (a *flag*) in your loop's statements and change the boolean from true to false when you want to break out of a nested loop.

```
bool keepLoopingI = true;
for (int i = 1; i <= 100; i++) {
     for (int j = 1; j <= 200; j++) {
          if (i+j > 250)
               keepLoopingI = false;
          cout << i << ",” << j << endl;
     }
     if (!keepLoopingI) break;
}
```

What does this print?

Practice Question: Pile of Money

Print a pile of money `n` stacks high, leaning up on the right against your mansion's wall.

Example for `n=4`:

```
   $|
  $$|
 $$$|
$$$$|
```

(Contributed by Matthew Wong)

Solution: Pile of Money

```
int n = 5; // Or any positive integer value

// Loop through all n rows
for (int i = 1; i <= n; i++) {
  // print the spaces preceding the dollar signs
  for (int spaces = 0; spaces < n - i; spaces++)
    cout << " ";

  // print the i dollar signs for the row
  for (int money = 0; money < i; money++)
    cout << "$";

  // print the wall and move to the next line
  cout << "|" << endl;
}
```

Practice Question: String to Int

Convert a variable of type `string` into the `integer` represented by that string.

Example, convert:

```
string number = “125”
```

into

```
int numberAsInt // holding the value 125 as an integer
```

Hints:

// `string name = “Daniel”;`

// `name[0]` is ‘D’

// `name.size()` is 6

// ‘8’ - ‘0’ *(the character eight minus the character zero)* is 8 *(the integer eight)*

(Contributed by Katie Luangkote)

Solution: String to Int

```
  string number = “125”; // or let the user cin their own string
  int result = 0;
  int multiplier = 1;
  for (int i = number.size() - 1; i >= 0; i--) {
    result += (number[i] - '0') * multiplier;
    multiplier *= 10;
  }
  int numberAsInt = result;

// “125” becomes 5 + 2*10 + 1*100 = 125
// Any other way to do this?
// std::stoi(string str)
```

Practice Question: `isPalindrome`

Write a function `is_palindrome` that takes a string as an argument and returns true if the string is a **palindrome** and false if it is not. A palindrome is a string that is read backwards the same way as it is forwards. For example `"racecar"` backwards is `"racecar"`.

Solution: `isPalindrome`

Start by checking that the first and the last letters in the string are equal, then move inward until we reach the middle of the string.

Note: This only requires `n / 2` iterations of the for-loop. Also, this code works even when `n` is **odd**. Trace through an example by hand to see why!

```
bool is_palindrome(string s) {
  int n = s.size();
  for (int i = 0; i < n / 2; i++) {
    // i is left char's index
    int j = (n - 1) - i;
    // j is right char's index
    if (s[i] != s[j]) return false;
  }
  return true;
}
```

Solution: `isPalindrome` with `while` loop

Here's a way to solve the same problem but with a while loop.

```
bool is_palindrome (string str) {
    int left_index = 0;
    int right_index = str.size() - 1;
    while (left_index < right_index) {
        if (str[left_index] != str[right_index])
            return false;
        left_index++;
        right_index--;
    }
    return true;
}
```

```
bool is_palindrome(string s) {
  int n = s.size();
  for (int i = 0; i < n / 2; i++) {
    // i is left char's index
    int j = (n - 1) - i;
    // j is right char's index
    if (s[i] != s[j]) return false;
  }
  return true;
}
```

Practice Question: Get “Switchy”

```
int main() {
  string morty;
  int rick = 5;
  rick = (rick + 2 * 5) / 10;
  switch (rick) {
    case 1:
      morty = “lubba”;
      break;
    case 3:
      morty = “aw geez”;
      break;
    default:
      morty = “oh man”;
      break;
  }
  ...
  ...
  morty += morty;
  morty[0] = ‘w’;

  for (int i = 0; i < 2; i++)
    morty += “dub”;

  cout << morty << endl;
}
```

What is the output of this code?

