Getting Ready for Canadian Computing Competition

Class 6
January 4, 2020

Outline

- Hop_steps Solution
- Call Stack
- Reading Input
- Final Version of Bubble Sort
- STL sort
- Range based for-loop
- Lvalue references
- Constructor

Solutions

A child is running up a staircase with N steps. They can hop 1 step, 2 steps or 3 steps at a time. Count how many possible ways the child can run up to the stairs.

Sample input:

3

Sample output:

4

Explanation:
Method 1: Hop 3 steps
Method 2: Hop 2 steps, then 1 step
Method 3: Hop 1 step, then 2 steps
Method 3: Hop 1 step, then 1 step, then 1 step

Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

A stack keeps track of all local variables and arguments supplied to the function.

Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

main:

a = 2

b = 3

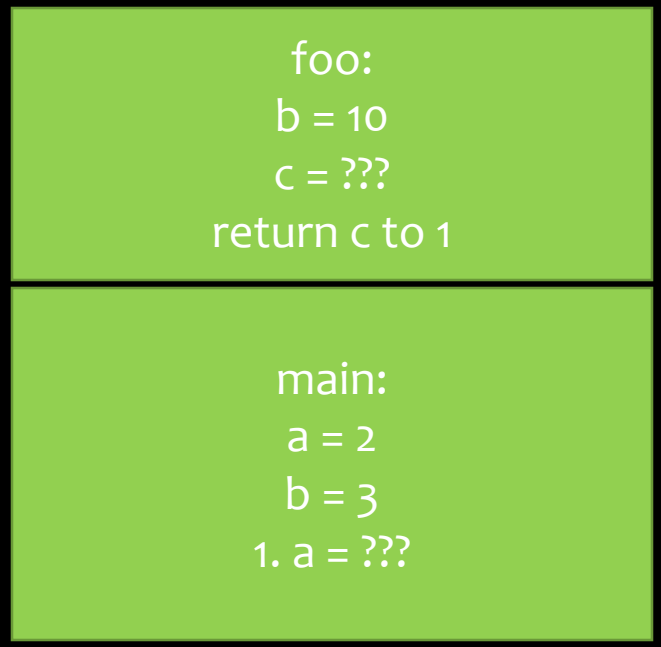
a = ???

Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

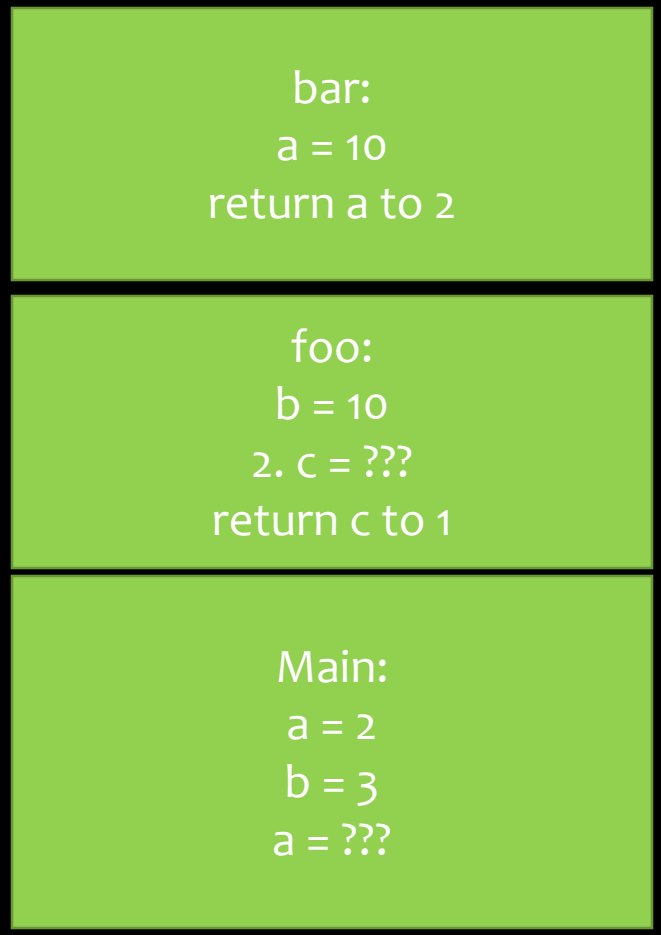


Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```



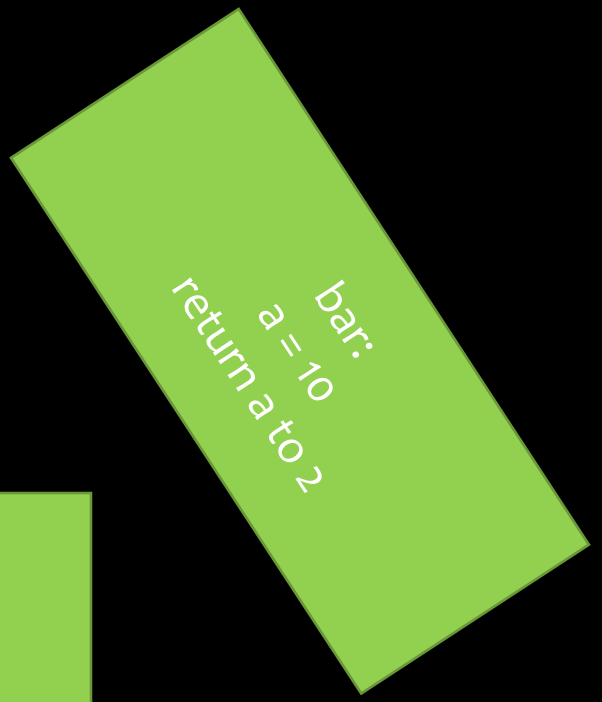
Call Stack

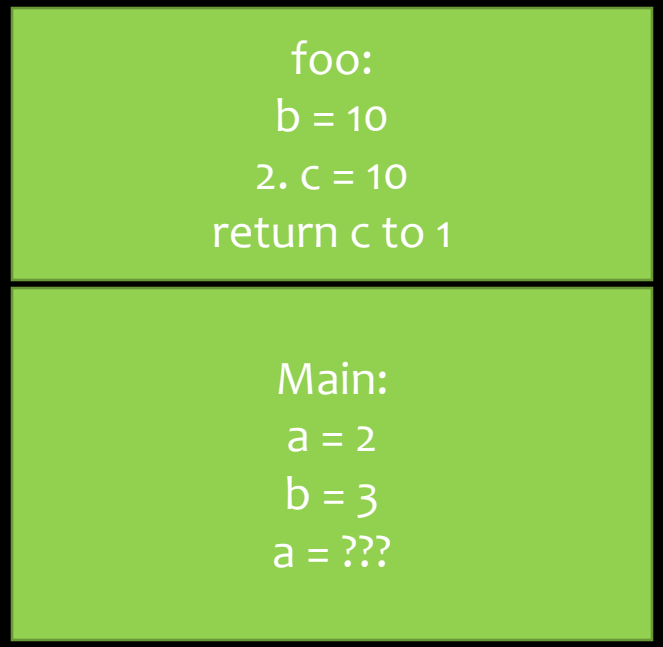
```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

Pop it off the stack, addresses that stored these variables are free, these data are “garbage values”





Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

Main:

a = 2

b = 3

a =10

Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

← Now executing

Main:

a = 2

b = 3

a = ~~10~~ 12

Call Stack

```
int bar(int a) {
    return a;
}

int foo() {
    int b = 10;
    int c = bar(b);
    return c;
}

int main() {
    int a = 2;
    int b = 3;
    a = foo();
    a = 12;
    return 0;
}
```

Main:
a = 2
b = 3
a =12

We finally pop main off when we see “return 0”

Exercise

Draw the stack when the code in blue is executing:

```
int bar(int a) {
    return a * 2;
}
int foo(int a, int b) {
    if (a == b) return a + 1;
    else {
        return bar (b);
    }
}
int main() {
    int k = 1, m = 2;
    k = foo(m, k);
    int a = foo(k, m);
    a = 4;
    return 0;
}
```

Reading Input

- Using a helper function

Example: Read in n integers

Sample input:

3

1 2 3

Reading Input Sample Code

```
vector <int> read_input () {
    int n, input;
    cin >> n;
    vector <int> temp;
    for (int i = 0; i < n; ++i) {
        cin >> input;
        temp.emplace_back(input);
    }
    return temp;
}

int main() {
    vector <int> data = read_input(); // pass by value
}
```

Reading Input Sample Code

