20.x — Chapter 20 summary and quiz

ALEX[1] FEBRUARY 8, 2025

Chapter Review

Another chapter down! There's just this pesky quiz to get past...

Function arguments can be passed by value, reference or address. Use pass by value for fundamental data types and enumerators. Use pass by reference for structs, classes, or when you need the function to modify an argument. Use pass by address for passing pointers or built-in arrays. Make your pass by reference and address parameters const whenever possible.

Values can be returned by value, reference, or address. Most of the time, return by value is fine, however return by reference or address can be useful when working with dynamically allocated data, structs, or classes. If returning by reference or address, remember to make sure you're not returning something that will go out of scope.

Function pointers allow us to pass a function to another function. This can be useful to allow the caller to customize the behavior of a function, such as the way a list gets sorted.

Dynamic memory is allocated on the heap.

The call stack keeps track of all of the active functions (those that have been called but have not yet terminated) from the start of the program to the current point of execution. Local variables are allocated on the stack. The stack has a limited size. std::vector can be used to implement stack-like behavior.

A recursive function is a function that calls itself. All recursive functions need a termination condition.

Command line arguments allow users or other programs to pass data into our program at startup. Command line arguments are always C-style strings, and have to be converted to numbers if numeric values are desired.

Ellipsis allow you to pass a variable number of arguments to a function. However, ellipsis arguments suspend type checking, and do not know how many arguments were passed. It is up to the program to keep track of these details.

Lambda functions are functions that can be nested inside other functions. They don't need a name and are very useful in combination with the algorithms library.

Quiz time

Question #1

Write function prototypes for the following cases. Use const if/when necessary.

a) A function named max() that takes two doubles and returns the larger of the two.

Show Solution (javascript:void(0))[2]

b) A function named swap() that swaps two integers.

Show Solution (javascript:void(0))[2]

c) A function named getLargestElement() that takes a dynamically allocated array of integers and returns the largest number in such a way that the caller can change the value of the element returned (don't forget the length parameter).

Show Solution (javascript:void(0))[2]

Question #2

What's wrong with these programs?

a)

```
int& doSomething()
{
    int array[]{ 1, 2, 3, 4, 5 };
    return array[3];
}
```

Show Solution (javascript:void(0))[2]

b)

```
int sumTo(int value)
{
    return value + sumTo(value - 1);
}
```

Show Solution (javascript:void(0))[2]

c)

```
float divide(float x, float y)
{
    return x / y;
}

double divide(float x, float y)
{
    return x / y;
}
```

Show Solution (javascript:void(0))[2]

d)

```cpp
#include <iostream>

int main()
{
    int array[100000000]{};

    for (auto x: array)
        std::cout << x << ' ';

    std::cout << '\n';

    return 0;
}
```

Show Solution (javascript:void(0))[2]

e)

```cpp
#include <iostream>

int main(int argc, char* argv[])
{
    int age{ argv[1] };
    std::cout << "The user's age is " << age << '\n';

    return 0;
}
```

Show Solution (javascript:void(0))[2]

Question #3

The best algorithm for determining whether a value exists in a sorted array is called binary search.

Binary search works as follows:

- Look at the center element of the array (if the array has an even number of elements, round down).
- If the center element is greater than the target element, discard the top half of the array (or recurse on the bottom half)
- If the center element is less than the target element, discard the bottom half of the array (or recurse on the top half).
- If the center element equals the target element, return the index of the center element.
- If you discard the entire array without finding the target element, return a sentinel that represents “not found” (in this case, we’ll use -1, since it’s an invalid array index).

Because we can throw out half of the array with each iteration, this algorithm is very fast. Even with an array of a million elements, it only takes at most 20 iterations to determine whether a value exists in the array or not! However, it only works on sorted arrays.

Modifying an array (e.g. discarding half the elements in an array) is expensive, so typically we do not modify the array. Instead, we use two integers (min and max) to hold the indices of the minimum and maximum elements of the array that we’re interested in examining.