Solution: Get “Switchy”

```
int main() {
  string morty;
  int rick = 5;
  rick = (rick + 2 * 5) / 10; rick = 1
  switch (rick) {
    case 1:
      morty = “lubba”; morty = “lubba”
      break;
    case 3:
      morty = “aw geez”;
      break;
    default:
      morty = “oh man”;
      break;
  }
  ...
```

```
  ...
  morty += morty; morty = “lubbalubba”
  morty[0] = ‘w’; morty = “wubbalubba”

  for (int i = 0; i < 2; i++)
    morty += “dub”; morty = “wubbalubbadubdub”

  cout << morty << endl;
}
```

Output: wubbalubbadubdub

Translation: I am in great pain please help me

Scoping

- Variables only exist within the curly brackets or the implied curly brackets that they were written in.

```
if (cond1) {
  statement1;
}
```

Scoping (cont.)

```
if (cond1) {
  int x = 5;
  cout << x << endl; // No error
} // x is destroyed!

cout << x << endl; // Error!! x doesn't exist
                   // outside the if statement
```

Scoping (cont.)

```
int x = 1;
if (cond1) {
  x = 5;
  cout << x << endl; // No error
}

cout << x << endl; // No error here either!
```

Scoping (cont.)

```
string s1 = “bonjour”;
for (int i = 0; i < s1.size(); i++) {
  char lastChar = s1[i];
}

// Both i and lastChar don't exist here!
cout << i << " " << lastChar << endl; // Error!
```

Scoping (cont.)

```
string s1 = “bonjour”;
int i; char lastChar;
for (i = 0; i < s1.size(); i++) {
  lastChar = s1[i];
}

// Now both i and lastChar exist here
cout << i << " " << lastChar << endl;
```

Scoping: Switch Statements

```
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1:
      int x = 10;
      cout << "You entered 1! 1 times 10 is " << x << endl;
      break;
    default:
      int x = 5;
      cout << "You didn't enter 1" << endl;
  }
}
```

Scoping: Switch Statements

Warning: the cases of a `switch` statement share the same scope!

```
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1:
      int x = 10;
      cout << "You entered 1! 1 times 10 is " << x << endl;
      break;
    default:
      int x = 5;  // This is an error! Compiler says “error: redefinition of 'x'”
      cout << "You didn't enter 1" << endl;
  }
}
```

Scoping: Switch Statements

Warning: the cases of a `switch` statement share the same scope!

```
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1:
      int x = 10;
      cout << "You entered 1! 1 times 10 is " << x << endl;
      break;
    default:
     // Does x exist here? It's within the switch’s curly braces, but "int x = 10" was never executed?!
      cout << "You didn't enter 1" << endl;
  }
}
```

Scoping: Switch Statements

Warning: the cases of a `switch` statement share the same scope!

```
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1:
      int x = 10;
      cout << "You entered 1! 1 times 10 is " << x << endl;
      break;
    default:
      // So, this is also an error! Compiler says "note: jump bypasses variable initialization"
      cout << "You didn't enter 1" << endl;
  }
}
```

Scoping: Switch Statements

Warning: the cases of a `switch` statement share the same scope!

```
int main() {
  int n;
  cin >> n;
  switch (n) {
    case 1: {
      int x = 10; // Now x is only known to the scope of this case
      cout << "You entered 1! 1 times 10 is " << x << endl;
      break;
    }
    default:
      cout << "You didn't enter 1" << endl;
  }
}
```

```
int main(){
  {
    int x = 1;
  }
}
```

Introduction to Functions

- Creating a function
  - Define the function prototype/signature (the function name, function parameters)
  - Give it a function body, which is the function implementation
    - Code what the function is going to do with its parameters

function name
parameter type
return type
parameter name

```
bool foo(int bar) {
    if (bar == 2020)
        return true;
    return false;
}
```

function body (implementation)

Introduction to Functions

- Using a function
  - Write a function call by using the function name and passing in the correct number of parameters, correct type of parameters
- We can state that a function exists by just writing the function prototype first, and writing the function body later.

```
bool foo(int bar);    // function prototype; if we remove this line, won't compile
int main() {
     foo(2019);      // because we declared foo's prototype earlier, we can call it here
}

bool foo(int bar) {  // now we write out the function body
    if (bar == 2020)
        return true;
    return false;
}
```

