Awesome GameDev Book

InfiniBrains Community Initiative

InfiniBrains

Table of contents

1. Awesome GameDev Resources

Estimated time to read: 15 minutes

> **Join is on Discord!**

How to use this repo: Read the topics, and if you're unsure if you understand the topics covered here it is a good time for you to revisit them.

Ways of reading:

- Website: read through your browser the interactive examples and animations will work better in this version;
- Github: You read through the github repo;
- PDF: download the latest release v1.8.7
- Amazon Kindle: You can buy the book in Amazon and read it in your kindle device;
- Contribute!: If you want to go deep and propose changes to repo, use the github repo.

1.1 Badges

CI: Documentation passing

Join us: Say Thanks chat 41 online stars 12.

Metrics: code quality B languages 4 todo counter 7 total lines invalid response data downloads 30 contributors 4 license MIT

Code of conduct: Contributor Covenant 2.1

1.2 Topics

1. Intro to Programming
2. Advanced Programming
3. Artificial Intelligence
4. Developer Portfolio

1.3 Philosophy

This repository aims to be practical, and it will be updated as we test the methodology. Frame it as a guidebook, not a manual. Most of the time, we are constrained by the time, so in order to move fast, we won't cover deeply some topics, but the basics that allows you to explore by yourself or point the directions for you to study in other places acting as a self-taught student, so you really should look for more information elsewhere if you feels so. I use lots of references and highly incentive you to look for other too and propose changes in this repo. Sometimes, it will mostly presented in a chaotic way, which implies that you will need to explore the concepts by yourself or read the manual/books. Every student should follow your own path to learning, it is impossible to cover every learning style, so it is up to you to build your own path and discover the best way to learn. What worked for me or what works for a given student probably won't work for you, so dont compare yourself to others too much, but be assured that we're here to help you to succeed. If you need help, just send private messages, or use public forums such as github issues and discussions.

1.4 Reflections on teaching and learning processes

1.4.1 Philosophies

I would like to categorize the classes into philosophies. so I can address them properly: - Advanced classes: are more focused on work and deliveries than theory, they are tailored toward the student goals more than the closed boxes and fixed expected results. It comprehends AI and Adv. AI; - Introduction classes: are focused on theory and practice. In those classes, they have more focus on structural knowledge and basic content. It comprehends classes such as Introduction to Programming. - Guidance: are more focused on how can we bring the student to the highest standard and get ready to be hired. It comprehends classes such as Capstone, Portfolio classes, and Mentoring activities.

1.4.2 Learning Styles

- Visual: You prefer using pictures, images, and spatial understanding;
- For this style I recently acquired a pen-tablet monitor, so I will be adding this type of content more often.
- I also use lots of diagrams via code2flow, sequence diagram and others
- I assume my handwriting is not the best, but I compensate it with lots of diagrams and pictures, and always project what I write in the computer.
- Aural: You prefer using sound and music;
- I always link to youtube videos and podcasts, so they can follow up with extra content and material;
- Verbal: You prefer using words, both in speech and writing;
- I setup my machine to record specific topics that might be hard to undestand in just one go, and I did some experimental recordings, but I am still struggling with video editing. I will be adding more videos in the future.
- My main issue here is that I am not a native english speaker, so I am still struggling with the language, but I am trying to improve it.
- Other issue that I can name is eye-to-eye contact. It feels overburned to me to keep eye-to-eye contact, that I usually look away.
- Physical: You prefer using your body, hands and sense of touch;
- Given my cultural origin, I am usually over expressive in this field, and I need more fine tuning my proxemic. Brazilians commonly talk and walk closer to each other than americans.
- While lecture I really enjoy to use my hands to express myself, and I am trying to use more body language to express myself.
- Logical: You prefer using logic, reasoning and systems;
- I always craft and test teaching experiences to push them to think and reason about the topics.
- I always use tools such as beecrowd to let them code and test their ability to solve problems.
- Social: You prefer to learn in groups or with other people;
- I incentive them to do in-class assignments in pairs, and do group assignments. But I recognize this might be a problem for some students, so I am trying to find a way to make it more inclusive.
- Strangelly for me, some students prefer to socialize with me by booking office hours more than working together. Probably next semester I will reserve a time to do a type of co-working time when I can be available to help them in their assignments.
- Solitary: You prefer to work alone and use self-study.
- Sometimes and some topics you really need to study by yourself, and it can be the best way for some. But I warn them about the effects of loneliness and impostor syndrome.
- This is usually the most common way to learn, and I always keep an eye on the ones that are struggling to keep up with the class. I always try to reach them and help them to keep up with the class.
- To compensate this solitude I incentive them to present their work to the class no they can experience having attention even when the lack social skills.

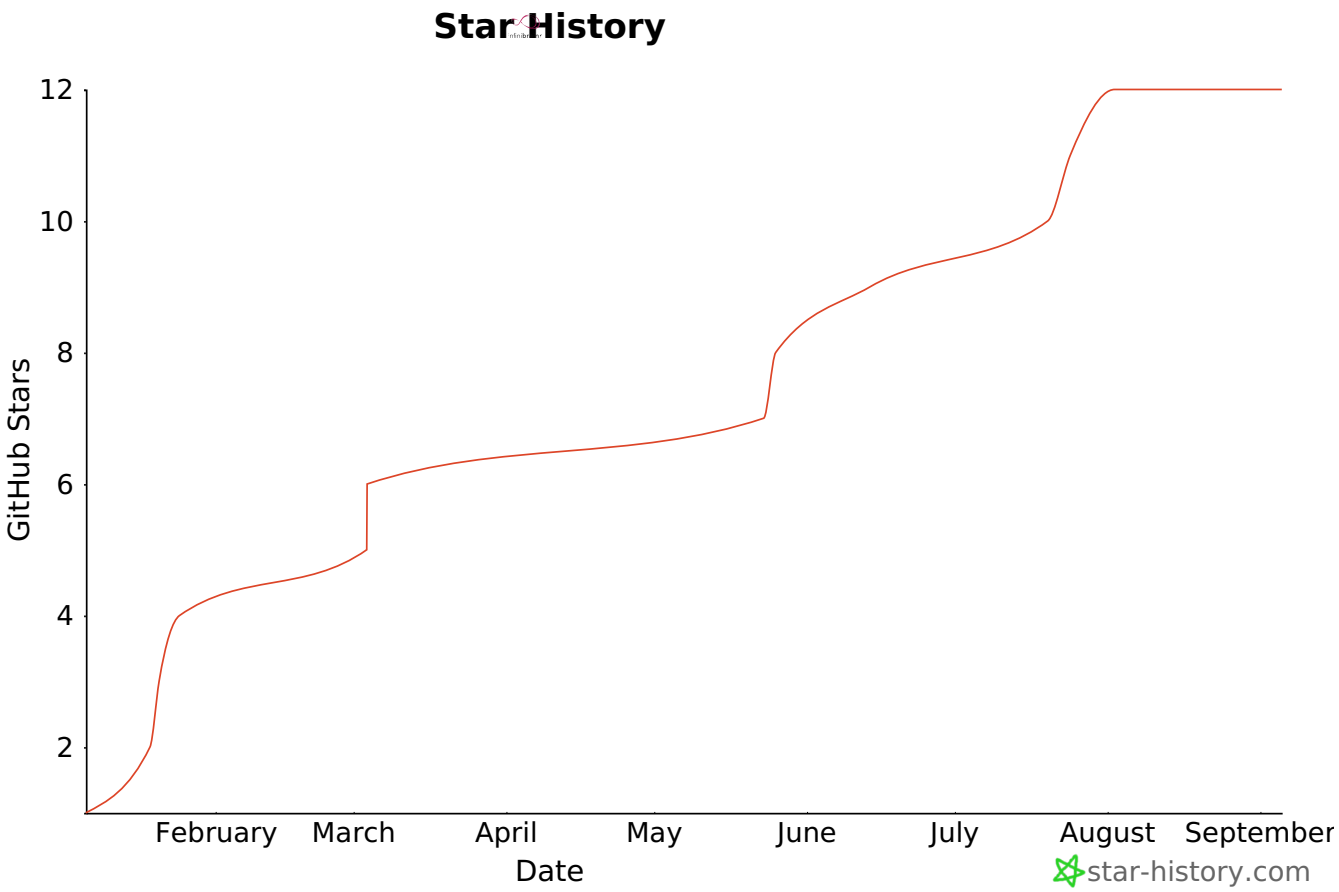
1.4.3 Teaching Styles

For every type of style, I try to give a bit of insights:

- Authoritative: control the classroom and maintain discipline;
- I create a set of rules that should be followed in order to guarantee the student's success;
- Delegator: give students control of their learning;
- For the intro classes I follow more this strategy;
- Facilitator: guide students and help them learn by themselves;
- I usually follow this strategy on advanced classes;
- Demonstrator: explain and show things to students;
- I usully provide a stream of references or even create my own content to show them how to do things;

2. Credits

Give us stars! Click -> stars 12



August 22, 2023

December 20, 2022

GitHub 

2.1 Comments

3. AI

3.1 Artificial Intelligence

Estimated time to read: 6 minutes

> **Warning**
>
> This is a work in progress, and the schedule is subject to change. Every change will be communicated in class. Use the github repo as the source of truth for the schedule and materials. The materials provided in canvas are just a copy for archiving purposes and might be outdated.

3.1.1 Schedule for Fall 2023

Relevant dates for the Fall 2023 semester:

- 09-10 Oct 2023 - Midterms Week
- 20-24 Nov 2023 - Thanksgiving Break
- 11-15 Dec 2023 - Finals Week

🔰 Introduction

- Week 1. 2023/08/28
- Topic: Introduction
- Formal Assignment: Flocking at Beecrowd
- Interactive Assignment: Flocking at MoBaGEn

🤖 Behavioral Agents

- Week 2. 2023/09/04
- Topic: Behavioral agents
- Formal Assignment: Flocking at Beecrowd
- Interactive Assignment: Flocking at MoBaGEn

▦ Finite Automata

- Week 3. 2023/09/11
- Topic: Automata Finite and 2D Grids
- Formal Assignment: Game of Life at Beecrowd
- Interactive Assignment: Game of Life at MoBaGEn

🔢 Random Numbers

- Week 4. Date: 2023/09/18
- Topic: Pseudo Random Number Generation
- Formal Assignment: PRNG at Beecrowd

🚶 DFS

- Week 5. 2023/09/25
- Topic: Depth First Search, Random walk, Maze Generation
- Formal Assignment: Maze at Beecrowd
- Interactive Assignment: Maze at Mobagen

🔍 Path finding

- Week 6. 2023/10/02
- Topic: Breadth First Search and Path Finding A*
- Interactive Assignment: Catch the Cat

⚠️ Midterms

- Week 7. Date: 2023/10/09
- Topic: Catch the Cat Challenge and Competition
- Catch the Cat

Spatial Quantization

- Week 8. 2023/10/16
- Topic: Spatial Quantization and Partitioning
- Readings: Spatial Quantization
- Formal Assignment: Hide and Seek

Spatial Quantization

- Week 9. 2023/10/23
- Topic: Spatial Quantization and Partitioning
- Readings: Spatial Quantization
- Formal Assignment: Hide and Seek

∿ Noise Functions

- Week 10. 2023/10/30
- Topic: Noise functions
- Formal Assignment:
- Interactive Assignment: Scenario Generation

Procedural Generation

- Week 11. 2023/11/06
- Topic: Procedural Content Generation - Scenario
- Formal Assignment:
- Interactive Assignment: Scenario Generation

Procedural Generation

- Week 12. 2023/11/13
- Topic: Procedural Content Generation - Scenario
- Formal Assignment:
- Interactive Assignment: Scenario Generation

Break

- Week 13. 2023/11/20
- Topic: BREAK. No classes

Work sessions

- Week 14. 2023/11/27
- Topic: Work sessions for final project

Work sessions

- Week 15. 2023/12/04
- Topic: Work sessions for final project

Finals

- Week 16. 2023/12/11
- Topic: Final Presentations

slide test: test

August 30, 2023

December 20, 2022

GitHub 

3.1.2 Comments

3.2 Space quantization

Estimated time to read: 21 minutes

Space quantization is a way to sample continuous space, and it can to be used in in many fields, such as Artificial Intelligence, Physics, Rendering, and more. Here we are going to focus primarily Spatial Quantization for AI, because it is the base for pathfinding, line of sight, field of view, and many other techniques.

Some of the most common techniques for space quantization are: grids, voxels, graphs, quadtrees, octrees, KD-trees, BSP, Spatial Hashing and more. Another notable techniques are line of sight(or field of view), map flooding, caching, and movement zones.

3.3 Grids

Grids are the most common technique for space quantization. It is a very simple technique, but it is very powerful. It consists in dividing the space in a grid of cells, and then we can use the cell coordinates to represent the space. The most common grid is the square grid, but we can use hexagonal and triangular grids, you might find some irregular shapes useful to exploit the space conformation better.

3.3.1 Square Grid

The square grid is a regular grid, where the cells are squares. It is very simple to implement and understand.

There are some ways to store data for squared grids. Arguably you could 2D arrays, arrays of arrays or vector of vectors, but depending on the way you implement it, it can hurt the performance. Example: if you use an array of arrays or vector of vectors, where every entry from de outer array is a pointer to the inner array, you will have a lot of cache misses, because the inner arrays are not contiguous in memory.

Notes on cache locality

So in order do increase data locality for squared grids, you can use a single array, and then use the following formula to calculate the index of the cell. We call this strategy matrix flattening.

```
int arrray[width * height]; // 1D array with the total size of the grid
int index = x + y * width; // index of the cell at x,y
```

There is a catch here, given we usually represent points as X and Y coordinates, we need to be careful with the order of the coordinates. While you are iterating over all the matrix, you need to iterate over the Y coordinate first, and then the X coordinate. This is because the Y coordinate is the one that changes the most, so it is better to have it in the inner loop. By doing that, you will have better cache locality and effectively the index will be sequential.

```
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        int index = x + y * width;
        // do something with the cell at index
    }
}
```

Quantization and dequantization of square grids

If your world is based on floats, you can use the square by using the floor function or just cast to integer type, because the default behavior of casting from float to integer is to floor it. Example: In the case of a quantization resolution of size of 1.0f, everything between 0 and 1 will be in the cell (0,0), everything between 1 and 2 will be in the cell (1,0), and so on.

```
Vector2int quantize(Vector2f position, float resolution) {
    return Vector2int((int)floor(position.x/resolution), (int)floor(position.y/resolution));
}
```

If you need to get the center of the cell in the world coordinates following the quantization resolution, you can use the following code.

```
Vector2f dequantize(Vector2int index, float resolution) {
    return Vector2f((float)index.x * resolution + resolution/2.0f, (float)index.y * resolution + resolution/2.0f);
}
```

If you need to get the corners of the cell following the quantization resolution, you can use the following code.

```
Rectangle2f cell_bounds(Vector2int index, float resolution) {
    return {index.x * resolution, index.y * resolution, (index.x+1) * resolution, (index.y+1) * resolution};
}
```

If you need to get the neighbors of a cell, you can use the following code.

```
std::vector<Vector2int> get_neighbors(Vector2int index) {
    return {{index.x-1, index.y}, {index.x, index.y-1},
            {index.x+1, index.y}, {index.x, index.y+1}};
}
```

3.3.2 Hexagonal Grid

Hexagonal grid is an extension of a square grid, but the cells are hexagons. It feels nicer to human eyes because we have more equally distant neighbors. If used as subtract for pathfinding, it can be more efficient because the path can be more straight.

It can be implemented as single dimension array, but you need to be careful with shift that happens in different odd or even indexes. You can use the following formula to calculate the index of the cell. In this world quantization can be in 4 conformations, depending on the rotation of the hexagon and the alignment of the first cell.