Let's look at a sample of how this algorithm works, given an array { 3, 6, 7, 9, 12, 15, 18, 21, 24 }, and a target value of 7. At first, min = 0, max = 8, because we're searching the whole array (the array is length 9, so the index of the last element is 8).

- Pass 1) We calculate the midpoint of min (0) and max (8), which is 4. Element #4 has value 12, which is larger than our target value. Because the array is sorted, we know that all elements with index equal to or greater than the midpoint (4) must be too large. So we leave min alone, and set max to 3.
- Pass 2) We calculate the midpoint of min (0) and max (3), which is 1. Element #1 has value 6, which is smaller than our target value. Because the array is sorted, we know that all elements with index equal to or lesser than the midpoint (1) must be too small. So we set min to 2, and leave max alone.
- Pass 3) We calculate the midpoint of min (2) and max (3), which is 2. Element #2 has value 7, which is our target value. So we return 2.

In C++20, calculating the midpoint can be done via calling `std::midpoint` (https://en.cppreference.com/w/cpp/numeric/midpoint)[3]. Prior to C++20, you'll need to calculate the midpoint yourself -- in this case you can use `min + ((max - min) / 2)` to avoid overflow (when min and max are both positive).

Given the following code:

```cpp
#include <iostream>
#include <iterator>

// array is the array to search over.
// target is the value we're trying to determine exists or not.
// min is the index of the lower bounds of the array we're searching.
// max is the index of the upper bounds of the array we're searching.
// binarySearch() should return the index of the target element if the target is
found, -1 otherwise
int binarySearch(const int* array, int target, int min, int max)
{

}

int main()
{
    constexpr int array[]{ 3, 6, 8, 12, 14, 17, 20, 21, 26, 32, 36, 37, 42, 44, 48 };

    // We're going to test a bunch of values to see if the algorithm returns the
result we expect

    // Here are the test values
    constexpr int testValues[]{ 0, 3, 12, 13, 22, 26, 43, 44, 48, 49 };

    // And here are the results that we expect for those test values
    int expectedResult[]{ -1, 0, 3, -1, -1, 8, -1, 13, 14, -1 };

    // Make sure we have the same number of test values and expected results
    static_assert(std::size(testValues) == std::size(expectedResult));

    int numTestValues { std::size(testValues) };
    // Loop through all of the test values
    for (int count{ 0 }; count < numTestValues; ++count)
    {
        // See if our test value is in the array
        int index{ binarySearch(array, testValues[count], 0, static_cast<int>
(std::size(array)) - 1) };
        // If it matches our expected value, then great!
        if (index == expectedResult[count])
            std::cout << "test value " << testValues[count] << " passed!\n";
        else // otherwise, our binarySearch() function must be broken
            std::cout << "test value " << testValues[count] << " failed.  There's
something wrong with your code!\n";
    }

    return 0;
}
```

a) Write an iterative version of the binarySearch function.

Hint: You can safely say the target element doesn't exist when the min index is greater than the max index.

Show Solution (javascript:void(0))[2]

b) Write a recursive version of the binarySearch function.

Show Solution (javascript:void(0))[2]

Tip

`std::binary_search` (https://en.cppreference.com/w/cpp/algorithm/binary_search)[4] returns true if a value exists in a sorted list.

`std::equal_range` (https://en.cppreference.com/w/cpp/algorithm/equal_range)[5] returns the iterators to the first and last element with a given value.

Don't use these functions to solve the quiz, but use them in the future if you need a binary search.

Next lesson
21.1 Introduction to operator overloading

6

Back to table of contents

7

Previous lesson
20.7 Lambda captures

8

9

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[12] are connected to your provided email address.

367 COMMENTS Newest

AJAY.CHALLA
June 6, 2025 11:41 pm PDT

QUIZ

QUESTION 1

a) double max(double x, y);

b) void swap(int& x, int& y);

c) int& getlargestelement(int* array, int length);

QUESTION 2

a) doSomething() returns reference to the local variable that will be destroyed when doSomething ends.

b) function sumTo() has no end condition Variable value will go negative, and the function will loop infinite until the stack overflows.

c) The two divide functions are not different, as they have the same names and same parameters. There is also a potential divide by zero issue.

d) They array should be dynamically typed because it is too large.

e) argv [1] may not exist. If it is, argv [1] is a string argument, and cant be converted to an integer via assignment.