Introduction to Functions: Trivial Examples

- Functions can call other functions
  - main() is a function, and we often use it to call other functions

```
// Add two ints and return the sum.
int add(int x, int y) {
    int sum = x + y;
    return sum;
}

// Multiply two ints and return the product.
int multiply(int x, int y) {
    int product = x * y;
    return product;
}
```

```
// Compute x^2 + y^2.

int squared_add(int x, int y) {
    int x_sq = multiply(x, x);
    int y_sq = multiply(y, y);

    return add(x_sq, y_sq);
}
```

Functions: Scope

- The function body is wrapped in curly braces { }, so that the variables declared inside the body do not exist outside the function.
- Variables declared outside the function do not exist inside the function <u>unless</u> the variable was a global variable.

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

int main() {
    int result = add(2,3);
    // let's try to use sum
    // which is a variable from inside add
    cout << sum << endl; // WON'T COMPILE
}
```

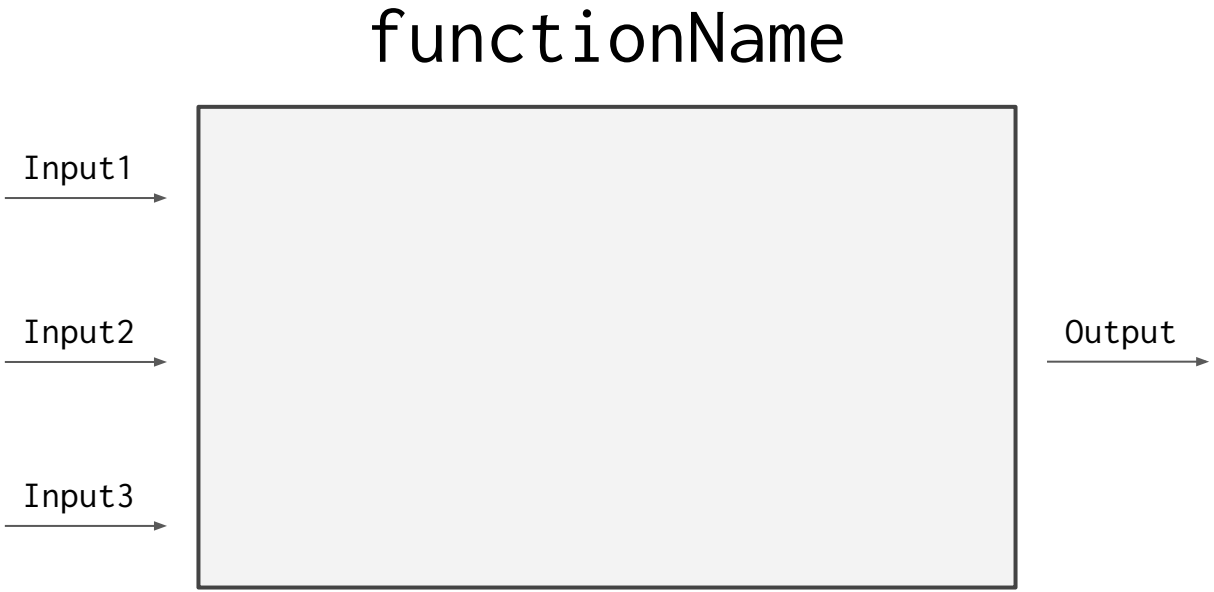
```
#include <iostream>
using namespace std;
void fubar(int foo);
int main() {
    int year = 2020;
    fubar(year);
}
// let's try to use year
// which is a variable from outside fubar
void fubar(int foo) {
    cout << year << endl; // WON'T COMPILE
}
```

Functions: Scope

- The function body is wrapped in curly braces { }, so that the variables declared inside the body do not exist outside the function.
- Variables declared outside the function do not exist inside the function <u>unless</u> the variable was a global variable.

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

int year = 2020;          // global variable

void fubar(int foo);
int main() {
    fubar(year);
}
void fubar(int foo) {
    cout << year << endl; // WILL compile
}
```