1. Point pointy top hexagon with first line aligned to the left:

```
 / \ / \ / \
| A | B | C |
 \ / \ / \ / \
  | D | E | F |
 / \ / \ / \ /
| G | H | I |
 \ / \ / \ /
```

2. Point pointy top hexagon with first line aligned to the right

```
   / \ / \ / \
  | A | B | C |
 / \ / \ / \ /
| D | E | F |
 \ / \ / \ / \
  | G | H | I |
   \ / \ / \ /
```

3. Flat top hexagon with first column aligned to the top:

```
 __   __
/A \__/C \
\__/B \__/
/D \__/F \
\__/E \__/
/G \__/I \
\__/H \__/
   \__/
```

4. Flat top hexagon with first column aligned to the bottom:

```
    __
 __/B \__
/A \__/C \
\__/E \__/
/D \__/F \
\__/H \__/
/G \__/I \
\__/  \__/
```

Quantization and dequantization of hexagonal grids

For simplicity, we are going to use the first conformation, where the first line is aligned to the left, and the hexagons are pointy top. The quantization is done by using the following formula.

```
// I am assuming that the hexagon is pointy top, and the first line is aligned to the left
// I am also assuming that the hexagon is centered in the cell, and the top left corner is at (0,0),
// y axis is pointing down and x axis is pointing right
// this dont work for all the cases, but it is a good approximation for locations near the center of the hexagon
/*
  / \ / \ / \
 | A | B | C |
  \ / \ / \ / \
   | D | E | F |
  / \ / \ / \ /
 | G | H | I |
  \ / \ / \ /
 */
Vector2int quantize(Vector2f position, float hexagonSide) {
    int y = (position.y - hexagonSide)/(hexagonSide * 2);
    int x = y%2==0 ?
      (position.x - hexagonSide * sqrt3over2) / (hexagonSide * sqrt3over2 * 2) : // even lines
      (position.x - hexagonSide * sqrt3over2 * 2)/(hexagonSide * sqrt3over2 * 2) // odd lines
    return Vector2int(x, y);
}
Vector2f dequantize(Vector2int index, float hexagonSide) {
    return Vector2f(index.y%2==0 ?
      hexagonSide * sqrt3over2 + index.x * hexagonSide * sqrt3over2 * 2 : // even lines
      hexagonSide * sqrt3over2 * 2 + index.x * hexagonSide * sqrt3over2 * 2, // odd lines
      hexagonSide + index.y * hexagonSide * 2);
}
```

You will have to figure out the formula for the other conformations. Or send a merge request to this repository adding more information.

3.4 Voxels and Grid 3D

Grids in 3D works the same way as in 2D, but you need to use 3D vectors/arrays or voxel volumes. Most concepts applies here. If you want to expand this section, send a merge request.

3.5 Quadtree

Quadtree is a tree data structure where each node has 4 children. It is used to partition a space in 2D. It is used to optimize collision detection, pathfinding, and other algorithms that need to iterate over a space. It is also used to optimize rendering, because you can render only the visible part of the space.

3.5.1 Quadtree implementation

Quadtree is a recursive data structure, so you can implement it using a recursive data structure. The following code is a simple implementation of a quadtree.

```
// this code is not tested, but it should work. It is just an example and send a merge request if you find any errors.
// node
template<class T>
struct DataAtPosition {
    Vector2f center;
    T data;
};

template<class T>
struct QuadtreeNode {
    Rectangle2f bounds;
    std::vector<DataAtPosition<T>> data;
    std::vector<QuadtreeNode<T>> children;
};

// insert
template<class T>
void insert(QuadtreeNode<T>& root, DataAtPosition<T> data) {
    if (root.children.empty()) {
        root.data.push_back(data);
        if (root.data.size() > 4) {
            root.children.resize(4);
            for (int i = 0; i < 4; ++i) {
                root.children[i].bounds = root.bounds;
            }
```

```
            root.children[0].bounds.max.x = root.bounds.center().x; // top left
            root.children[0].bounds.max.y = root.bounds.center().y; // top left
            root.children[1].bounds.min.x = root.bounds.center().x; // top right
            root.children[1].bounds.max.y = root.bounds.center().y; // top right
            root.children[2].bounds.min.x = root.bounds.center().x; // bottom right
            root.children[2].bounds.min.y = root.bounds.center().y; // bottom right
            root.children[3].bounds.max.x = root.bounds.center().x; // bottom left
            root.children[3].bounds.min.y = root.bounds.center().y; // bottom left
            for (auto& data : root.data) {
                insert(root, data);
            }
            root.data.clear();
        }
    } else {
        for (auto& child : root.children) {
            if (child.bounds.contains(data.center)) {
                insert(child, data);
                break;
            }
        }
    }
}

// query
template<class T>
void query(QuadtreeNode<T>& root, Rectangle2f bounds, std::vector<DataAtPosition<T>>& result) {
    if (root.bounds.intersects(bounds)) {
        for (auto& data : root.data) {
            if (bounds.contains(data.center)) {
                result.push_back(data);
            }
        }
        for (auto& child : root.children) {
            query(child, bounds, result);
        }
    }
}
```

3.5.2 Quadtree optimization

The quadtree is a recursive data structure, so it is not cache friendly. You can optimize it by using a flat array instead of a recursive data structure.

3.6 Octree

Section WiP. Send a merge request if you want to contribute.

3.7 KD-Tree

KD-Trees are a tree data structure that are used to partition a spaces in any dimension (2D, 3D, 4D, etc). They are used to optimize collision detection(Physics), pathfinding(AI), and other algorithms that need to iterate over a space. Also they are also used to optimize rendering, because you can render only the visible part of the space. Pay attention that KD-Trees are not the same as Quadtree and Octrees, even if they are similar.

In KD-trees, every node defines an orthogonal partition plan that alternate every deepening level of the tree. The partition plan is defined by a dimension, a value. The dimension is the axis that is used to partition the space, and the value is the position of the partition plan. The partition plan is orthogonal to the axis, so it is a line in 2D, a plane in 3D, and a hyperplane in 4D.

3.8 BSP Tree

BSP inherits almost all characteristics of KD-Trees, but it is not a tree data structure, it is a graph data structure. The main difference is to instead of being orthogonal you define the plane of the section. The plane is defined by a point and a normal. The normal is the direction of the plane, and the point is a point in the plane.

3.9 Spatial Hashing

Spatial hashing is a data structure that is used to partition a space. It consists in a hash table where the keys are the positions of the elements, and the values are the elements in buckets. It is very fast to insert and query elements. But it is not good for iteration, because it is not cache friendly.

Usually when you want to use a spatial hashing, you create hash functions for the bucket keys, there is no limit on how you do that, but you have to keep in mind that the hash functions have to be fast and have to be good for the distribution of the elements. Here is a good example of a hashing function for 2D vectors.

```
namespace std {
    template<>
    struct hash<Vector2f> {
        // I am assuming size_t is 64 bits and the float is 32 bits
        size_t operator()(const Vector2f& v) const {
            // get the bits of the float in a integer
            uint64_t x = *(uint64_t*)&v.x;
            uint64_t y = *(uint64_t*)&v.y;
            // mix the bits of the floats
            uint64_t hash = x & (y << 32);
            return hash;
        }
    };
}
```

Pay attention that the hashing function above generates collisions, so you have to use a data structure that can handle collisions. You will use datastructures like `unordered_map<Vector2D, unordered_set<DATATYPE>>` or `unordered_map<Vector2D, vector<DATATYPE>>`. The first one is better for insertion and query, but it is not cache friendly.

To avoid having one bucket per every possible position, you have to setup properly the dimension of the bucket, a good sugestion is to alwoys floor the position and have buckets dimension of 1.0f. That would be good enough for most cases.

August 30, 2023

January 18, 2023

GitHub 

3.9.1 Comments

3.10 Assignments

3.10.1 Setup the repos

Estimated time to read: 6 minutes

We are going to use the following repositories:

1. Read about Privacy and FERPA compliance here
2. This one, for in class coding assignments. https://github.com/InfiniBrains/Awesome-GameDev-Resources
3. MoBaGEn, for interactive assignments. https://github.com/InfiniBrains/mobagen

Types of coding assignments

There are two types of coding assignments:

1. Algorithm: Beecrowd - This is an automatic grading system, and I am still creating assignments for it. I will try my best to make it work through it. If it does not work, you could just submit the code on canvas and I will grade it manually;
2. Interactive: For the interactive assignments you can choose whatever Game Engine you like, but I recommend you to use the framework I created for you: MoBaGEn.

> **Danger**
>
> Under no circunstaces, you should make your algorithm solutions public. Be aware that I spend so much time creating them and it is hard to me to always create new assignments.

Code assignments

> **Inline**
>
> If you are a enrolled in a class that uses this material, you SHOULD use the institutional and internal git server to be FERPA compliant. If you want to use part of this assignments to build your portfolio I recommend you to use github and make the interactive assignment public. If you are just worried about privacy concerns, you can use a private repo on github.

1. Create an account on github.com or any `git` hosting on your preference;
2. Fork repos or duplicate the target repo on your account;

a. If you want to make it count as part of your portfolio, fork the repo follow this;

b. If you want to keep it private or be FERPA compliant, duplicate the repo following this;

3. Add my user to your repo to it with `read` role. My userid is `tolstenko` on github, for other options, talk with me in class. Follow this;
4. Send me a message on canvas with the link to your repo;

Development tools

I will be using CMake for the classes, but you can use whatever you want. Please read this to understand the C++ toolset.

Recordings

In all interactive assignmets, you will have to record a 5 minute video explaing your code. Use OBS or any software you prefer to record your screen while you explain your code. But for this one, just send me the video showing the repo and the repo invites sent to me.

Grading

20 points total:

- 5 Points – inviting me to the repos;
- 5 Points – sending me the video explaining the repo and the invite sent to me;
- 10 Points - Show the repos cloned on your computer;

August 31, 2023

August 22, 2023

GitHub 

Comments

3.10.2 Flocking agents behavior assignment

Estimated time to read: 17 minutes

You are in charge of implementing some functions to make some AI agents flock together in a game. After finishing it, you will be one step further to render it in a game engine, and start making reactive NPCs and enemies. You will learn all the basic concepts needed to code and customize your own AI behaviors.

What is flocking?

Flocking is a behavior that is observed in birds, fish and other animals that move in groups. It is a very simple behavior that can be implemented with a few lines of code. The idea is that each agent will try to move towards the center of mass of the group (cohesion), and will try to align its velocity with the average velocity of the group (AKA alignment). In addition, each agent will try to avoid collisions with other agents (AKA avoidance).

Formal Notation Review

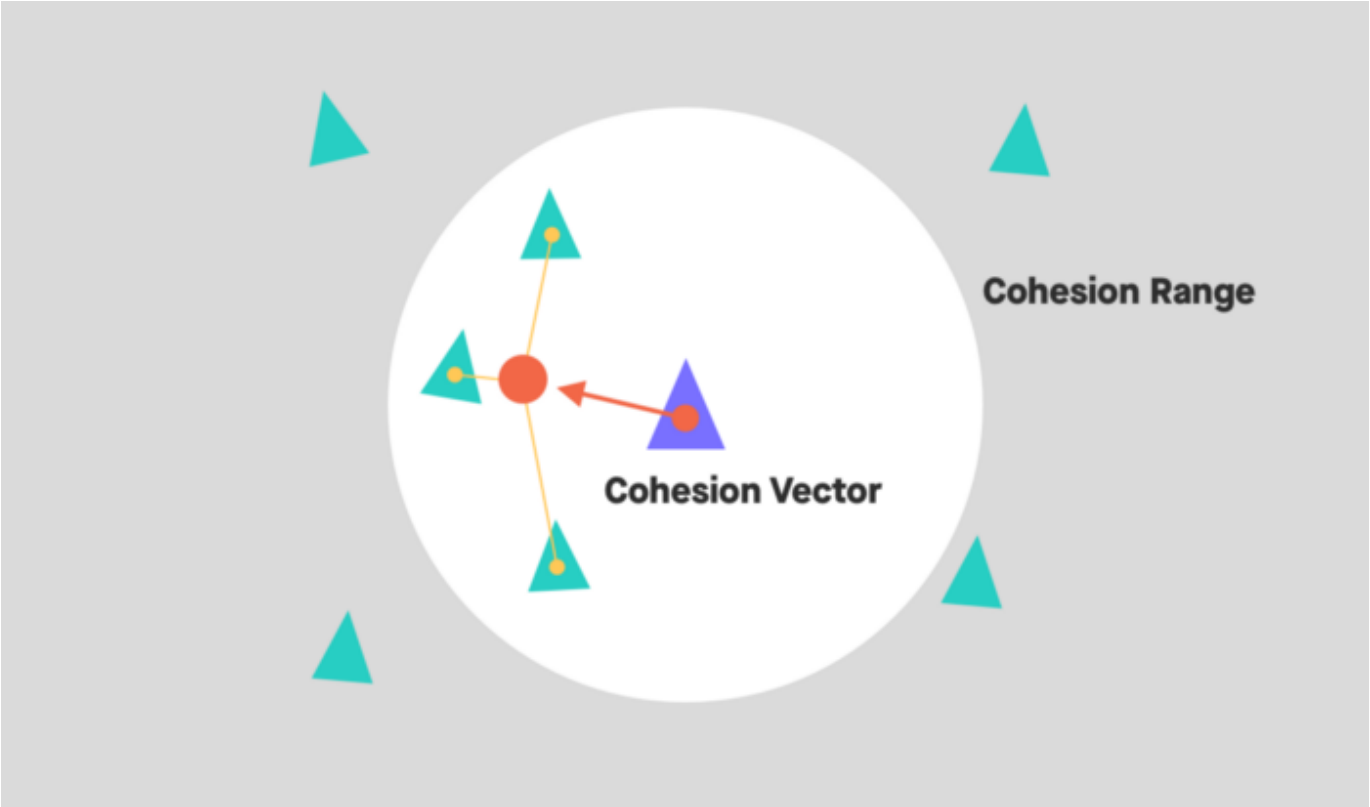
- $\vec{F}$ means a vector F that has components. In a 2 dimensional vector it will hold F_x and F_y. For example, if $F_x = 1$ and $F_y = 3$, then $\vec{F} = (1, 3)$
- Simple math operations between vectors are done component-wise. For example, if $\vec{F} = (1, 1)$ and $\vec{G} = (2, 2)$, then $\vec{F} + \vec{G} = (3, 3)$
- The notation $\overrightarrow{P_1P_2}$ means the vector that goes from P_1 to P_2. It is the same as $P_2 - P_1$
- The modulus notation means the length (magnitude) of the vector. $|\vec{F}| = \sqrt{F_x^2 + F_y^2}$ For example, if $\vec{F} = (1, 1)$, then $|\vec{F}| = \sqrt{2}$
- The hat ^ notation means the normalized vector(magnitude is 1) of the vector. $\hat{F} = \frac{\vec{F}}{|\vec{F}|}$ For example, if $\vec{F} = (1, 1)$, then $\hat{F} = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$
- The hat notation over 2 points means the normalized vector that goes from the first point to the second point. $\widehat{P_1P_2} = \frac{\overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_2}|}$ For example, if $P_1 = (0, 0)$ and $P_2 = (1, 1)$, then $\widehat{P_1P_2} = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$
- The sum $\sum$ notation means the sum of all elements in the list going from `0` to `n-1`. Ex. $\sum_{i=0}^{n-1} \vec{V}_i = \vec{V}_0 + \vec{V}_1 + \vec{V}_2 + \ldots + \vec{V_{n-1}}$

It is your job to implement those 3 behaviors following the ruleset below:

COHESION

Apply a force towards the center of mass of the group.

1. The n neighbors of an agent are all the other agents that are within a certain radius r_c(`<` operation) of the agent. It doesn't include the agent itself;
2. Compute the location of the center of mass of the group (P_{CM});
3. Compute the force that will move the agent towards the center of mass($\overrightarrow{F_c}$)
   The farther the agent is from the center of mass, the force increases linearly up to the limit of the cohesion radius r_c.