QUESTION #=3

a)

```
// #include <cassert>
#include <iostream>
#include <numeric>

int binarySearch(const int* array, int target, int min, int max)
{
    assert(array); // make sure array exists

    while (min <= max)
    {
        // implement this iteratively
        int midpoint{ std::midpoint(min, max) };
        // Before C++20
        // int midpoint{ min + ((max-min) / 2) };

        if (array[midpoint] > target)
        {
            max = midpoint - 1;
        }
        else if (array[midpoint] < target)
        {
            min = midpoint + 1;
        }
        else
        {
            return midpoint;
        }
    }

    return -1;
}
```

b)

```cpp
#include <cassert>
#include <numeric>

int binarySearch(const int* array, int target, int min, int max)
{
    assert(array);

    if (min > max)
        return -1;

    int midpoint{ std::midpoint(min, max) };
    // Before C++20
    // int midpoint{ min + ((max-min) / 2) };
    if (array[midpoint] > target)
    {
        return binarySearch(array, target, min, midpoint - 1);
    }
    else if (array[midpoint] < target)
    {
        return binarySearch(array, target, midpoint + 1, max);
    }

    return midpoint;
}
```

I guess these are correct!! ╭つ ◕_◕╮つ(•'◡'•)

0 Reply

Olawole

May 23, 2025 4:52 am PDT

```cpp
//Iterative
inline int getMidPoint(int min, int max)
{
    return min + ((max - min) / 2);
}

int binarySearch(const int* array, int target, int min, int max)
{

    while(true)
    {
        int midPoint{getMidPoint(min, max)};
        // check if the value at that index is the same as the index of the target
number
        //if yes return the index
        if(array[midPoint] == target)
        {
            return midPoint;
        }
        else if(array[midPoint] > target)
        {
            //if the midPoint index is greater than the target then change the value
of the max index
            max = midPoint - 1;
        }
        else
        {
            //if the midPoint index is less than the target then change the value of
the min index
            min = midPoint + 1;
        }

        if(min > max)
        return -1;
    }
}

//Recursive

int binarySearch(const int* array, int target, int min, int max)
{
    int midPoint{getMidPoint(min, max)};

    if(min > max)
    return -1;

    if(array[midPoint] == target)
    return midPoint;

    return (array[midPoint]  > target) ? binarySearch(array, target, min, midPoint -
1) : binarySearch(array, target, midPoint + 1, max);
}
```

Last edited 1 month ago by Olawole

0 Reply

Olawole

May 23, 2025 4:51 am PDT

`//Iterative`

`inline int getMidPoint(int min, int max)`

```
{
return min + ((max - min) / 2);
}

int binarySearch(const int* array, int target, int min, int max)
{
while(true)
{
int midPoint{getMidPoint(min, max)};
// check if the value at that index is the same as the index of the target number
//if yes return the index
if(array[midPoint] == target)
{
return midPoint;
}
else if(array[midPoint] > target)
{
//if the midPoint index is greater than the target then change the value of the max
index
max = midPoint - 1;
}
else
{
//if the midPoint index is greater than the target then change the value of the max
index
min = midPoint + 1;
}
if(min > max)
return -1;
}
}
//Recursive
int binarySearch(const int* array, int target, int min, int max)
{
int midPoint{getMidPoint(min, max)};
if(min > max)
return -1;
if(array[midPoint] == target)
return midPoint;
return (array[midPoint] > target) ? binarySearch(array, target, min, midPoint - 1) :
binarySearch(array, target, midPoint + 1, max);
}
```