Functions: The Box Model



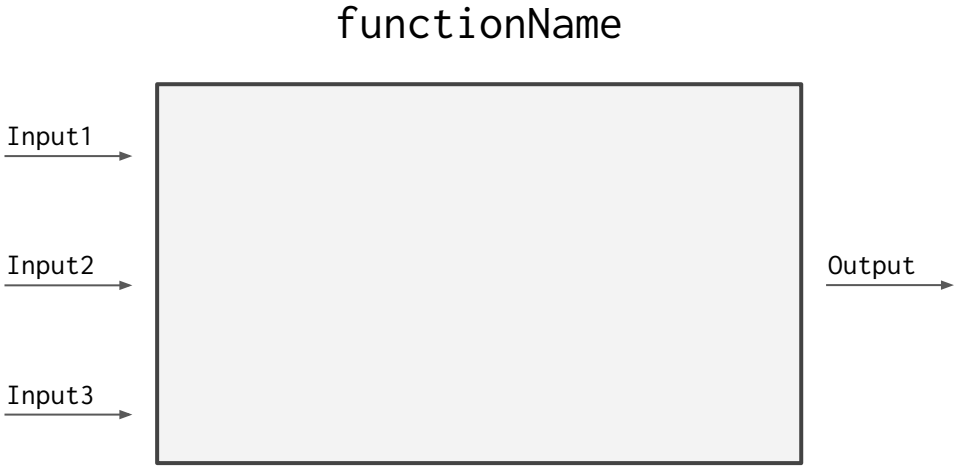
Functions: Example

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

int hypotenuse(int side1, int side2) {
     /* Function body */
     /* We don't need to know how the function is implemented */
     ...
}

int main() {
  int x = hypotenuse(3,4);
}
```

Functions: Mathematical Example

- f(x, y, z) = x - y - z;
- f(10, 5, 4) = 10 - 5 - 4 = 1
- f(3, 5, 4) = 3 - 5 - 4 = -6

- Symbolically, x, y, and z are input1, input2, and input3



Math Example (cont.)

Suppose we define *f(x, y, z) = x - y - z.*

```
int main() {
  int b = 5;
  int a = 4;
  int c = 6;
  cout << f(c,b,a) << endl;
}
```

Then for some variables *a*, *b*, *c*, we have *f(c, b, a) = c - b - a = 6 - 5 - 4 = -3.*

```
> -3
```

5	4	6
b	a	c

Mathematical Example (cont.)

```
#include <iostream>
using namespace std;
int functionName(int num1, int num2, int num3) {
  return num1 - num2 - num3;
}
```

- f(x, y, z) becomes functionName(num1, num2, num3)
- functionName(10, 5, 4) = 10 - 5 - 4 = 1;
- functionName(3, 5, 4) = 3 - 5 - 4 = -6;

Functions: Nested Loops Example

Problem: Write a function `firstChar(string string1, string string2)` that returns the first character in `string1` that exists in `string2`, or returns the null byte if no such character is found.

Examples:

- `string1 = “hello”, string2 = “there”, result = ‘h’`
- `string1 = “aabc”, string2 = “xyzab”, result = ‘a’`
- `string1 = “aabc”, string2 = “xyzzb”, result = ‘b’`
- `string1 = “abcd”, string2 = “wxyz”, result = ‘\0’`

Functions: The Box Model

```
char firstChar(string string1, string string2);
```

Functions: Nested Loops Solution

```
char firstChar(string string1, string string2) {
  for (int i = 0; i < string1.size(); i++)
    for (int j = 0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}
```

Functions: Calling the `firstChar` Function