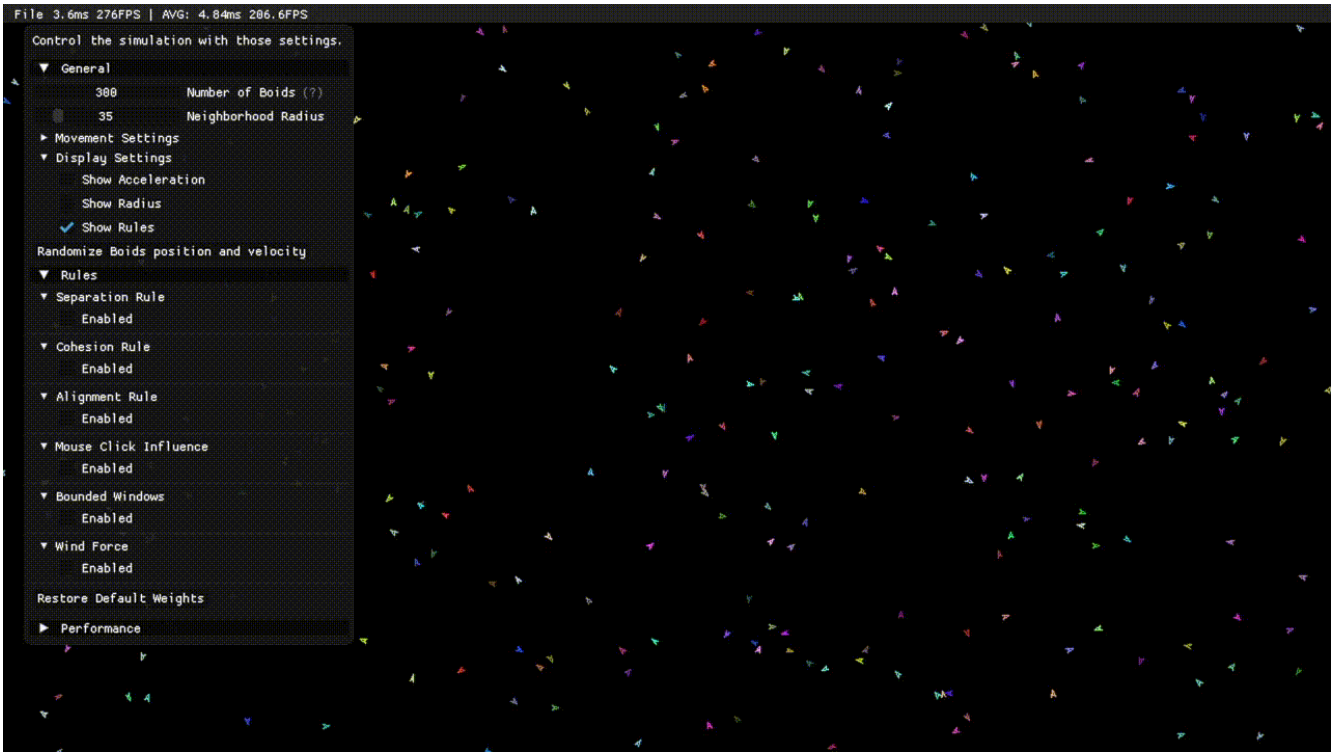
$$P_{CM} = \frac{\sum_{i=0}^{n-1} P_i}{n}$$

$\overrightarrow{F_c}$

Tip

Note that the maximum magnitude of
$\overrightarrow{F_c}$
is 1. Inclusive. This value can be multiplied by a constant K_c to increase or decrease the cohesion force to looks more appealing.

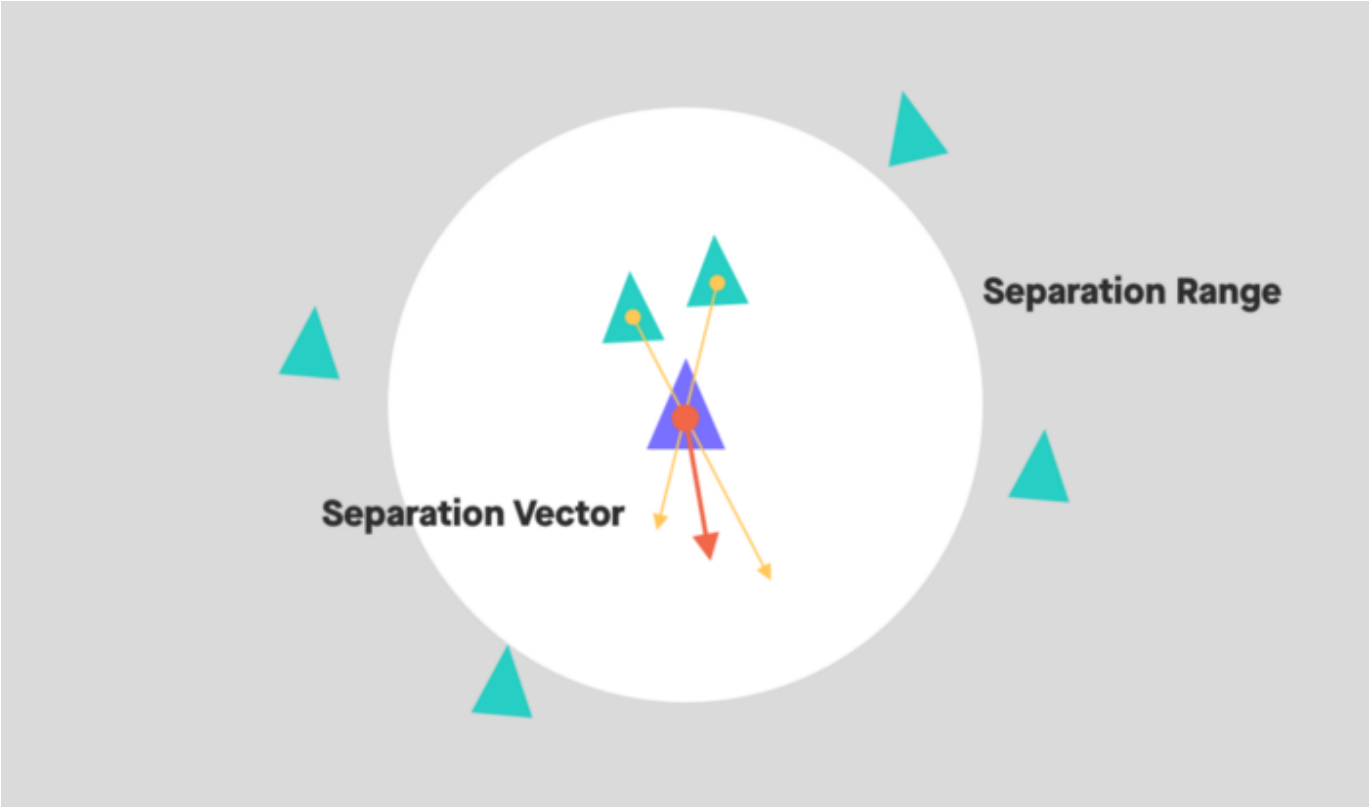
Cohesion Example



SEPARATION

It will move the agent away from other agents when they get too close.

1. The n neighbors of an agent are all the other agents that are within the separation radius r_s of the agent;
2. If the distance to a neighbor is less than the separation radius, then the agent will move away from it inversely proportionally to the distance between them.
3. Accumulate the forces that will move the agent away from each neighbor ($\overrightarrow{F_s}$).
   And then, clamp the force to a maximum value of F_{Smax}.



$\overrightarrow{F_s}$

Tip

Here you can see that if we have more than one neighbor and one of them is way too close, the force will be very high and make the influence of the other neighbors irrelevant. This is the expected behavior.

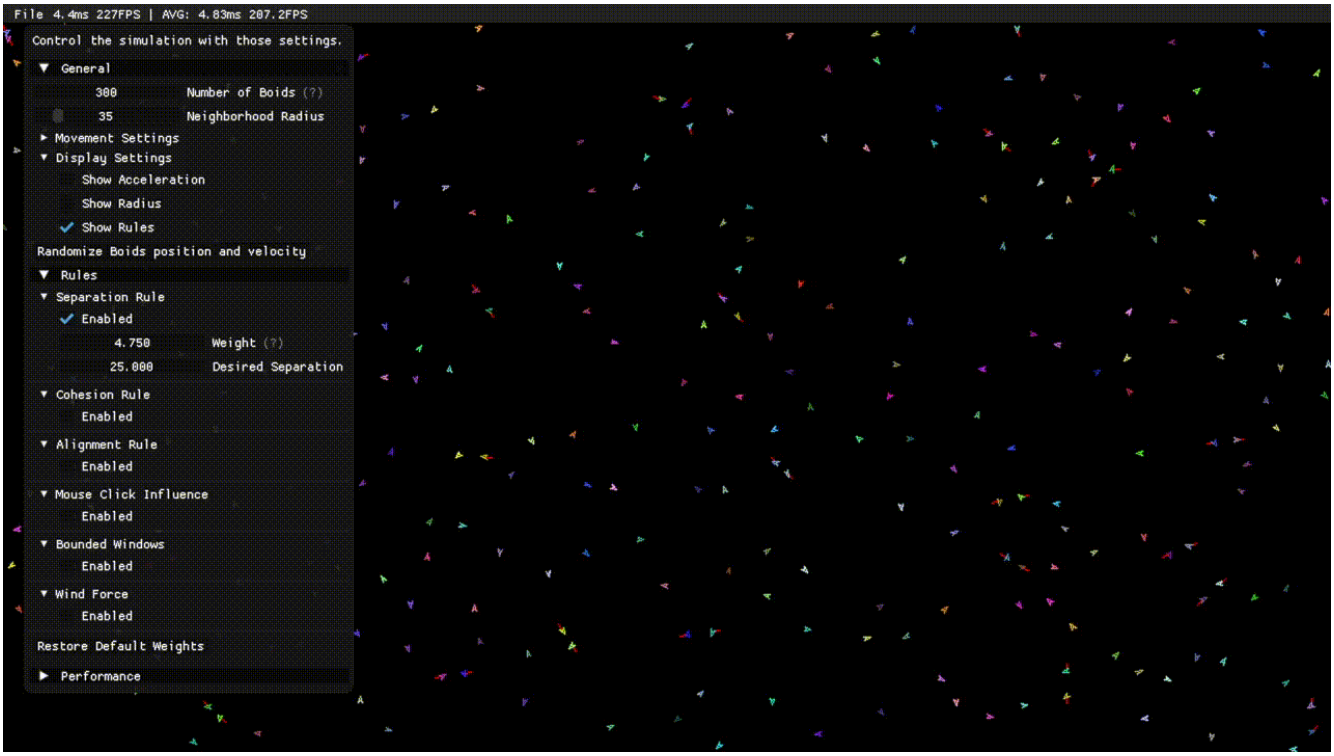
The force will go near infinite when the distance between the agent and the n neighbor is 0. To avoid this, after accumulating all the influences from every neighbor, the force will be clamped to a maximum magnitude of F_{Smax}.

$\overrightarrow{F_s}$

Tip

- You can implement those two math together, but it is better to isolate in two steps to make it easier to understand and debug.
- This is not an averaged force like the cohesion force, it is a sum of forces. So, the maximum magnitude of the force can be higher than 1.

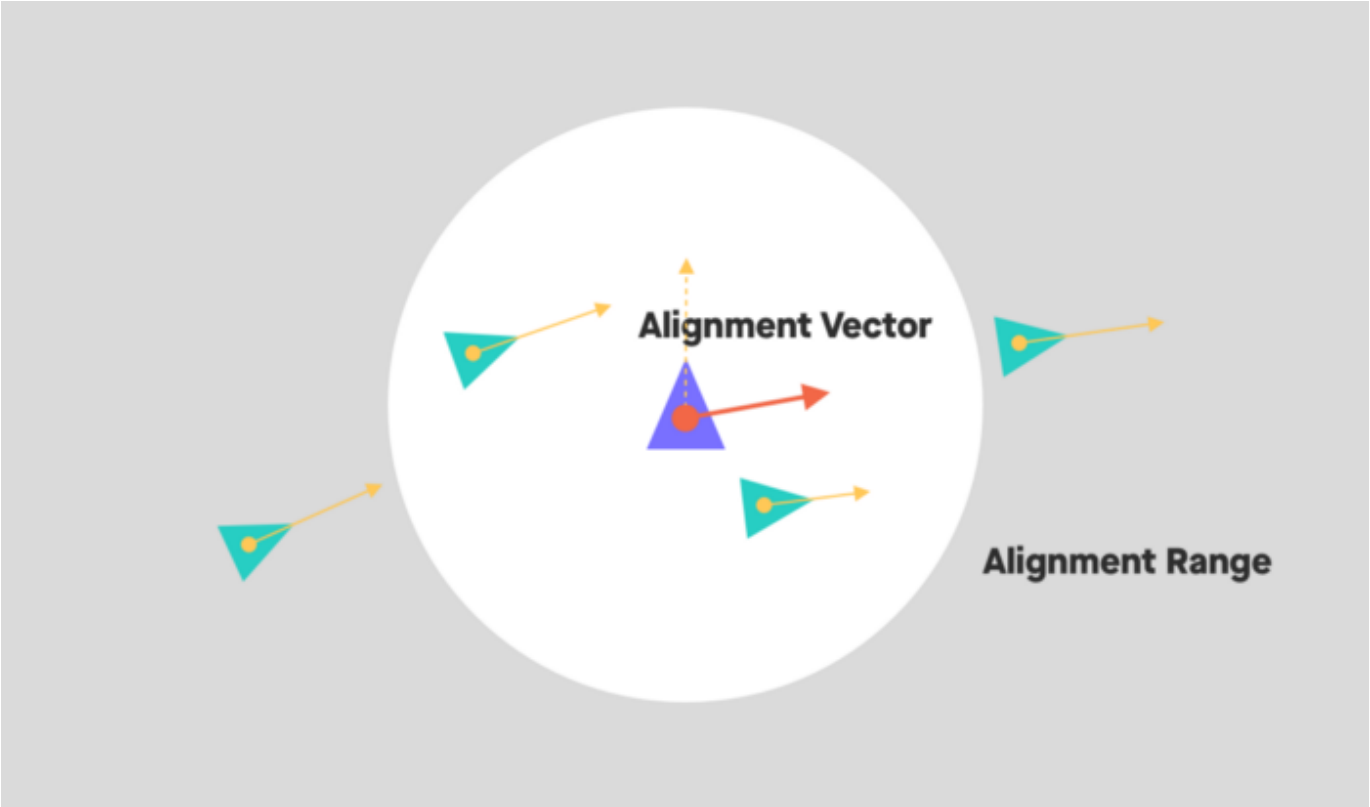
Separation Example



ALIGNMENT

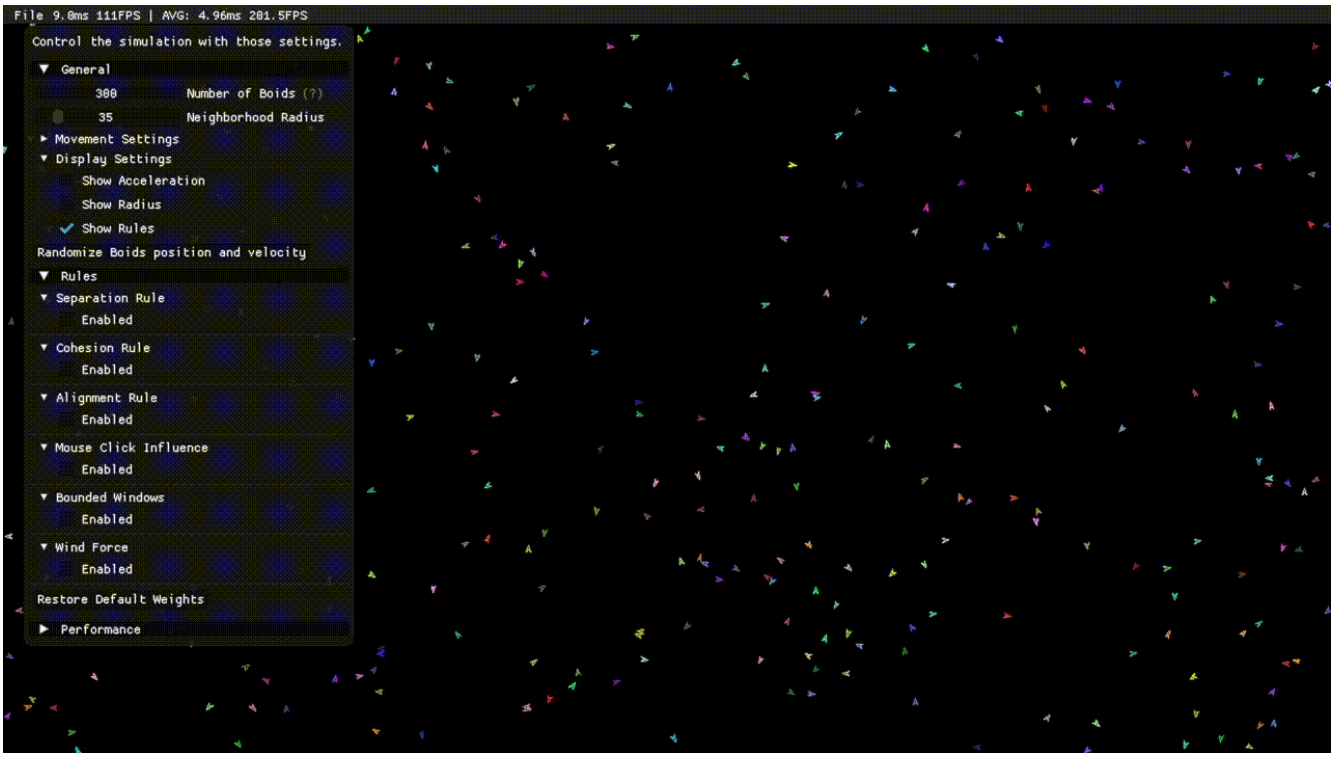
It is the force that will align the velocity of the agent with the average velocity of the group.