0 Reply

sandersan

April 5, 2025 12:24 am PDT

```
int binarySearch(const int* array, int target, int min, int max) {
    // iterative version
    if (array[min] == target) return min;
    if (array[max] == target) return max;
    while (array[(min + max) / 2] != target && max != min + 1)
        (array[(min + max) / 2] > target) ? (max = (min + max) / 2) : (min = (min +
max) / 2);
    return (max == min + 1) ? -1 : ((min + max) / 2);

    // recursive version
    if (max == min + 1) {
        if (array[min] == target) return min;
        if (array[max] == target) return max;
        return -1;
    }
    int mid { (min + max) / 2 };
    return (array[mid] == target) ? mid : binarySearch(array, target, (array[mid] >
target) ? min : mid, (array[mid] > target) ? mid : max);
}
```

Last edited 3 months ago by sandersan

0 Reply

shygosh

March 11, 2025 11:48 am PDT

I always challenge myself to write a solution in the least possible line of code, easy to understand, and performant if possible XD.
Here's my solution, I hope it's not too complicated to understand lol:

```cpp
int binarySearchIterate(const int* array, int target, int min, int max) {
  while (max > min) {
    int mid{min + ((max - min) >> 1)};
    if (array[min] == target) return min;
    if (array[max] == target) return max;
    (array[mid] > target) ? max = mid - 1 : --max;
    (array[mid] < target) ? min = mid + 1 : ++min;
  }
  return -1;
}

int binarySearchRecurse(const int* array, int target, int min, int max) {
  if (min >= max) return -1;
  int mid{min + ((max - min) >> 1)};
  if (array[min] == target) return min;
  if (array[max] == target) return max;
  (array[mid] > target) ? max = mid - 1 : --max;
  (array[mid] < target) ? min = mid + 1 : ++min;
  return binarySearchRecurse(array, target, min, max);
}
```

Last edited 3 months ago by shygosh

0 Reply

abdu

February 5, 2025 10:49 pm PST

When i ues

```cpp
std::midpoint
```

the code don't compile when i run it on 2022 version but it's work fine on 2020 version

---

And this metho doesn't avoid overflow

`int midpoint{ min + ((max-min) / 2) }; // this way of calculating midpoint doesn't avoid overflow`

0 Reply

Alex Author

Reply to abdu [13] February 8, 2025 9:00 pm PST

Are you including the <numeric> header?

Regarding your second comment, it doesn't overflow when min and max are both positive values, which is true in this quiz question.

0 Reply

fhs

January 7, 2025 6:00 am PST

one comment: I am surprised that the array's upper bound (48) was not included in the test suite. Failing to match the boundary values with a binary search implementation is one of the most common mistakes.

For my implementation, I made two changes just to make things a bit more interesting.

1. changed the indices to be the std::size_t type to be more consistent with the standard.
2. changed the binary search failure sentinel to be array.size() (i.e., max+1) to be more consistent with the approach used by iterators.

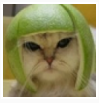
I also added the value 48 to the test suite.