```
int main() {
  cout << firstChar(“hello”, “there”) << endl;
  cout << firstChar(“aabc”, “xyzab”) << endl;
  char c = firstChar(“aabc”, “xyzzb”);
}
```

Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```

Walkthrough

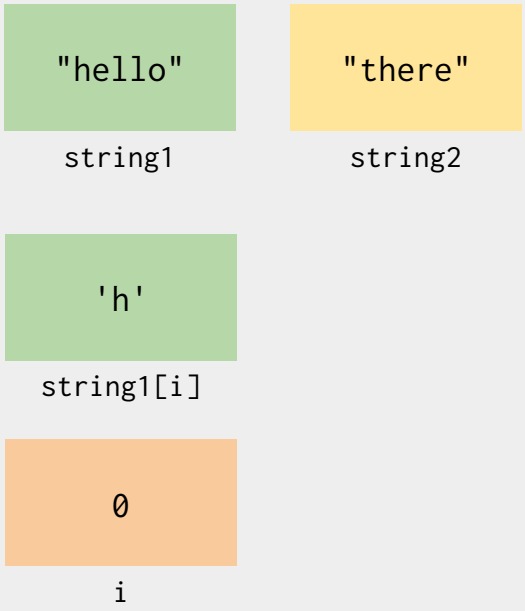
```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```

Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```

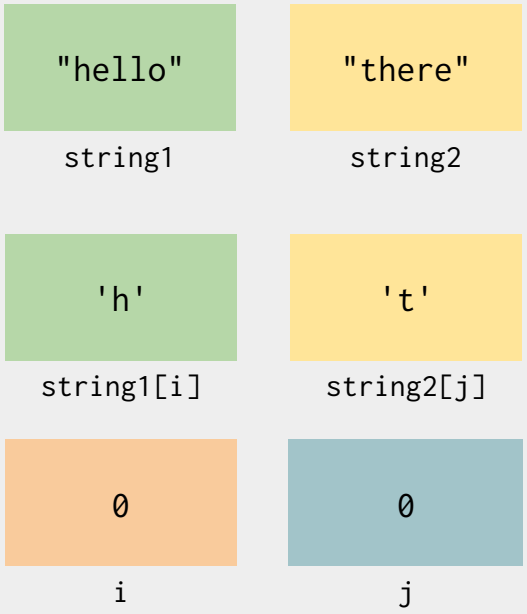
Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



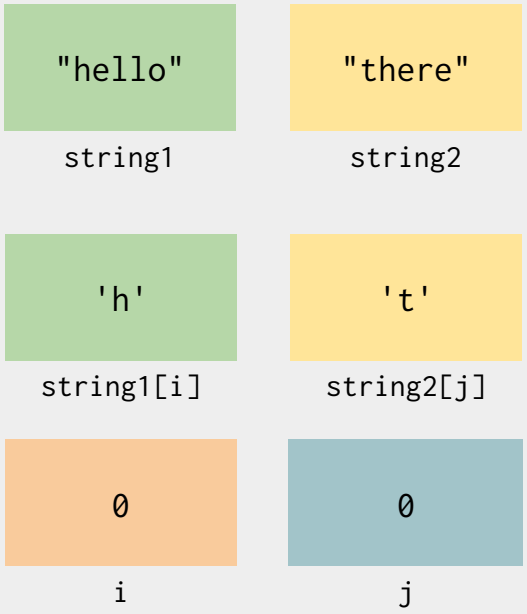
Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



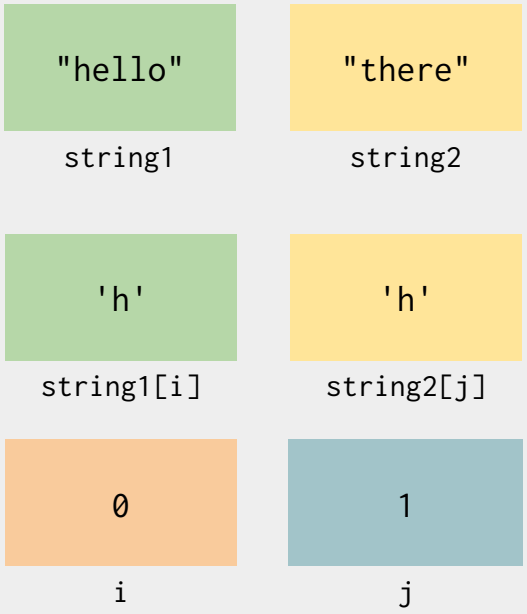
Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