1. The n neighbors of an agent are all the agents that are within the alignment radius r_a of the agent, including itself;
2. Compute the average velocity of the group
   ); ($\overrightarrow{V_{avg}}$
3. Compute the force that will move the agent towards the average velocity
   ); ($\overrightarrow{F_a}$



$\overrightarrow{V_{avg}}$

Alignment Example



Behavior composition

The force composition is made by a weighted sum of the influences of those 3 behaviors. This is the way we are going to work, this is not the only way to do it, nor the more correct. It is just a way to do it.

- $\vec{F} = K_c \cdot \overrightarrow{F_c}$
  `This is a weighted sum!`
- $\overrightarrow{V_{new}}$
  `This is a simplification!`
- $P_{new} = P_{cur} + \overrightarrow{V_{new}}$
  `This is an approximation!`

> **Warning**
>
> A more precise way for representing the new position would be to use full equations of motion. But given timestep is usually very small and it even squared, it is acceptable to ignore it. But here they are anyway, just dont use them in this assignment:
>
> - $\overrightarrow{V_{new}}$
> - $P_{new} = P_{cur} + \overrightarrow{V_{cur}}$

Where:

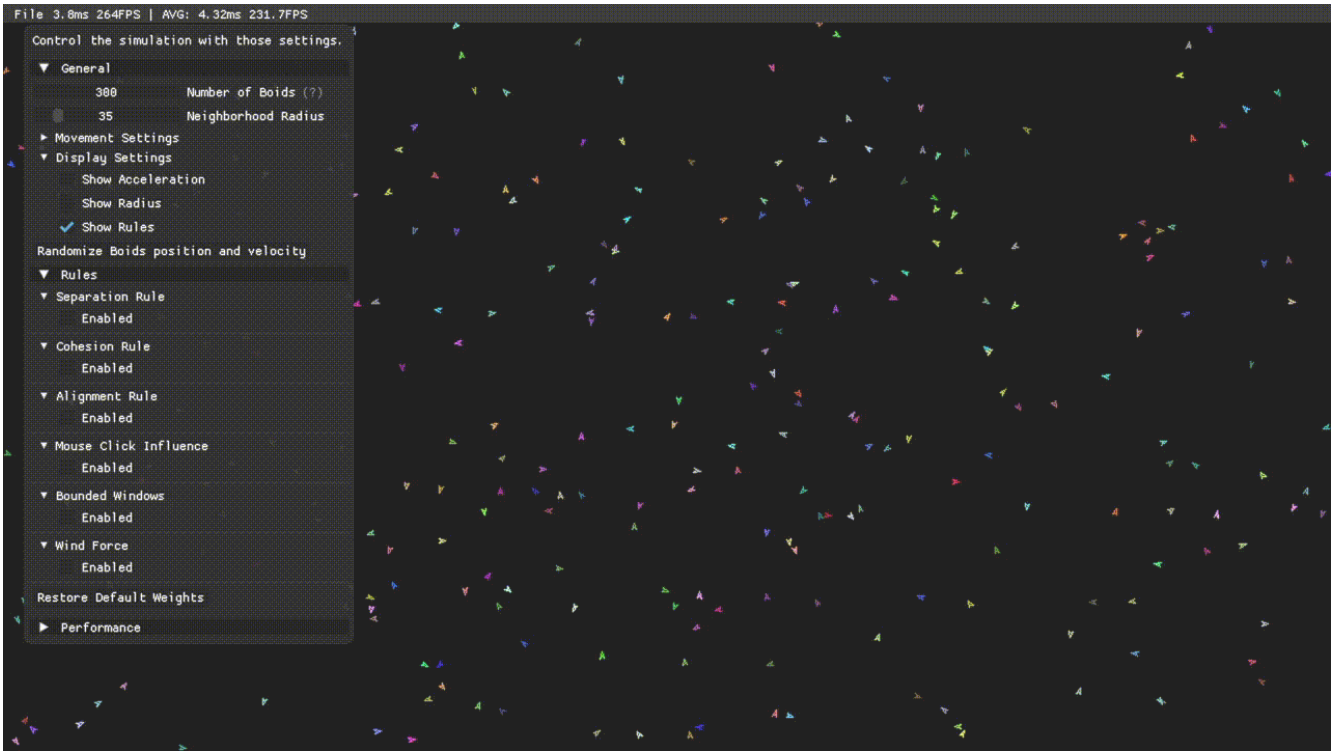
- $\overrightarrow{F}$ is the force applied to the agent;
- $\overrightarrow{V}$ is the velocity of the agent;
- P is the position of the agent;
- m is the mass of the agent, here it is always 1;
- Δt is the time frame (1/FPS);
- cur is the current value of the variable;
- new is the new value of the variable to be used in the next frame.

The $\overrightarrow{V_{new}}$ and P_{new} are the ones that will be used in the next frame and you will have to print to the console at the end of every single frame.

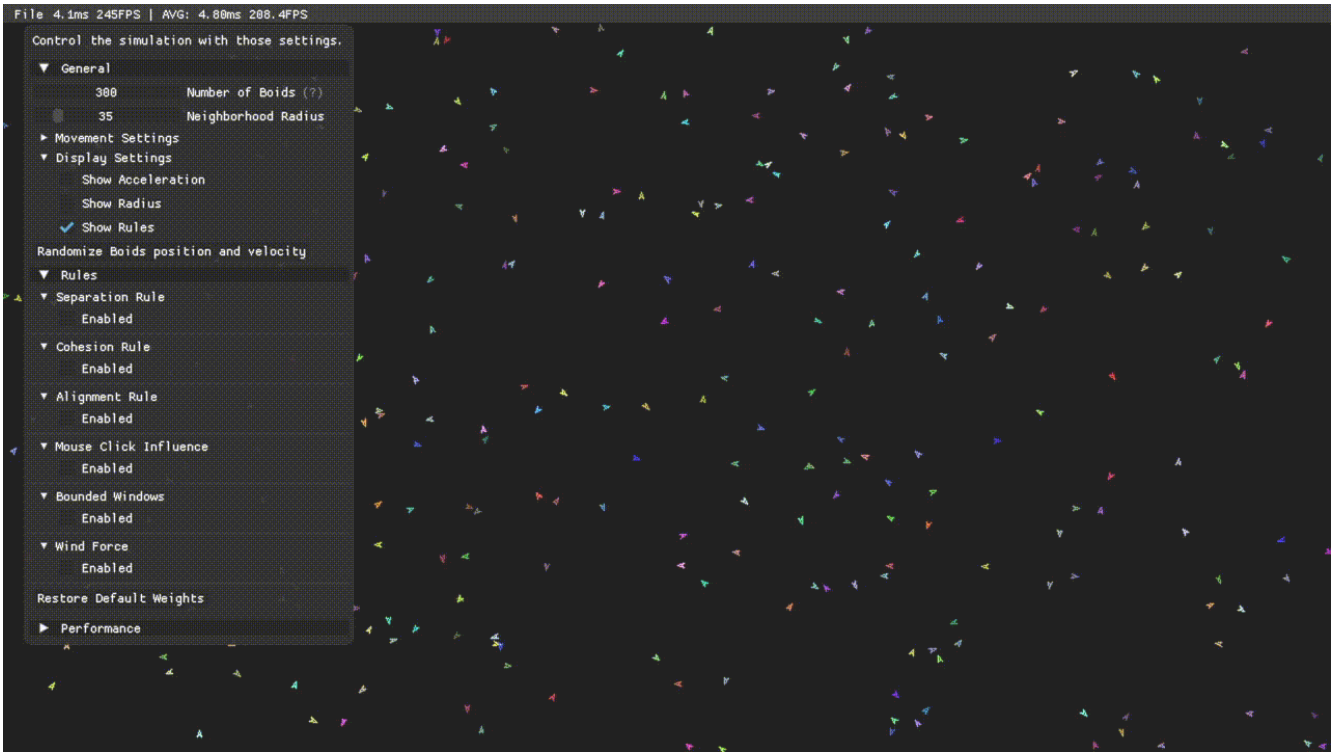
> **Note**
>
> - For simplicity, we are going to assume that the mass of all agents is 1.
> - In a real game simulation, it would be nice to apply some friction to the velocity of the agent to make it stop eventually or just clamp it to prevent the velocity get too high. But, for simplicity, we are going to ignore it.

Combined behavior examples

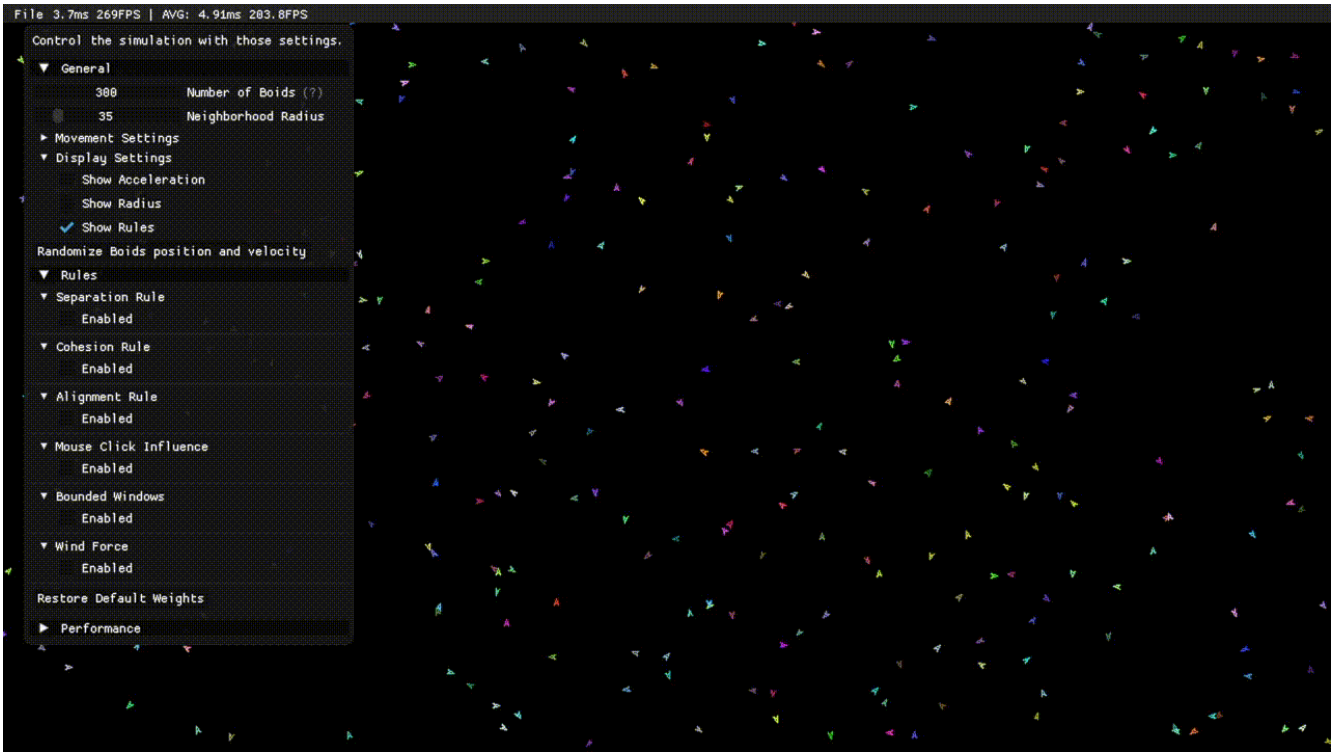
Alignment + Cohesion:



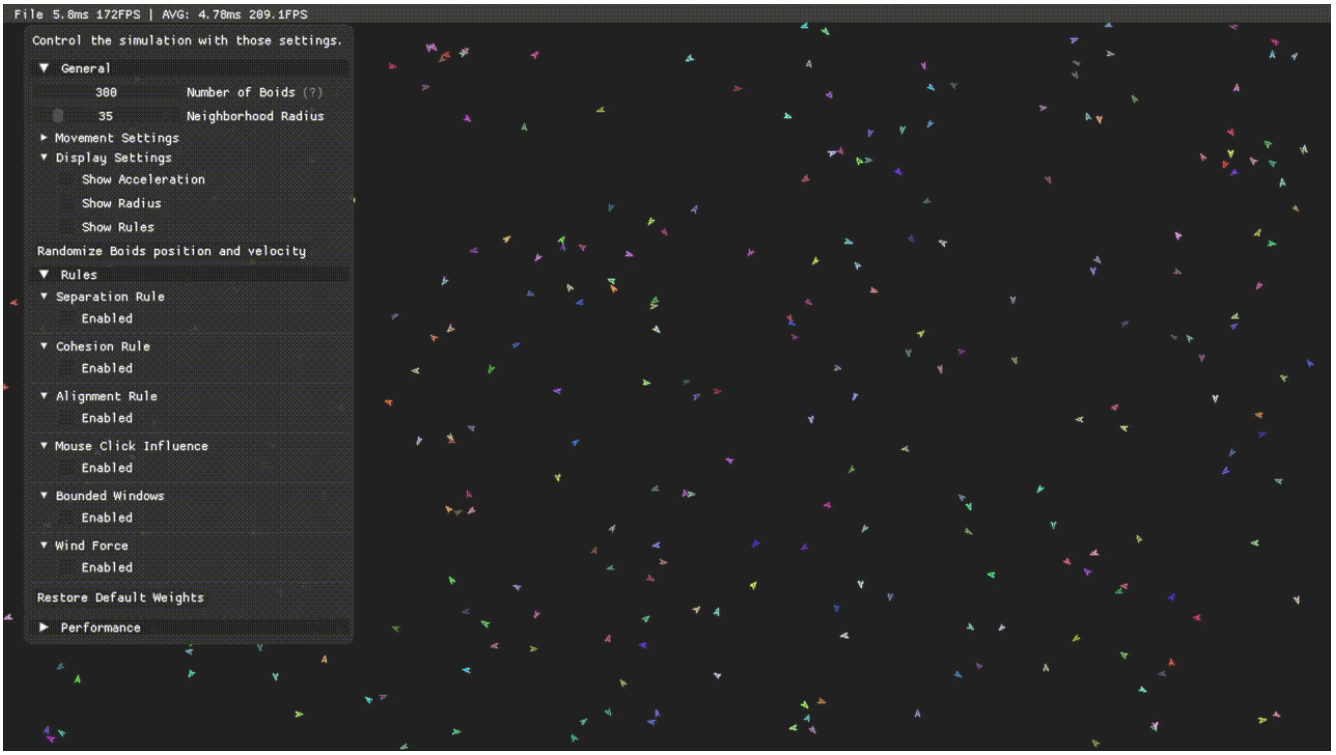
Separation + Cohesion:



Separation + Alignment:



All 3:



Input

The input consists in a list of parameters followed by a list of agents. The parameters are:

- r_c - Cohesion radius
- r_s - Separation radius
- F_{Smax} - Maximum separation force
- r_a - Alignment radius
- K_c - Cohesion constant
- K_s - Separation constant
- K_a - Alignment constant
- N - Number of agents

Every agent is represented by 4 values in the same line, separated by a space:

- x - X coordinate
- y - Y coordinate
- vx - X velocity
- vy - Y velocity

After reading the agent's data, the program should read the time frame (Δt), simulate the agents and then output the new position of the agents in the same sequence and format it was read. The program should keep reading the time frame and simulating the agents until the end of the input.