Also since I am a lazy person and it takes work to generate a new project in my IDE, I merged the implementations and testing of the iterative and recursive binary searches into a single program. Just for grins, here is my implementation:

```cpp
//
//  main.cpp
//  test26_20.xa
//
//  Created by FHS on 1/6/25.
//  Modified from quiz in Chapter 20x question 3 to better match C++ typing
//  use std::size_t for indices;
//  binarySearch() failure changed to array.size() to be more consistent
//      with the approach used by iterators.

#include <iostream>
#include <iterator>

// array is the array to search over.
// target is the value we're trying to determine exists or not.
// min is the index of the lower bounds of the array we're searching.
// max is the index of the upper bounds of the array we're searching.
// binarySearch() should return the index of the target element if the target is
found, max+1 otherwise

using binSearchFunc = std::size_t(*)(const int *, int, std::size_t, std::size_t);

std::size_t binarySearch1(const int* array, int target, std::size_t min, std::size_t
max)
{ // iterative solution
    const std::size_t noMatch { max+1 };

    // check boundary conditions
    if (array[max] == target) { return max; }
    if (array[min] == target) { return min; }

    // search for match until there is no space left to check
    //  returns if match is found
    while (min+1 < max) {
        std::size_t midIndex = (min + (max - min)/2);
        if (array[midIndex] == target) { return midIndex; } // match found
        if (array[midIndex] < target) { min = midIndex; }
        else if (array[midIndex] > target) { max = midIndex; }
    }

    //  no match found, return error condition
    return noMatch;
}

std::size_t binarySearch2(const int* array, int target, std::size_t min, std::size_t
max)
{ // recursive solution
    const std::size_t noMatch { max+1 };
    std::size_t match { noMatch };

    std::size_t midIndex = (min + (max - min)/2);
    if (array[midIndex] == target) { return midIndex; }
    if (min < midIndex) {
        if (array[midIndex] > target) {
            match = binarySearch2( array, target, min, midIndex );
            if (midIndex+1 == match) { match = noMatch; }
        } else if (array[midIndex] < target) {
            match = binarySearch2( array, target, midIndex, max );
        }
    }

    if (noMatch == match) {
        // check boundary conditions
        if (array[max] == target) { return max; }
    }

    return match;
}

void testBinarySearch( binSearchFunc bsFunc,
```

```
                        const int array[], std::size_t arraySize,
                        const int testValues[], std::size_t numTestValues,
                        std::size_t expectedValues[] ) {

    // Loop through all of the test values
    for (std::size_t count{ 0 }; count < numTestValues; ++count)
    {
        // See if our test value is in the array
        std::size_t index{ (*bsFunc)(array, testValues[count], 0, arraySize - 1) };
        // If it matches our expected value, then great!
        if (index == expectedValues[count])
            std::cout << "test value " << testValues[count] << " passed!\n";
        else // otherwise, our binarySearch() function must be broken
            std::cout << "test value " << testValues[count] << " failed.  There's
something wrong with your code!\n";
    }

}

int main()
{
    constexpr int array[]{ 3, 6, 8, 12, 14, 17, 20, 21, 26, 32, 36, 37, 42, 44, 48 };
    constexpr std::size_t arraySize { std::size(array) };

    // We're going to test a bunch of values to see if they produce the expected
results
    constexpr std::size_t numTestValues{ 10 };
    // Here are the test values
    constexpr int testValues[numTestValues]{ 0, 3, 12, 13, 22, 26, 43, 44, 48, 49 };
    // And here are the expected results for each value
    std::size_t expectedValues[numTestValues]{
        arraySize, 0, 3, arraySize, arraySize, 8, arraySize, 13, 14, arraySize };

    std::cout << "Test with binarySearch1 (iterative solution)\n";
    testBinarySearch( binarySearch1, array, arraySize, testValues, numTestValues,
expectedValues );

    std::cout << "Test with binarySearch2 (recursive solution)\n";
    testBinarySearch( binarySearch2, array, arraySize, testValues, numTestValues,
expectedValues );

    return 0;
}
```

0 Reply



Alex Author

Reply to fhs [14] January 19, 2025 10:42 pm PST

Added a test case for 48 to the solution as well. Thanks for noting that this was missing!

Also made a few minor tweaks to way the arrays are defined.

0 Reply

Zendorr

December 14, 2024 11:26 pm PST

Figured it out this way

```cpp
int binarySearchIterative(const int* array, int target, int min, int max)
{
    int index{};
    while (true)
    {
        index = std::midpoint(min, max);
        if (array[index] == target)
            return index;
        else if (max - min <= 1)
            return -1;
        else if (array[index] > target)
            max = index;
        else if (array[index] < target)
            min = index;
    }
}

int binarySearchRecursive(const int* array, int target, int min, int max)
{
    int midpoint{ std::midpoint(min, max) };
    if (array[midpoint] == target)
        return midpoint;
    if (min >= max)
        return -1;
    if (array[midpoint] > target)
        return binarySearchRecursive(array, target, min, midpoint - 1);
    if (array[midpoint] < target)
        return binarySearchRecursive(array, target, midpoint + 1, max);
}
```

0 Reply

EmtyC

December 11, 2024 6:08 am PST

`Another chapter down! There's just this pesky quiz to get past…`

I NEED MORE QUIZZES

What do you think is the hardest quiz in the series (proly the next chapter's project)?