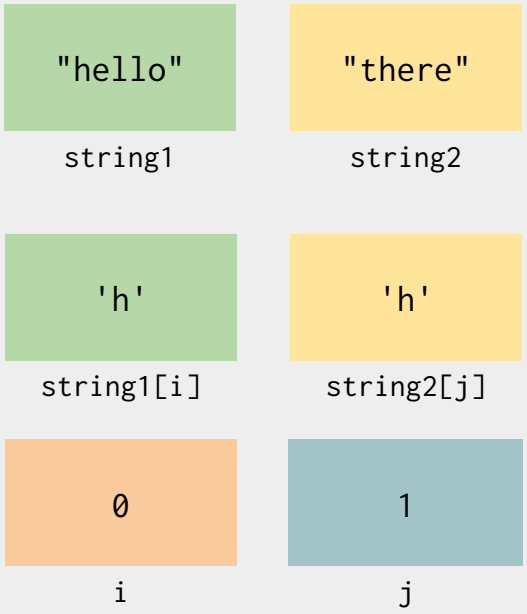
Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



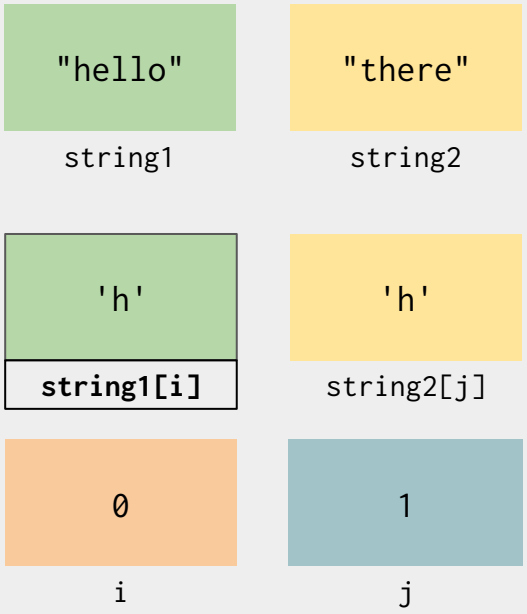
Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```

```
> h
```

Walkthrough

```
char firstChar(string string1,
    string string2) {
  for (int i=0; i < string1.size(); i++)
    for (int j=0; j < string2.size(); j++)
      if (string1[i] == string2[j])
        return string1[i];
  return '\0';
}

int main() {
  cout << firstChar("hello", "there");
}
```

```
> h
```

Functions: Parameters

- The types, modifiers, order, and number of parameters are all important in a function declaration
  - types: `string`, `int`, `bool`, etc.
  - modifiers: `&`, `const`, `*`, etc. (*Note: these aren't on the midterm!)
  - number: how many parameters are passed to a particular function?

- Function call **must** match the pattern of the declaration

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  string x = isEqual(s1,s2, position);
}
```

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  string x = isEqual(s1,s2, position);
}
```

No. The return type is boolean.

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(s1,s2);
}
```

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(s1,s2);
}
```

No. The number of arguments doesn't match.

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(position, s1, s2);
}
```

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(position, s1, s2);
}
```

No. The order of arguments doesn't match.

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(s1, s2, position);
}
```

Will this compile?

```
// Assume this function is defined later.
bool isEqual(string s1, string s2, int position);

int main() {
  string s1 = "hello";
  string s2 = "there";
  int position = 1;
  bool x = isEqual(s1, s2, position);
}
```

Yes.

Good luck!

UPE Links

Sign-in	https://tinyurl.com/cs31m1fall21form
Feedback:	https://tinyurl.com/upe31mt1feedback
Slides	https://tinyurl.com/cs31m1fall21slides
Practice	https://github.com/uclaupe-tutoring/practice-problems/wiki

Questions? Need more help?

- Come up and ask us! We'll try our best.
- UPE offers daily computer science tutoring:
  - Location: ACM/UPE Clubhouse (Boelter 2763)
  - Schedule: https://upe.seas.ucla.edu/tutoring/
- You can also post on the Facebook event page.