```
vector <int> * read_input () {
    int n, input;
    cin >> n;
    vector <int> temp;
    for (int i = 0; i < n; ++i) {
        cin >> input;
        temp.emplace_back(input);
    }
    return &temp;
}

int main() {
    vector <int> *data = read_input();
}
```

Reading Input

- Read your input in main
- Or create a vector in main and pass it by reference to a read_input function

Final Bubble Sort Version

```
void bubbleSort(vector <int> arr, int n) {
    bool swapped;
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            swapped = false;
            if (arr[j] > arr[j+1]) {
                swap(arr[j], arr[j+1]); // or swap(&arr[j], &arr[j+1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}
```

<algorithm> - sort

- There is a built- in sort function in C++ STL
- Include <algorithm> as header
- Sort (iterator start_loc, iterator end_loc[,compare])
- Sorts from [start_loc, end_loc)
- End_loc must be recheable from start_loc by incrementing start_loc
- Compare is optional and has default operator<

Sorting an Array

Sort (iterator start_loc, iterator end_loc[,compare])

We want to sort this array:

int arr[3]

What is start_loc?

A. arr
B. arr[0]
C. arr.begin()
D. &arr

Sorting an Array

Sort (iterator start_loc, iterator end_loc[,compare])

We want to sort this array:

int arr[3]

What is end_loc?

A. arr + sizeof(int) * 3

B. arr + 12

C. arr + 3

D. arr.end()

Example

Sorting an integer array.

```
 int arr[3] = {2, 3, 1};
sort (arr, arr + 3);
for (int i = 0; i < 3; ++i) {
    cout << arr[i] << " ";
}
```

```
Output: 1 2 3
```

Exercise

What will this print?

What will this print?

```
int arr[3] = {2, 3, 1};
sort (arr, arr + 2);
for (int i = 0; i < 3; ++i) {
    cout << arr[i] << " ";
}
```

A. Error
B. 2 3 1
C. 1 2 3
D. 2 3 0

Exercise

What will this print?

```
int arr[10] = {2, 3, 1};
    sort (arr, arr + 4);
    for (int i = 0; i < 3; ++i) {
        cout << arr[i] << " ";
    }
```

A. Error
B. 0 1 2
C. 1 2 3
D. 2 3 1

Exercise

How do you sort a vector?

Example:

vector <int> temp {1,3,4, 0, 2}; // sort this vector

// your code goes here

Exercise

How do you sort a vector?

Example:

vector <int> temp {1,3,4, 0, 2}; // sort this vector

sort(temp.begin(), temp.end());

Sorting Order

- What if we want to sort in ascending order?
- Recall sort:

sort (iterator start_loc, iterator end_loc[,compare])

- We can pass in a function to replace compare
- We can also change the default < operator (next class)

Example

```
bool my_greater(int a, int b) {
    return a > b;
}

int main() {
    vector<int> a = {2,3,1};
    sort (a.begin(), a.end(), my_greater);
    for (int i = 0; i < 3; ++i) {
        cout << a[i] << " ";
    }
}
```

Note: make sure that you have no variable named my_greater in main

Exercise

Recall the structure and function

```
struct Student {

    string name;

    int studentNum, grader;

};

bool compare_student (Student s1, Student s2) {

    return (s1.grade == s2.grade) ? (s1.studentNum >
s2.studentNum) : (s1.grade > s2.grade);

}
```

Exercise

Complete the main function to sort the students

```
int main() {
    vector<Student> a = {{"a", 0, 90}, {"b", 1, 98}, {"c", 2, 90}};
    // your code goes here
    for (int i = 0; i < 3; ++i) {
        cout << a[i].name << " ";
    }
}
```

This should print “b c a”.

Practice

Jasmine plans to live in a forest for k days. This forest has n berry trees with each tree producing 50 berries per day. Jasmine lives Tn minutes away from each tree. She returns home immediately after picking one berry tree to keep the berries fresh. If berries are not eaten on the day they were picked, they will go bad. If Jasmine has visited a tree before, it will take her half the time (Tn/2) to get to the same tree next time. She has already visited some trees. Given that she needs B + 25d berries a day, where d is number of days she has lived in the forest and B is a base value, produce a summary of the shortest time she needs to pick all her berries each day. Note that 50n >= B, which means that Jasmine will have enough berries before she runs out of berry trees.

Practice

Input Specification:

k B n
followed by n lines of Tn Vn)
Where Vn = 1 if Jasmine has visited it, otherwise Vn = 0

Output specification:
K rows, each in the format:
day d: time needed

Sample input:

3 150 5
5 0
12 1
6 0
9 1
10 0

Sample output:

day 1: 21.5
day 2: 16
day 3: 26

Range Based for-loop

- Executes a for loop over a range of values
- Syntax:

for (range_declaration : range_exp) {…}

- range_declaration defines that variable that is the type in the range_exp

Example:

```
 for (int i : {1,3,5}) {
        cout << i ;
   }
// prints 1 3 5
```

Range Based for-loop

- Executes a for loop over a range of values
- Syntax:

for (range_declaration : range_exp) {…}

- range_declaration defines that variable that is the type in the range_exp

Example:

```
 for (int i : {1,3,5}) { // but this is too much work…
        cout << i ;
   }
// prints 1 3 5
```

Range Based for-loop

- Executes a for loop over a range of values
- Syntax:

for (range_declaration : range_exp) {…}

- range_declaration defines that variable that is the type in the range_exp

Example:

```
 for (auto i : {1,3,5}) { // auto detection of type
        cout << i ;
   }
// prints 1 3 5
```

Range Based for-loop

To print an array:

```
int arr[] = {1, 2, 3};
 for (auto i: arr) {
     cout << i << " ";
 }
```

To print a vector:

```
 vector <int> test = {1,2,3};
 for (auto i: test) {
     cout << i << " ";
 }
```

Exercise

What is the type of c?

```
string name = "Jamsine";
for (auto c: name) {
    cout << c;
}
```

A. String
B. Char
C. Error
D. Int

Range Based for-loop

vector <int> test = {1,2,3}; // this prints a vector

```
for (auto i: test) {
    cout << i << " ";
}
```

This is similar to:

```
vector <int> test = {1,2,3};
int len = test.size();
for (int j = 0; j < len; ++j) {
    auto i = test[j];
    cout << i << " ";
}
```

Exercise

What will this print?

```
vector <int> test = {1,2,3};
for (auto i: test) {
    i = 0;
}
for (auto i: test) {
    cout << i << " ";
}
```

A. 1 2 3
B. 0 0 0

Exercise

What will this print?

```
vector <int> test = {1,2,3};
for (auto i: test) {
    i = 0;
}
for (auto i: test) {
    cout << i << " ";
}
```

A. 1 2 3
B. 0 0 0

What if we really want to modify test to contain all 0's?

Lvalue References

- An lvalue points to a memory address
  - A variable
- An rvalue does not point anywhere
  - Temporary
  - Short-lived

- Example:

int a = 1; // 1 is destroyed after this line is executed

a -> lvalue

1 -> rvalue

vec.push_back() vs vec.emplace_back()

- vec.push_back(x) makes a copy of x and pushes the value onto vec

- vec.emplace_back(x) checks if x is an rvalue that is about to be destroyed. If it is, it will “steal” x and put in on vec instead of making a copy
  - “Copying is bad, you should be stealing”

Lvalue References

- Other than pointers, we can use an lvalue reference to keep track of memory addresses
- Lvalues do not store memory addresses
- They are **aliases**
  - They are “different names” of the same variable
- They must be initialized at declaration
- Declaration syntax:

Type & name = variable;

Example

We can change the value of a variable implicitly using lvalue references:

```
int a = 10;

int &b = a;

b = 12;

cout << a << endl; // 12
```

Passing by Reference

- We can use lvalues instead of pointers to pass by reference
- Last class:

```
void foo (vector <int> *t) {
    (*t) ... // this will get you temp
}

int main() {
    vector <int> temp = {1,2,3};
    foo (temp);
}
```

Passing by Reference

- Now:

```
void foo (vector <int> t) {
    t ... // this will get you temp
}
```

Exercise - swap

In bubble sort, we used the C++ built-in function swap to swap the location of two integers. We can now implement our own swap.

```
void my_swap (int *a, int *b) {

    int temp = *b;

    *b = *a;

    *a = temp;

}
```

Exercise - Swap

Rewrite swap using lvalue references.

Example 2 – passing a vector by reference

Given the main function, edit_vec functions that changes the first element of v to 0

```
int main() {
    vector <int> v = {1,2,3};
    edit_vec (v);
    cout << v[0] << endl; // 0
}
```

Example 2 – passing a vector by reference

Given the main function, edit_vec functions that changes the first element of v to 0

```
int main() {
    vector <int> v = {1,2,3};
    edit_vec (v);
    cout << v[0] << endl; // 0
}
```

Solution:

```
void edit_vec (vector <int> &a) {
    a[0] = 1;
}
```

Back to Exercise

What will this print?