Also what do you think about a final project for the series (ofc this is not a demand I am just asking whether your open for this idea)

Last edited 6 months ago by EmtyC

0 Reply

Alex Author

Reply to EmtyC [15] December 16, 2024 12:38 pm PST

Hardest is probably next chapter's project.

A final project would be great -- ideally something that made use of inheritance, dynamic memory allocation, smart pointers, a custom container, move semantics, exceptions, and maybe serialization to disk and using a 3rd party library (like PDCurses). I've been thinking about some kind of grid-based fighting game where the user can move around and fight monsters. This would involve implementing a custom priority queue which holds dynamically allocated creatures sorted by who gets to move next,

and a Creature base class that has Player and Monster derived classes. A baby ASCII roguelike, perhaps.

2 Reply

Delici0us_

Reply to Alex [16] January 26, 2025 1:19 pm PST

If you are going for a console game, and like want to do not turn based stuff or so,
i've written a short, still very pretty bad console based window.
It runs on a set fps. Ive also made a short example.
https://github.com/Delici0u-s/finalQuizWindowFrame (https://github.com/Delici0u-s/finalQuizWindowFrame)[17]
It uses a modified Timer.h (which you provided in 18.4) and some console manipulation stuff.
If you seem interested feel free to use this. Although its still missing many features i recon it wont be very hard to implement. But if you wont its fine eatherway, idk if its really a good way of going at what you got planned.

0 Reply

Delici0us_

Reply to Delici0us_ [18] March 17, 2025 7:40 am PDT

I've removed this repository as its redacted nowadays. I did fully create a console engine. Like rendering and physics calculations. https://github.com/Delici0u-s/ConsoleRenderer This would be more suited.

0 Reply

EmtyC

Reply to Alex [16] December 16, 2024 2:40 pm PST

I could possibly help, when I finish I will probably do something like the project you presented, if everything went well I will provide the code (will probably be multi files).
Also I think that for a final project, the reader may just be given a description of the project with no help about the steps, and give the steps after as hints
(I may provide my suggestion on those too).
Although since I am a beginner my code or suggestions might not be the best.
(I would be happy to contribute to this site, I am pretty sad that the end is coming very fast 'I will miss this site a lot' :<)

0 Reply

dmigwi

September 15, 2024 7:22 am PDT

I have followed the argument that the array under consideration should be array[min : max] inclusive of the min position but exclusive of the max position. That is how I got the following code.

```
int binarySearch(const int* array, int target, int min, int max)
{
    if (max == min) // if they are equal there is no array to compare. i.e. array
length is zero.
        return -1;

    int midpoint { min + ((max - min)/2) };
    if (target > array[midpoint])
        return binarySearch(array, target, midpoint+1, max);
    else if (target < array[midpoint])
        return binarySearch(array, target, min, midpoint);
    else
        return midpoint;
}
```

with my code above: this `int index{ binarySearch(array, testValues[count], 0, static_cast<int>(std::size(array)) - 1) };` can be replaced with `int index{ binarySearch(array, testValues[count], 0, static_cast<int>(std::size(array))) };`

0 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. javascript:void(0)
3. https://en.cppreference.com/w/cpp/numeric/midpoint
4. https://en.cppreference.com/w/cpp/algorithm/binary_search
5. https://en.cppreference.com/w/cpp/algorithm/equal_range
6. https://www.learncpp.com/cpp-tutorial/introduction-to-operator-overloading/
7. https://www.learncpp.com/
8. https://www.learncpp.com/cpp-tutorial/lambda-captures/
9. https://www.learncpp.com/chapter-20-summary-and-quiz/
10. https://www.learncpp.com/cpp-tutorial/stdvector-resizing-and-capacity/
11. https://www.learncpp.com/cpp-tutorial/non-static-member-initialization/
12. https://gravatar.com/
13. https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/#comment-607437
14. https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/#comment-606357
15. https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/#comment-605125
16. https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/#comment-605315
17. https://github.com/Delici0u-s/finalQuizWindowFrame
18. https://www.learncpp.com/cpp-tutorial/chapter-20-summary-and-quiz/#comment-606984
19. https://g.ezoic.net/privacy/learncpp.com