> **Data Types**
>
> All values are double precision floating point numbers to improve consistency between different languages.

INPUT EXAMPLE

In this example we are going to test only the cohesion behavior. The input is composed by the parameters and 2 agents.

```
1.000 0.000 0.000 0.000 1.000 0.000 0.000 2
0.000 0.500 0.000 0.000
0.000 -0.500 0.000 0.000
0.125
```

Output

The expected output is the position and velocity for each agent after the simulation step using the time frame. After printing each simulation step, the program should wait for the next time frame and then simulate the next step. All values should have exactly 3 decimal places and should be rounded to the nearest.

```
0.000 0.484 0.000 -0.125
0.000 -0.484 0.000 0.125
```

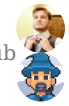
Grading

10 points total:

- 3 Points – by following standards;
- 2 Points – properly submitted in Canvas;
- 5 Points – passed on test cases;

September 5, 2023

July 13, 2023

GitHub 

Comments

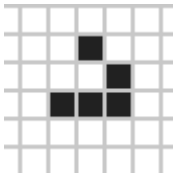
3.10.3 Game of Life

Estimated time to read: 3 minutes

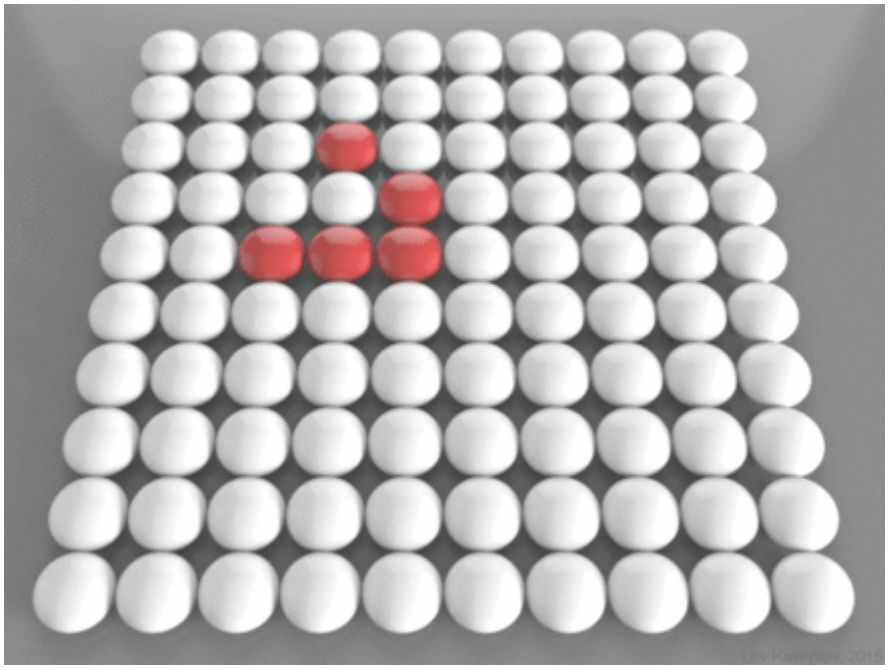
You are are applying for a internship position at Valvule Corp and they want to test your abilities to manage states. You were tasked to code the Conway's Game of Life.

The game consists in a C x L matrix of cells (Columns and Lines), where each cell can be either alive or dead. The game is played in turns, where each turn the state of the cells are updated according to the following rules:

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.



The map is continuous on every direction, so the cells on the edges have the cells on the opposite edge as neighbors. It is effectively a toroidal surface.



Input

The first line of the input are three numbers, C, L and T, the number of columns, lines and turns, respectively. The next L lines are the initial state of the cells, where each line has C characters, either `.` for dead cells or `#` for alive cells.

```
5 5 4
.#...
..#..
###..
.....
.....
```

Output

The output should be the state of the cells after T turns, in the same format as the input.

```
.....
..#..
...#.
.###.
.....
```

August 30, 2023

December 27, 2022

GitHub 

Comments

3.10.4 Pseudo Random Number Generation

Estimated time to read: 6 minutes

You are a game developer in charge to create a fast an reliable random number generator for a procedural content generation system. The requirements are:

- Do not rely on external libraries;
- Dont need to be cryptographically secure;
- Be blazing fast;
- Fully reproducible via automated tests if used the same seed;
- Use exactly 32 bits as seed;
- Be able to generate a number between a given range, both inclusive.

So you remembered a strange professor talking about the xorshift algorithm and decided it is good enough for your use case. and with some small research you found a the Marsaglia "Xorshift RNGs". You decided to implement it and test it.

XorShift

The xorshift is a family of pseudo random number generators created by George Marsaglia. The xorshift is a very simple algorithm that is very fast and have a good statistical quality. It is a very good choice for games and simulations.

`xorshift` is the process of shifting a number and then `xor`'ing it to the original value to create a new value.

```
value = value xor (value shift by number)
```

The shift operators can be to the left `<<` or to the right `>>`. When shifted to the left, it is the same thing as multiplying by 2 at the power of the number. When shifted to the right, it is the same thing as dividing.

The `xorshift` algorithm from Marsaglia is a combination of 3 `xorshifts`:

1. `xorshift` the seed by `13` bits to the left;
2. `xorshift` the seed by `17` bits to the right;
3. `xorshift` the seed by `5` bits to the left;

At the end of this 3 `xorshifts`, the current state of the seed is your current random number.

In order to clamp a random number the value between two numbers (max and min), you should follow this idea:

```
value = min + (random % (max - min + 1))
```

Input

Receives the seed `S`, the number `N` of random numbers to be generated and the range `R1` and `R2` of the numbers should be in, there is no guarantee the range numbers are in order. The range numbers are both inclusive. `S` and `N` are both `32 bits` unsigned integers and `R1` and `R2` are both `32 bits` signed integers.

```
1 1 0 99
```

Output

The list of numbers to be generated, one per line. In this case, it would be only one and the random number should be clamped to be between 0 and 99.

```
seed in decimal:       1
seed in binary:        0b00000000000000000000000000000001

seed:                  0b00000000000000000000000000000001
seed << 13:            0b00000000000000000010000000000000
```

```
seed xor (seed << 13): 0b00000000000000000010000000000001

seed:                  0b00000000000000000010000000000001
seed >> 17:            0b00000000000000000000000000000000
seed xor (seed >> 17): 0b00000000000000000010000000000001

seed:                  0b00000000000000000010000000000001
seed << 5:             0b00000000000000100000000000100000
seed xor (seed << 5):  0b00000000000000100001000000100001

The final result is 0b00000000000001000010000000100001 which is 270369 in decimal.
```

Now in order to clamp it to be between 0 and 99, we do:

```
value = min + (random % (max - min + 1))
value = 0 + (270369 % (99 - 0 + 1))
value = 0 + (270369 % 100)
value = 0 + 69
value = 69
```

So this output would be:

```
69
```

August 30, 2023

July 18, 2023

GitHub 

Comments

3.10.5 Maze generation via Depth First Search

Estimated time to read: 11 minutes

You are in charge of implementing a new maze generator for a procedurally generated game. The game is a 2D top-down game, where every level is composed by squared rooms blocked by walls. The rooms are generated by a maze generator, and the walls can be removed to create paths.

There are many ways to implement a maze generation and one of the most common is the Depth First Search algorithm combined with a Random Walk. The algorithm is simple and can be implemented in a recursive or interactive way. The suggested algorithm is as follows:

1. All walls are up;
2. Add the top left cell to the stack;
3. While the stack is not empty:

a. If the stack top cell has visitable neighbor(s):

i. Mark the top cell as visited;

ii. List visitable neighbors;

iii. Choose a neighbor (see below);

iv. Remove the wall between the cell and the neighbor;

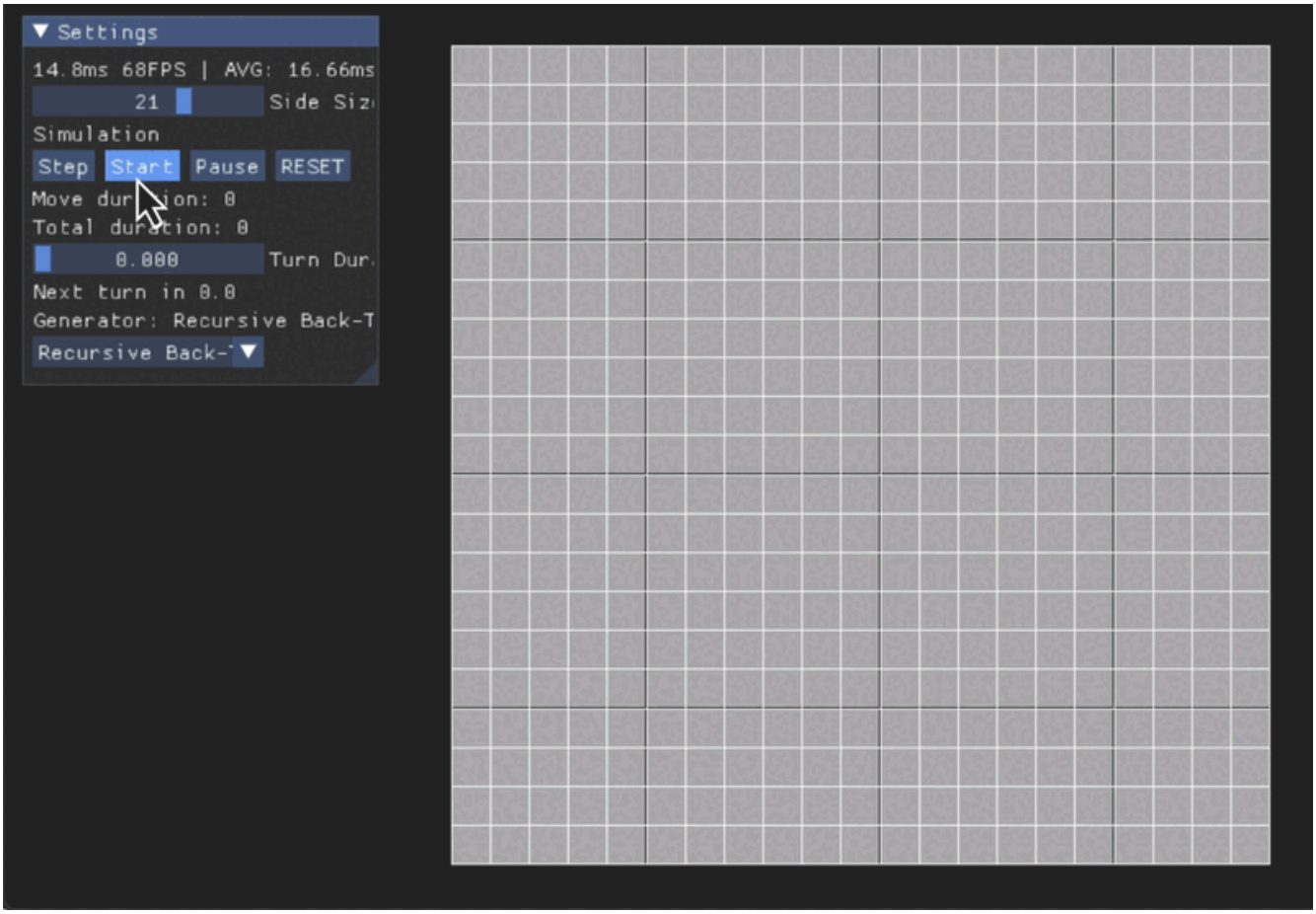
v. Add the neighbor to the stack;

b. Else:

i. Remove the top cell from the stack, backtracking;

Simulation

If you simulate the algorithm visually, the result would be something similar to the following



Random Number Generation

In order to be consistent with all languages and random functions the pseudo random number generation should follow the following sequence of 100 numbers:

```
[72, 99, 56, 34, 43, 62, 31, 4, 70, 22, 6, 65, 96, 71, 29, 9, 98, 41, 90, 7, 30, 3, 97, 49, 63, 88, 47, 82, 91, 54, 74, 2, 86, 14, 58, 35, 89, 11, 10, 60, 28,
21, 52, 50, 55, 69, 76, 94, 23, 66, 15, 57, 44, 18, 67, 5, 24, 33, 77, 53, 51, 59, 20, 42, 80, 61, 1, 0, 38, 64, 45, 92, 46, 79, 93, 95, 37, 40, 83, 13, 12,
78, 75, 73, 84, 81, 8, 32, 27, 19, 87, 85, 16, 25, 17, 68, 26, 39, 48, 36];
```

Every call to the random function should return the current number the index is pointing to, and then increment the index. If the index is greater than 99, it should be reset to 0.

Direction decision making

In order to give consistency on how to decide the direction of the next cell, the following procedure should be followed:

1. List all visitable neighbors of the current cell;
2. Sort the list of visitable neighbors by clockwise order, starting from the top neighbor: UP, RIGHT, DOWN, LEFT;
3. If there is one visitable, do not call random, just return the first neighbor found;
4. If there are two or more visitable neighbors, call random and return the neighbor at the index of the random number modulo the number of visitable neighbors. `vec[i]%visitableCount`

Input

The input is a single line with three `32 bits` unsigned integer numbers, `C`, `L` and `I`, where `C` and `L` are the number of columns and lines of the maze, respectively, and `I` is the index of the first random number to be used> `I` can varies from `0` to `99`.

```
2 2 0
```

In this case, our map will have `2` columns, `2` lines and the first random number to be used is the first one, `72` because it is pointed by the index `0`.

Output

Every line is a combination of underscore `_`, pipe `|` and empty ` ` characters. The `_` character represents a horizontal wall and the `|` character represents a vertical wall.

The initial state of the 2 x 2 map is:

```
 _ _
|_|_|
|_|_|
```

In order to interactively solve this, we will add `(0,0)` to the queue.

The neighbors of the current top (0,0) are RIGHT and DOWN, `(0,1)` and `(1,0)` respectively.

Following the clockwise order, the sorted neighbor list will be `[(0,1), (1,0)]`.

We have more than one neighbor, so we call random. The current random index is `0`, so the random number is `72` and we increment the index.

The random number is `72` and the number of neighbors is `2`, so the index of the neighbor to be chosen is `72 % 2 = 0`, so we choose the neighbor `(0,1)`, the RIGHT one.

The wall between `(0,0)` and `(0,1)` is removed, and `(0,1)` is added to the queue. Now it holds `[(0,0), (0,1)]`. The map is now:

```
 _ _
|_ _|
|_|_|
```

Now the only neighbor of (0,1) is DOWN, (1,1). So no need to call random, we just choose the only neighbor.