```
vector <int> test = {1,2,3};
    for (auto i: test) {
        i = 0;
    }
    for (auto i: test) {
        cout << i << " ";
    }
```

A. 1 2 3
B. 0 0 0

What if we really want to modify test to contain all 0's? change the i to an lvalue

Range Based for-loop

vector <int> test = {1,2,3}; // this prints a vector

```
for (auto i: test) {
    cout << i << " ";
}
```

This is similar to:

```
vector <int> test = {1,2,3};
int len = test.size();
for (int j = 0; j < len; ++j) {
    auto i = test[j];
    cout << i << " ";
}
```

Range Based for-loop

vector <int> test = {1,2,3}; // this prints a vector

```
for (auto i: test) {
    cout << i << " ";
}
```

This is similar to:

```
vector <int> test = {1,2,3};
int len = test.size();
for (int j = 0; j < len; ++j) {
    auto &i = test[j];
    cout << i << " ";
}
```

Back to Exercise

Simply change the i to an lvalue

```
vector <int> test = {1,2,3};
    for (auto &i: test) {
        i = 0;
    }
    for (auto i: test) {
        cout << i << " ";
    }
```

So why pointers?

- Lvalue references must be initialized
- Lvalue cannot have nullptr
  - Useful for implement trees

Constructor

- Terminology:
  - When a struct is defined, no memory is allocated.
  - When a structure is declared, memory is allocated. We call this an **object** or **instance of the class**
- When an object is created, the constructor is called
- If we do not write a constructor, the C++ compile generates two default constructors for us
- Once we write our own, we lose the default constructors

Constructor

- Constructor has same name as the struct itself
- Constructors do not have return type because it is the object itself

Example:

```
struct Student {
    string name;
    int student_num, grade;
};
int main() {
    Student s1; // default constructor
    Student s2{"Jasmine", 1, 99}; // default constructor
}
```

Example

- Every object has a built-in pointer called **this** which points to itself
- Example:

Assume that we want to automatically assign a student number to every student. We also want to set grade = 0 for beginning of term.

```
int counter = 0;
struct Student {
    string name;
    int student_num, grade;
    Student (string name) { // same name as struct
        this->name = name;
        this->grade = 0;
        this->student_num = counter;
        ++counter;
    }
};
```

Calling a Constructor

The corresponding main function can be:

```
int main() {

    Student s1 {"A"};

    cout << s1.name << " " << s1.student_num << " " << s1.grade
<< endl; // A 0 0

    Student s2 {"B"};

    cout << s2.name << " " << s2.student_num << " " << s2.grade
<< endl; // B 1 0

}
```

Exercise

What would this line of code in main produce?

```
Student s3;
```

A. An uninitialized student
B. Error
C. Student {“C”, 2, 0}
D. A pointer

Exercise

What would this line of code in main produce?

Student s3 {“C”, 2, 0};

A. An uninitialized student
B. Error
C. Student {“C”, 2, 0}
D. A pointer

Simplify the Constructor

```
int counter = 0;
struct Student {
    string name;
    int student_num, grade;
    Student (string name) { // same name as struct
        this->name = name;
        grade = 0;
        student_num = counter;
        ++counter;
    }
};
```

C++ is smart enough to know grade means this->grade

Simplify the Constructor - MIL

- Member Initialization List
- Syntax: Constructor (parameters) : field_name1 {value from parameter}, field_name2 {value2} ,… field_namen {valuen} { body }
- You can leave out any field name and it will be filled with default values

```
int counter = 0;
struct Student {
    string name;
    int student_num, grade;
    Student (string name) : name {name}, grade{0}, student_num
{counter} {
       ++counter;
       }
};
```

Constructors

- You can define multiple constructors because C++ allows function overloading
- You can also declare constructors in the struct and define it outside. You must use StructName::ConstructorName instead of just ConstructorName

Exercise

1. Complete the definition of the constructor by setting x and y to 0.

```
struct Posn {
    int x, y;
    Posn();
};
```

2. Add another constructor that takes in two numbers and sets x and y accordingly

3. Write a main function that calls these two constructors

Constructor Calling

- Always use {} instead of ()
- Example:

```
struct Posn {
    int x, y;
};

int main() {
    Posn a(1,2); // error
    Posn b{1,2};
}
```

Practice

CCC 2016 S3