The wall between `(0,1)` and `(1,1)` is removed, and `(1,1)` is added to the queue. Now it holds `[(0,0), (0,1), (1,1)]`. The map is now:

```
 _ _
|_  |
|_|_|
```

Now the only neighbor of `(1,1)` is LEFT, `(1,0)`. So no need to call random, we just choose the only neighbor.

The wall between (1,1) and (1,0) is removed, and (1,0) is added to the queue. Now it holds `[(0,0), (0,1), (1,1), (1,0)]`. The map is now:

```
 _ _
|_  |
|_ _|
```

Now, the current top of the queue is `(1,0)` and there isn't any neighbor to be visited, so we remove the current top `(1,0)` from the queue and backtrack. The queue is now `[(0,0), (0,1), (1,1)]`.

The current top is `(1,1)` and there isn't any neighbor to be visited, so we remove `(1,1)` from the queue and backtrack. The queue is now `[(0,0), (0,1)]`.

The current top is `(0,1)` and there isn't any neighbor to be visited, so we remove `(0,1)` from the queue and backtrack. The queue is now `[(0,0)]`.

The current top is `(0,0)` and there isn't any neighbor to be visited, so we remove `(0,0)` from the queue and backtrack. The queue is now empty and we finish priting the map state. The final map is:

```
 _ _
|_  |
|_ _|
```

And this the only one that should be printed. No intermediary maps should be printed.

Example 1

INPUT 1

```
3 3 0
```

OUTPUT 1

```
 _ _ _
|_  | |
|  _| |
|_ _ _|
```

Example 2

INPUT 2

```
3 3 1
```

OUTPUT2

```
 _ _ _
| |_  |
|_ _  |
|_ _ _|
```

August 30, 2023

July 19, 2023

GitHub 

Comments

3.10.6 Catch the Cat

Estimated time to read: 1 minute

August 30, 2023

December 27, 2022

GitHub 

Comments

4. Intro. Programming

4.1 Intro to Programming

Estimated time to read: 7 minutes

4.1.1 Learning Objectives

- Understand the fundamental concepts of programming and computer science;
- Practice how to solve problems programatically using C++;
- Use tools to write and compile C++ programs;
- Code, document, test, and implement a well-structured, robust computer program using the C++ programming language.
- Write reusable modules (collections of functions).
- Use version control to manage your code;
- Use the debugger to find and fix bugs in your code;
- Understand the basics of file input/output;
- Work in groups to solve problems;

4.1.2 Learning Outcomes

- Be able to understand computer science concepts and terminology;
- To describe the basic components of a computer system and their functions;
- Differentiate between the various types of programming languages;
- To describe and use software tools in the programming process;
- Use modern concepts and principles of C++ programming language;
- To design, code, test, and debug a computer program using the C++ programming language;
- To demonstrate an understanding of primitive data types, values, operators and expressions in C/C++;
- Manage and manipulate files in C++;
- Deliver a full working project collaboratively;

4.1.3 Schedule

Warning

This is a work in progress, and the schedule is subject to change. Every change will be communicated in class. Use the github repo as the source of truth for the schedule and materials. The materials provided in canvas are just a copy for archiving purposes and might be outdated.

Relevant dates for the Fall 2023 semester:

- 09-13 Oct 2023 - Midterms Week
- 20-24 Nov 2023 - Thanksgiving Break
- 11-15 Dec 2023 - Finals Week

Week	Date	Topic
1	2023/08/28	1. Introduction, 2. Tools for first Program
2	2023/09/04	Data Types, Arithmetic Operations, Type conversion
3	2023/09/11	Conditionals, Boolean and Bitwise Operations
4	2023/09/18	Loops, for, while, goto and debugging
5	2023/09/25	Functions, Base Conversion, Pointers, Reference
6	2023/10/02	Streams, File IO
7	2023/10/09	Midterm
8	2023/10/16	Arrays, Vectors, String
9	2023/10/23	Recursion
10	2023/10/30	Sorting
11	2023/11/06	Structs, Unions, Enumerations
12	2023/11/13	Work sessions
13	2023/11/20	Thanks giving week
14	2023/11/27	Work sessions / Review
15	2023/12/04	Review / Presentations
16	2023/12/11	Finals

4.1.4 References

10^{th} edition Gaddis, T. (2020) Starting out with C++. Early objects / Tony Gaddis, Judy Walters, Godfrey Muganda. Pearson Education, Inc. Available at: https://research-ebsco-com.cobalt.champlain.edu/linkprocessor/plink?id=047f7203-3c9c-399b-834f-42cdaac4c1da

9th edition Gaddis, T. (2017) Starting out with C++. Early objects / Tony Gaddis, Judy Walters, Godfrey Muganda. Pearson. Available at: https://discovery-ebsco-com.cobalt.champlain.edu/linkprocessor/plink?id=502e29d6-3b46-38ff-9dc2-65e79c81c29b

- C++ early objects. Amazon (Champlain: 9th 10th)
- Modern C++ Programming
- learncpp
- cpluslus
- cprogramming
- programming-books
- google style guide
- riptutorial
- cpp manual
- cpp core guidelines
- rooksguide
- cpp best practices

September 1, 2023

December 20, 2022

GitHub 

4.1.5 Comments

4.2 Reasons why you should learn how to program with C++

Estimated time to read: 26 minutes

Before we start, this repository aims to be practical, and I highly incentive you to look for other references. I want to add this another awesome repository it holds an awesome compilation of modern C++ concepts.

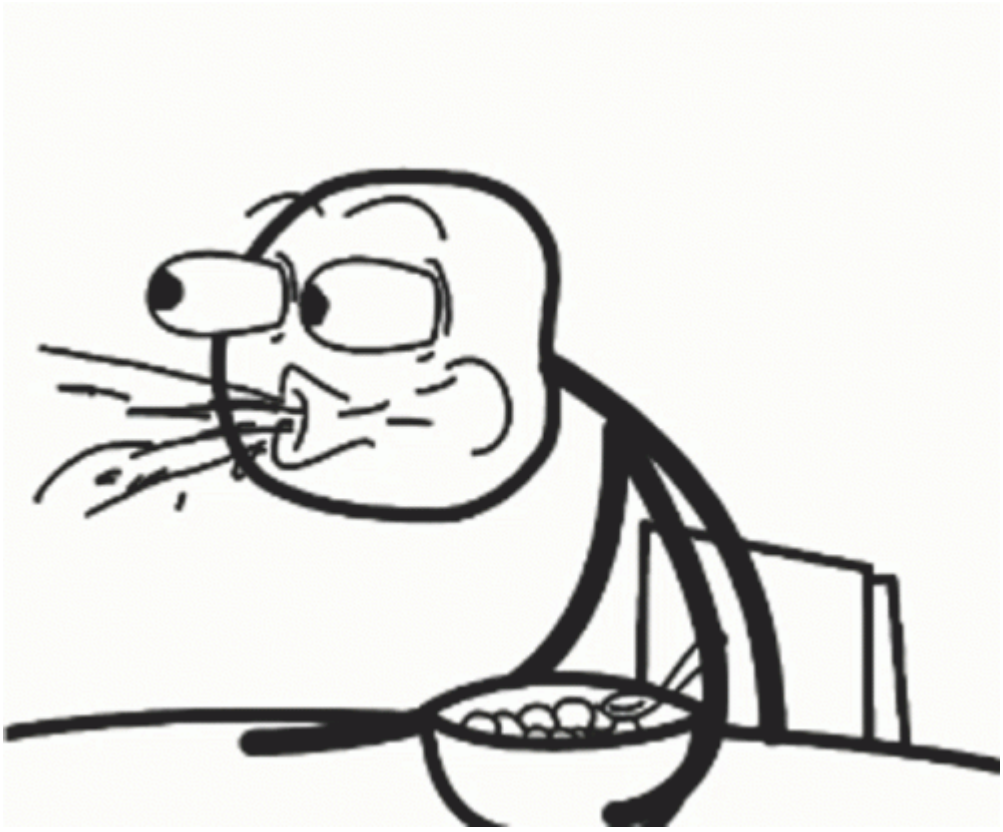
Another relevant reference for what we are going to cover is the updated core guidelines that explain why some syntax or style is bad and what you should be doing instead.

4.2.1 Why?

The first thing when you think of becoming a programmer is **HOW DO I START?** Well, **C++** is one of the best programming languages to give you insights into how a computer works. Through the process of learning how to code C++, you will learn not only how to use this language as a tool to solve your problems, but the farther you go, the more you will start uncovering and **exploring exciting computer concepts**.

C++ gives you the **simplicity of C and adds a lot of steroids**. It delivers lots of quality-of-life stuff, increasing the developer experience. Let's compare C with C++, shall we?

1. The iconic book "*The C Programming Language*" *by Brian W. Kernighan and Dennis M. Ritchie* has **only 263 pages**. Pretty simple, huh?
2. The book "*C++ How to Program*" *by Harvey and Paul Deitel* It holds around **1000 pages**, and the pages are way bigger than the other one.



So, don't worry, you just need to learn the basics first, and all the rest are somehow advanced concepts. I will do my best to keep you focused on what is relevant to each moment of your learning journey.

Without further ado. Get in the car!

4.2.2 Speed Matters

A LOT. Period. **C++** is one of the **closest intelligible programming languages before reaching the level of machine code**, as known as **Assembly Language**. If you code in machine code, you obviously will code precisely what you want the machine to do, but this task is too painful to be the *de-facto* standard of coding. So we need something more straightforward and more human-readable. So C++ lies in this exact area of being close to assembly language and still able to be "easily" understandable. Note the quotes, they are there because it might not be that easy when you compare its syntax to other languages, **C++ has to obey some constraints to keep the generated binary fast** as a mad horse while trying to be easier than assembly. Remember, it can always get worse.

The main philosophy that guides C++ is the **"Zero-cost abstractions"**, and it is the main reason why C++ is so fast. It means that **the language does not add any overhead to assembly**. So, if someone proposes a new core feature as a Technical specification, it should pass through this filter. And it is a very high bar to pass. I am looking at you, ts reflection, everyone I know that want to make games, ask for this feature, but it is not there yet.

4.2.3 Why does speed matter?

Mainly because we don't want to waste time. Right? But it has more impactful consequences. Let's think a bit more, you probably have a smartphone, and it lives only while it has enough energy on its battery. So, **if you are a lazy mobile developer and do not want to learn how to do code efficiently, you will make your app drain more energy from the battery** just by making the user wait for the task to be finished or by doing lots of unnecessary calculations! You will be the reason the user has not enough power to use their phones up to the end of the day. In fact, **you will be punishing your user by using your app**. You don't want that, right? So let's learn how to code appropriately. For the sake of the argument, worse than that, a lazy blockchain smart contract developer will make their users pay more for extra gas fee usage for the extra inefficient CPU cycles.

4.2.4 Language benchmarks

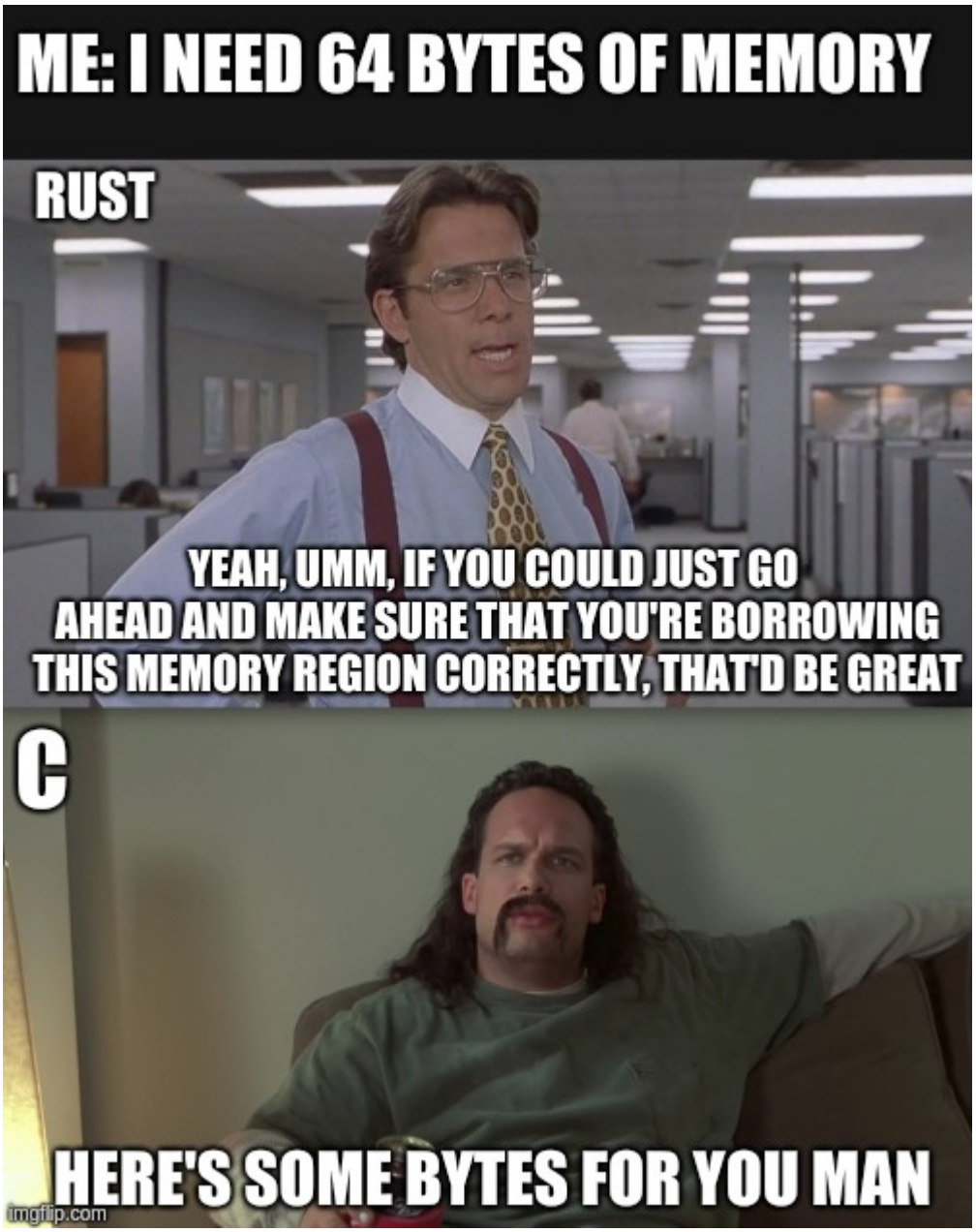
I don’t want to point fingers at languages, but, hey, excuse me, python, are you listening to me, python? Python? Please answer! reference cpp vs python. Nevermind. It is still trying to figure things out. Ah! Hey ruby, don’t be shy, I know you look gorgeous, and I admire you a lot, but can you dress up faster and be ready to run anytime soon?

You don’t need makeup to run fast. That’s the idea. If the language does lots of fancy stuff, it won’t be extracting the juicy power of the CPU.

So let’s first clarify some concepts for a fair comparison. Some languages do not generate binaries that run in your CPU. Some of them run on top of a virtual machine. **The Virtual Machine(VM) is a piece of software that, in runtime, translates the bytecode or even compiles source code to something the CPU can understand.** It’s like an old car; some of them will make you wait for the ignition or even get warm enough to run fast. I am looking at you Java and JavaScript. It is a funny concept, I admit, but you can see here that the ones that run on top of a translation device would never run as fast as a compiled binary ready to run on the CPU.

So let’s bring some ideas from my own experience, and I invite you to test by yourself. Just search for "*programming languages benchmark*" on your preferred search engine or go here.

I don’t want to start a flame-war. Those numbers might be wrong, but the overall idea is correct. Assuming C++ does not add much overhead to your native binary, let’s set the speed to run as 1x. Java would be around 1.4x slower, and JavaScript is 1.6x, python 40x, and ruby 100x. The only good competitor in the house is Rust because its compiled code runs straight on the CPU efficiently with lots of quality-of-life additions. Rust gives almost similar results if you do not play around with memory-intensive problems. Another honorable mention is WASM - Web Assembly, although it is a bytecode for a virtual machine, many programming languages are able to target it(compile for it), it is becoming blazing fast and it is getting traction nowadays, keep tuned.



4.2.5 Who should learn C++

YOU! Yes, seriously, I don't know you, but I am pretty sure **you should know how to code in any language**. C++ can be challenging, it is a fact, but if you dare to challenge yourself to learn it, your life will be somewhat better.

Let's cut to the bullets:

1. The ones who seek to build efficient modules for mobile apps, such as the video/image processing unit;
2. Game developers. Even the gameplay developers that usually only script things should know how to ride a horse(CPU) fast;
3. Researchers looking to not waste time by coding inefficient code and wait hours, even days, to see the result of their calculations. They should reduce the costs of renting CPU clusters;
4. Computer scientists are those who should know how a computer works. After all, C++ is one of the preferred programming languages that unlocks all the power of the CPU;
5. Engineers, in general, should know how to simulate things efficiently;

4.2.6 How do machines run code?

The first thing is: the CPU does not understand any programming language, only binary instructions. So you have to convert your code into something the machine can understand. This is the job of the compiler. **A compiler is a piece of software that reads a text file written following the rules of a programming language and essentially converts it into binary instructions that the CPU can execute**. There are many strategies and many ways of doing it. So, given its nature of being near assembly, **with C++, you will control precisely what instructions the CPU will run**.

But, there is a catch here: for each CPU, you will need a compiler for that instruction set. Ex.: the compiler GCC can generate an executable program for ARM processors, and the generated program won't work on x86 processors; In the same way, an x64 executable won't work on an x86; you need to match the binary instructions generated by the compiler with the same instruction set available on the target CPU you want to run it. Some compilers can cross-compile: the compiler runs in your machine on your CPU with its instruction set, but the binary generated only runs on a target machine with its own instruction set.

```
graph TD
    START((Start))-->
    |Source Code|PreProcessor-->
    |Pre-processed Code|Compiler-->
    |Target Assembly Code|Assembler-->
    |Relacable Machine Code|Linker-->
    |Executable Machine Code|Loader-->
    |Operation System|Memory-->
    |CPU|RUN((Run))
```

4.3 Program Life Cycle

Software development is complex and there is lots of styles, philosophies and standard, but the overall structure looks like this:

1. Analysis, Specification, Problem definition
2. Design of the Software (pseudocode/algorithm, flowchart), Problem analysis
3. Implementation / Coding
4. Testing and Debugging - In TDD(Test Driven Development) we write the tests first.
5. Maintenance - Analytics and Improvements
6. End of Life

4.4 Pseudocode

Pseudocode is a way of expressing algorithms using a combination of natural language and programming constructs. It is not a programming language and cannot be compiled or executed, but it provides a clear and concise way to describe the steps of an algorithm. Here is an example of pseudocode that describes the process of finding the maximum value in a list of numbers:

```
set maxValue to 0
for each number in the list of numbers
  if number is greater than maxValue
    set maxValue to number
output maxValue
```

Pseudocode is often used as a planning tool for programmers and can help to clarify the logic of a program before it is written in a specific programming language. It can also be used to communicate algorithms to people who are not familiar with a particular programming language. Reference

4.5 Flowcharts

A flowchart is a graphical representation of a process or system that shows the steps or events in a sequential order. It is a useful tool for demonstrating how a process works, identifying potential bottlenecks or inefficiencies in a process, and for communicating the steps involved in a process to others.

Flowcharts are typically composed of a series of boxes or shapes, connected by arrows, that represent the steps in a process. Each box or shape usually contains a brief description of the step or event it represents. The arrows show the flow or movement from one step to the next.

Flowcharts can be used in a variety of settings, including business, engineering, and software development. They are particularly useful for demonstrating how a process works, identifying potential issues or bottlenecks in the process, and for communicating the steps involved in a process to others.

There are many symbols and notations that can be used to create flowcharts, and different organizations and industries may have their own standards or conventions for using these symbols. Some common symbols and notations used in flowcharts include:

1. Start and end symbols: These are used to indicate the beginning and end of a process.
2. Process symbols: These are used to represent the various steps or events in a process.
3. Decision symbols: These are used to represent a decision point in a process, where the flow of the process depends on the outcome of a decision.
4. Connector symbols: These are used to connect the various symbols in a flowchart, showing the flow or movement from one step to the next.
5. Annotation symbols: These are used to add additional information or notes to a flowchart.

By using a combination of these symbols and notations, you can create a clear and concise flowchart that effectively communicates the steps involved in a process or system. Reference

I suggest using the tool Code2Flow to write pseudocode and see the flowchart drawn in real time. But you can draw them on Diagrams. If you are into sequence diagrams, I suggest using sequencediagram.org.

4.6 Practice

Try to think ahead the problem definition by questioning yourself before expressing the algorithm as pseudocode or flowchart: - What are the inputs? - What is a valid input? - How to compute the math? - What is the output? - How many decimals is needed to express the result?

Use diagrams to draw a flowchart or use Code2Flow to write a working pseudocode to: 1. Compute the weighted average of two numbers. The first number has weight of 1 and the second has weight of 3; 2. Area of a circle; 3. Compute GPA; 4. Factorial number;

4.7 Glossary

It is expected for you to know superficially these terms and concepts. Research about them. It is not strictly required, because we are going to cover them in class.

- CPU
- GPU
- ALU
- Main Memory
- Secondary Memory
- Programming Language
- Compiler
- Linker
- Assembler
- Pseudocode
- Algorithms
- Flowchart

4.8 Activities

1. Sign up on beecrowd. If you are a enrolled student, look for the key in canvas to be assigned to the coding assignments.
2. https://blockly.games/maze - test your ability to solve small problems via block programming
3. https://codecombat.com/ - very interesting game
4. https://scratch.mit.edu/ - start a project and make it say hello when you click on it

4.9 Troubleshooting

If you have problems here, start a discussion this is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me.

August 31, 2023

December 20, 2022

GitHub 

4.9.1 Comments

4.10 Tools for C++ development

Estimated time to read: 45 minutes

> **Opinion**
>
> This list is a mix of standard tools and personal choice. It is a good starting point, but in the future you will be impacted by other options, just keep your mind open to new choices.

- Version Control
- GIT
- Github
- GitKraken
- Compiler
- CMake

Every programing language use different set of tools in order to effectively code. In C++ you will need to learn how to use a bunch of them to solve problems and develop software.

4.10.1 Version Control

Version control are tools that help you to keep track of your code changes. It is a must have tool for any developer. You can keep track the state of your code, and if you mess up something, you can go back to a previous state. It is also a great tool to collaborate with other developers. You can work on the same codebase without messing up each other work.

GIT

> **Optional**
>
> **Install Git**

Git is a version control system that is used to track changes to files, including source code, documents, and other types of files. It allows multiple people to work on the same files concurrently, and it keeps track of all changes made to the files, making it easy to go back to previous versions or merge changes made by different people. Git is widely used by developers for managing source code, but it can be used to track changes to any type of file. It is particularly useful for coordinating work on projects that involve multiple people, as it allows everyone to see and track changes made by others.

Github

> **Action**
>
> **Github Student Pack**

Github is a web-based platform for version control and collaboration on software projects. It is a popular platform for developers to share and collaborate on code, as well as to track and manage software development projects. GitHub provides version control using Git, a version control system that allows developers to track changes to their codebase and collaborate with other developers on the same codebase. It also includes a range of features such as bug tracking, project management, and team

communication tools. In addition to being a platform for software development, GitHub is also a community of developers and a marketplace for buying and selling software development services.

In this course we are going to extensively use GITHUB functionalities. So create an account now with your personal account. Use a meaningful username. Avoid names that hard to associate with you. If you have a educational email or student id, apply for the Github Student Pack, so you will have access to lots of free tools.

It is nice to have `git` in your machine, but it is not required, because we are going to use gui via gui tools. See GitKraken below.

GitKraken

Action

Install Gitkraken

GitKraken is a Git client for Windows, Mac, and Linux that provides a graphical interface for working with Git repositories. It allows users to manage Git repositories, create and review changes to code, and collaborate with other developers. Some features of GitKraken include a visual representation of the repository's commit history, the ability to stage and discard changes, and support for popular version control systems like GitHub and GitLab. GitKraken is designed to be user-friendly and to make it easier for developers to work with Git, particularly for those who may be new to version control systems.

Gitkraken is a paid software, and it is free for public repositories, but you can have all enterprise and premium functionalities enabled for free with the student pack and described before.

Install Gitkraken. If you login into gitkraken using GitHub with student pack it will unlock all pro features.

4.10.2 C++ Compiler

A compiler is a type of computer program that translates source code into machine instructions that can be run or the `CPU` or interpreted in a `Virtual Machine`.

```
graph TD
  SRC[Source Code] --> |Assembly| OBJ[Machine Code];
  OBJ --> EXE[Executable];
  OBJ --> LIB[Library];
```

- `Source Code` in C++, is associated to two different type of textual file extensions: `.cpp` for sources and `.h` for header files. It is what the developer writes.
- `Assembly` is a human readable representation of the `Machine Code`. It is not the `Machine Code` itself, but it is a representation of it. It is a way to make the `Machine Code` human readable.
- `Machine Code` is what the `CPU` can run and understand. It is a sequence of `0` and `1` that the `CPU` can understand and execute. It is not human readable.
- `Executable` is the result of the compilation process. It is a file that can be executed by the `Operating System`.
- `Library` is a collection of `Machine Code` that can be used by other programs.
- `Executable` and `Library` Are binary file that contains the `Machine Code` instructions that the `CPU` can execute.

Note

In compiled languages, the end user only receives the executables and libraries. The source code is not distributed.

Here you can see briefly a small function to square a number in `C++` compiled via `GCC` into a `x86-64 assembly`. The left side is the `Source Code` and the right side is the code compiled into a human-readble `Assembly`. This code still needs links to the `Operation System` in order to be executed.

Notes on Virtual Machines (VM)

> **Tip**
>
> The knowledge of this section is not required for this course, but it is good to know.

Some languages such as `Java`, `C#` and others, compile the `Source Code` into `bytecode` that runs on top of an abstraction layer called `Virtual Machine` (VM). The VM is a software that runs on top of the `Operating System` and it is responsible to translate the `bytecode` into `Machine Code` that the `CPU` can understand. This is a way to make the `Source Code` portable across different `Operating Systems` and `CPU` architectures - `cross-platform`. But this abstraction layer has it cost and it is not as efficient as the `Machine Code` itself.

To speed up the execution, some `VM` can `Just In Time` (`JIT`) compile the `bytecode` into `Machine Code` at runtime when the `VM` detects parts of `Source Code` is running a lot(`Hotspots`), to speed up the execution. When this optmization step is happening, the machine is warming up.

```
graph TD
  SRC[Source Code] --> |Compiles| BYT[Bytecode];
  BYT --> |JIT Compiler| CPU[Machine Code];
```

> **Note**
>
> In languages that uses VMs, the end user receives the bytecode. The source code is not distributed.

Notes on Interpreters

> **Tip**
>
> The knowledge of this section is not required for this course, but it is good to know.

Some languages such as `Python`, `Javascript` and others, do not compile the `Source Code`, instead, they run on top a program called `Interpreter` that reads the `Source Code` and executes it line by line.

```
graph TD
  SRC[Source Code] --> |read line| INT[Interpreter];
  INT --> |translates| CPU[Machine Code];
```

Some `Interpreters` are `Ahead Of Time` (`AOT`) and they compile the `Source Code` into `Machine Code` before the `Source Code` is executed.

```
graph TD
  SRC[Source Code] --> |AoT compile| INT[Bytecode / Machine Code];
  INT --> CPU;
```

> **Note**
>
> In intrepreted languages, the end user receives the source code. Sometimes the source code is obfuscated, but it is still readable.

Platform specific

This where things get tricky, C++ compiles the code into a binary that runs directly on the processor and interacts with the operating system. So we can have multiple combinations here. Most compilers are cross-platform, but there is exceptions. And to worsen it, some Compilers are tightly coupled with some IDEs(see below, next item).

I personally prefer to use CLang to be my target because it is the one that is most reliable cross-platform compiler. Which means the code will work as expected in most of the scenarios, the feature support table is the same across all platforms. But GCC is the more bleeding edge, which means usually it is the first to support all new fancy features C++ introduces.

No need to download anything here. We are going to use the CLion IDE. See below topics.

4.10.3 CMake

CMake CMake is a cross-platform free and open-source software tool for managing the build process of software using a compiler-independent method. It is designed to support directory hierarchies and applications that depend on multiple libraries. It is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice.

> **Note**
>
> If you use a good IDE(see next topic), you won't need to download anything here.

CMake is typically used in conjunction with native build environments such as Make, Ninja, or Visual Studio. It can also generate project files for IDEs such as Xcode and Visual Studio. You can see a full list of supported generators here.

Here is a simple example of a `CMakeLists.txt` file that can be used to build a program called "myproject" that consists of a single source file called "main.cpp":

```
# Set minimum version of CMake that can be used
cmake_minimum_required(VERSION 3.10)
# Set the project name
project(myproject)
# Add executable named "myproject" to be built from the source "main.cpp"
add_executable(myproject main.cpp)
```

> **Warning**
>
> Every executable can only cave one `main` function. Each file with a main function describes a new executable program. If you want to have multiple executables in the same project, in other words, you want to manage multiple executables in the same place, you can change the cmake descriptor to match that as follows, and use your IDE to switch between them:
>
> ```
> cmake_minimum_required(VERSION 3.10)
> project(myproject)
> add_executable(myexecutable1 main1.cpp)
> add_executable(myexecutable2 main2.cpp)
> ```

> **Tip**
>
> If you are using a nice IDE, you won't need to run this on the command line. So go to next topic.

If you want to build via command line this project, you would first generate a build directory, and then run CMake to build the files using the detected compiler or IDE:

```
cmake -S. -Bbuild
cmake --build build -j20
```

This will create a Makefile or a Visual Studio solution file in the build directory, depending on your platform and compiler. You can then use the native build tools to build the project by running "make" or opening the solution file in Visual Studio.

CMake provides many options and variables that can be used to customize the build process, such as setting compiler flags, specifying dependencies, and configuring installation targets. You can learn more about CMake by reading the documentation at https://cmake.org/.

[Subscribe to our newsletter][#]{ .md-button }

4.10.4 IDE

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE typically integrates a source code editor, build automation tools, and a debugger. Some IDEs also include additional tools, such as a version control system, a class browser, and a support for literate programming. IDEs are designed to maximize programmer productivity by providing tight-knit components with similar user interfaces. This can be achieved through features such as auto-complete, syntax highlighting, and code refactoring. Many IDEs also provide a code debugger, which allows the programmer to step through code execution and find and fix errors. Some examples of popular IDEs include Eclipse, NetBeans, Visual Studio, and Xcode. Each of these IDEs has its own set of features and capabilities, and developers may choose one based on their specific needs and preferences.

In this course, it is strongly suggested to use an IDE in order to achieve higher quality of deliveries, a good IDE effectively flatten the C++ learning curve. You can opt out and use everything by hand, of course, and it will deepen your knowledge on how things work but be assured it can slow down your learning process. Given this course is result oriented, it is not recommended to not use an IDE here. So use one.

OPINION: The most pervasive C++ IDE is CLion and this the one I am going to use. If you use it too, it would be easier to follow my recorded videos. It works on all platforms Windows, Linux and Mac. I recommend downloading it via Jetbrains Toolbox. If you are a student, apply for student pack for free here. On Windows, CLion embeds a GCC compiler or optionally can use visual studio, while on Macs it requires the xcode command line tools, and on Linux, uses GCC from the system installation.

The other options I suggest are:

On all platforms

REPLIT - an online and real-time multiplayer IDE. It is slow and lack many functionalities, but can be used for small scoped activities or work with a friend.

VSCode - a small and highly modularized code editor, it have lots of extensions, but it can be complex to set up everything needed: git, cmake, compiler and other stuff.

On Windows:

Visual Studio - mostly for Windows. When installing, mark C++ development AND search and install additional tools "CMake". Otherwise, this repo won't work smoothly for you.

DevC++ - an outdated and small IDE. Lacks lots of functionalities, but if you don't have HD space or use an old machine, this can be your option. In long term, this choice would be bad for you for the lack of functionalities. It is better to use REPLIT than this tool, in my opinion.

On OSX

XCode - for OSX and Apple devices. It is required at least to have the Command Line Tools. CLion on Macs depends on that.

Xcode Command Line Tools is a small suite of software development tools that are installed on your Mac along with Xcode. These tools include the GCC compiler, which is used to compile C and C++ programs, as well as other tools such as Make and GDB, which are used for debugging and development. The Xcode Command Line Tools are necessary for working with projects from the command line, as well as for using certain software development tools such as Homebrew.

To install the Xcode Command Line Tools, you need to have Xcode installed on your Mac. To check if Xcode is already installed, open a Terminal window and type:

```
xcode-select -p
```

If Xcode is already installed, this command will print the path to the Xcode developer directory. If Xcode is not installed, you will see a message saying "xcode-select: error: command line tools are not installed, use xcode-select --install to install."

To install the Xcode Command Line Tools, open a Terminal window and type:

```
xcode-select --install
```

This will open a window that prompts you to install the Xcode Command Line Tools. Follow the prompts to complete the installation.

Once the Xcode Command Line Tools are installed, you can use them from the command line by typing commands such as gcc, make, and gdb. You can also use them to install and manage software packages with tools like Homebrew.

On Linux

If you are using Linux, you know the drill. No need for further explanations here, you are ahead of the others.

If you are using an Ubuntu distro, you can try this to install most of the tools you will need here:

```
sudo apt-get update && sudo apt-get install -y build-essential git cmake lcov xcb libx11-dev libx11-xcb-dev libxcb-randr0-dev
```

In order to compile:

```
g++ inputFile.cpp -o executableName
```

Where `g++` is the compiler frontend program to compile your C++ source code; `inputFile.cpp` is the filename you want to compile, you can pass multiple files here separated by spaces ex.: `inputFile1.cpp inputFile2.cpp`; `-o` means the next text will be the output program name where the executable will be built, (for windows, the name should end with `.exe` ex.: `program.exe`).

You will have a plethora of editors and IDEs. The one I can suggest is the VSCode, Code::Blocks or KDevelop. But I really prefer CLion.

4.11 CLion project workflow with CMake

When you create a new project, select `New C++ Executable`, set the `C++ Standard` to the newest one, `C++20` is enough, and place in a folder location where you prefer.

CLion automatically generate 2 files for you. - `CMakeLists.txt` is the CMake multiplatform project descriptor, with that, you can share your project with colleagues that are using different platforms than you. - `main.cpp` is the entry point for your code.

It is not the moment to talk about multiple file projects, but if you want to get ready for it, you will have to edit the `CMakeLists.txt` file and add them in the `add_executable` function.

4.12 Hello World

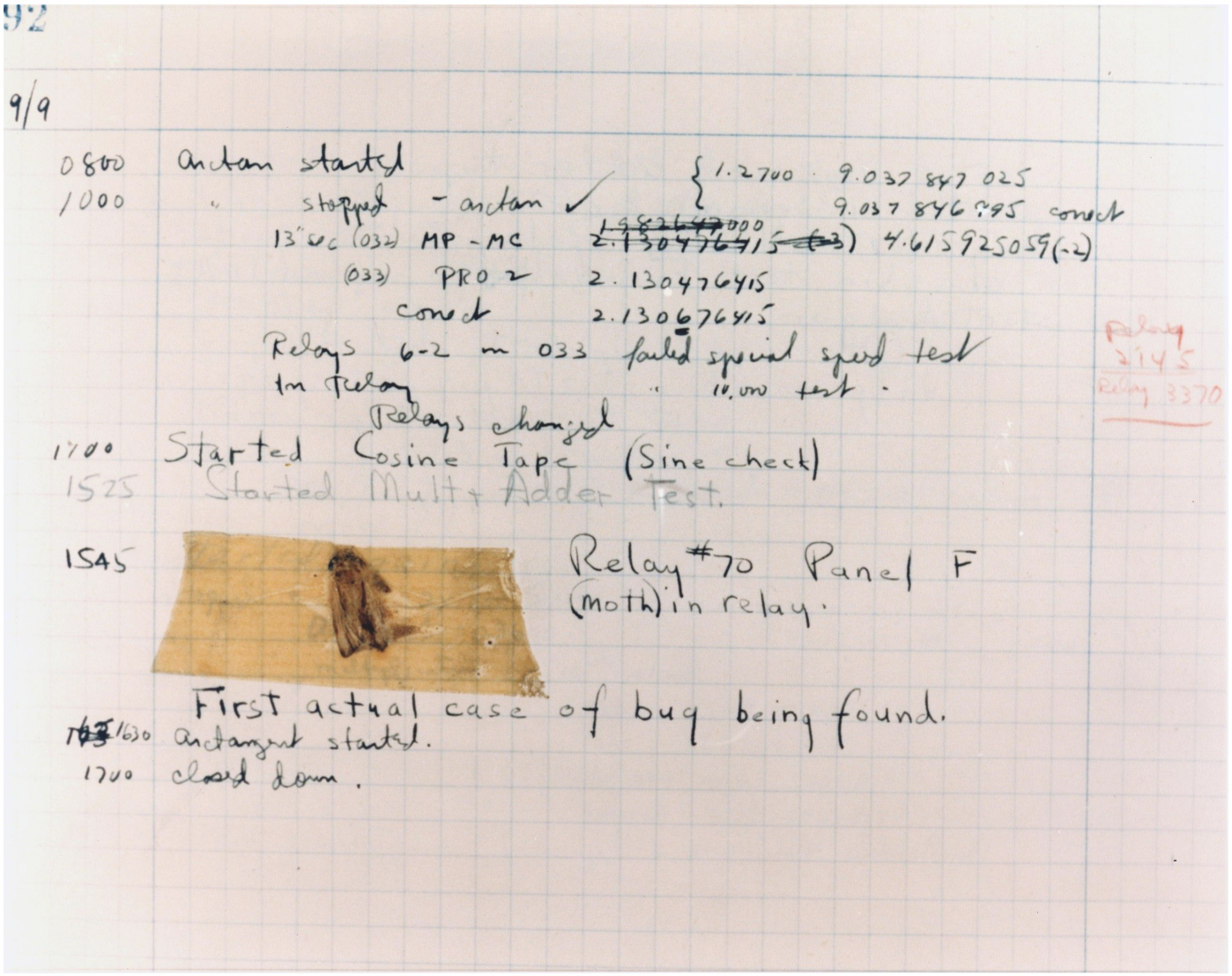
Hello World

```
// this a single line comment and it is not compiled. comments are used to explain the code.
// you can do single line comment by adding // in front of the line or
// you can do multi line comments by wrapping your comment in /* and */ such as: /* insert comment here */
/* this is
 * a multi line
 * comment
 */
#include <iostream> // this includes an external library used to deal with console input and output

using namespace std; // we declare that we are going to use the namespace std of the library we just included

// "int" means it should return an integer number in the end of its execution to communicate if it finished properly
// "main()" function where the operating system will look for starting the code.
// "()" empty parameters. this main function here needs no parameter to execute
// anynthing between { and } is considered a scope.
// everything stack allocated in this scope will be deallocated in the end of the scope. ex.: local variables.
int main() {
    /* "cout" means console output. Print to the console the content of what is passed after the
     * "<<" stream operator. Streams what in the wright side of it to the cout object
     * "endl" means end line. Append a new line to the stream, in the case, console output.
     */
    cout << "Hello World" << endl;

    /* tells the operating system the program finished without errors. Any number different from that is considered
     * a error code or error number.
     */
    return 0;
}
```

4.13 Hello Username

```cpp
#include <iostream>
#include <string> // structure to deal with a char sequence, it is called string
using namespace std;
int main(){
    // invites the user to write something
    cout << "Type your name: " << endl;

    /* * string means the type of the variable, this definition came from the string include
     * username means the name of the variable, the container to hold and store the data
     */
    string username;
    /*
     * cin mean console input. It captures data from the console.
     * note the opposite direction of the stream operator. it streams what come from the cin object to the variable.
     */
    cin >> username;
    // example of how to stream and concatenate texts to the console output;
    cout << "Hello " << username << endl;
}
```

4.14 Common Bugs



First documented bug found in 1945

4.14.1 1. Syntax error

Syntax errors in C++ are usually caused by mistakes in the source code that prevent the compiler from being able to understand it. Some common causes of syntax errors include: 1. Omitting a required component of a statement, such as a semicolon at the end of a line or a closing curly brace. 2. Using incorrect capitalization or spelling in a keyword or identifier. 3. Using the wrong punctuation, such as using a comma instead of a semicolon. 4. Mixing up the order of operations, such as using an operator that expects two operands before the operands have been provided.

To fix a syntax error, you will need to locate the source of the error and correct it in the code. This can often be a challenging task, as syntax errors can be caused by a variety of factors, and it is not always immediately clear what the problem is. However, there are a few tools that can help you locate and fix syntax errors in your C++ code: 1. A compiler error message: When you try to compile your code, the compiler will often provide an error message that can help you locate the source of the syntax error. These error messages can be somewhat cryptic, but they usually include the line number and a brief description of the problem. 2. A text editor with syntax highlighting: Many text editors, such as Visual Studio or Eclipse, include syntax highlighting, which can help you identify syntax errors by coloring different parts of the code differently. For example, keywords may be highlighted in blue, while variables may be highlighted in green. 3. A debugger: A debugger is a tool that allows you to step through your code line by line, examining the values of variables and the state of the program at each step. This can be a very useful tool for tracking down syntax errors, as it allows you to see exactly where the error occurs and what caused it.

Reference

4.14.2 2. Logic Error

A logic error in C++ is an error that occurs when the code produces unintended results or behaves in unexpected ways due to a mistake in the logic of the program. This type of error is usually caused by a coding mistake, such as using the wrong operator, omitting a necessary statement, or using the wrong variable. Here are some common causes of logic errors in C++:

- Incorrect use of conditional statements (e.g., using the wrong comparison operator or forgetting to include a necessary else clause)
- Mistakenly using the assignment operator (=) instead of the equality operator (==) in a conditional statement
- Omitting a necessary loop iteration or failing to terminate a loop at the appropriate time
- Using the wrong variable or array index
- Incorrectly calling a function or passing the wrong arguments to a function

To fix a logic error in C++, you will need to carefully examine your code and identify the mistake. It may be helpful to use a debugger to step through your code and see how it is executing, or to add print statements to help you understand what is happening at each step.

Reference

4.14.3 3. Run-time error

A runtime error in C++ means that there is an error in your program that is causing it to behave unexpectedly or crash during runtime, i.e., after you have compiled and run the program. There are many possible causes of runtime errors in C++, including:

- Dereferencing a null pointer
- Accessing an array out of bounds
- Using an uninitialized variable
- Trying to divide by zero
- Attempting to use an object that has been deleted or has gone out of scope

To troubleshoot a runtime error, you'll need to identify the source of the error by examining the error message and the code that is causing the error. Some common tools and techniques you can use to troubleshoot runtime errors include:

- Using a debugger to step through your code line by line
- Printing out the values of variables to see where the error might be occurring
- Adding additional debug statements or logging to your code to help identify the source of the error

It's also a good idea to ensure that you have compiled your code with debugging symbols enabled, as this will allow you to use the debugger to get a better understanding of what is happening in your code. will cause the program to crash during run-time

Reference

4.15 Exercises:

- Research and read about other notable errors: segmentation fault, stack overflow, buffer overflow.
- Hello World - just print `hello world`.

4.16 Homework

1. Setup your environment for your needs following the choices given above. If you are unsure, use CLion and you will be mostly safe.
2. Fork this repo privately. You will have to do your assignments there. Go to the home repo and hit fork.
3. Clone this repo to your machine. gitkraken + github gitkraken clone gitkraken big tutorial
4. Make sure the CMake option "ENABLE_INTRO" is set as ON in CMakeLists.txt file in the root directory in order to see and enable all activities.
5. (enrolled students) If you are enrolled in a class with me, share your repo with me, so I can track your evolution. And do the activities described there.
6. (optional) star this repo :-)

4.17 Troubleshooting

If you have problems here, start a discussion this is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me.

August 30, 2023

December 20, 2022

GitHub 

4.17.1 Comments

4.18 Variables, Data Types, Expressions, Assignment, Formatting

Estimated time to read: 49 minutes

4.19 Variables

Variables are containers to store information and facilitates data manipulation. They are named and typed. Detailed Reference

Container sizes are measured in Bytes. Bytes are the smallest addressable unit in a computer. Each byte is composed by 8 bits. Each bit can be `1` or `0` (`true` or `false`). If one byte have 8 bits and each bit one can hold 2 different values, the combination of all possible cases that a byte can be is `2^8` which is `256`, so one byte can hold up to `256` different states or possibilities.

4.19.1 Data Types

There are several types of variables in C++, including:

- Primitive data types: These are the most basic data types in C++ and include integer, floating-point, character, and boolean types.
- Derived data types: These data types are derived from the primitive data types and include arrays, pointers, and references.
- User-defined data types: These data types are defined by the programmer and include structures, classes, and enumerations.

Detailed Reference

Numeric types

There are some basic integer container types with different sizes. It can have some type modifiers to change the default behavior or the type.

The common size of the integer containers are `1` (`char`), `2` (`short int`), `4` (`int`) or `8` (`long long`) bytes. For a more detailed coverage read this.

> **Note**
>
> But the only guarantee the C++ imposes is: `1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)` and it can result in compiler defined behaviours where a `char` can have 8 bytes and a `long long` can be 1 byte.

> **Note**
>
> If you care about being cross-platform conformant, you have to always specify the sign modifier or use a more descriptive type such as listed here.

For floating pointing numbers, the container size can be `4` (`float`), `8` (`double`), `10` (deprecated) or `16` (`long double`) bytes.

The sign modifiers can be `signed` and `unsigned` and are applicable only for integer types.

The default behavior of the types in a x86 cpu are as signed numbers and the first bit of the container is the signal. If the first bit is `0`, it means it is positive. If the first bit is `1`, it means it is negative. More details.

Which means that if the container follow two complement and is the size of 1 byte(8 bits), it have 1 bit for the signal and 7 bit for the content. So this number goes from `-128` up to `127`, this container is typically a `signed char`. The positive size has 1 less quantity in absolute than the negative because 0 is represented in positive side. There are `256` numbers between `-128` and `127` inclusive.

CHAR

A standard `char` type uses 1 byte to store data and follows complement of 1. Going `-127` to `127`, so tipically represents `255` numbers.

A `signed char` follows complement of 2 and it can represent `2^8` or `256` different numbers. By default, in x86 machine char is signed and the represented numbers can go from `-2^7` or `-128` up to `2^7 - 1` or `127`.

An `unsigned char`

Chars can be used to represent letters following the ascii table where each value means a specific letter, digit or symbol.

> **Note**
>
> A `char` can have different sizes to represent different character coding for different languages. If you are using hebrew, chinese, or others, you probably will need more than 1 byte to represent the chars. Use `char8_t` (UTF8), `char16_t` (UTF16) or `char36_t` (UTF32), to cover your character encoding for the language you are using.

INTEGER

> **Note**
>
> Most of the information that I am covering here might be not precise, but the overall idea is correct. If you want a deep dive, read this.

A standard `int` type uses 4 bytes to store data. It is signed by default.

It can represent `2^32` or `4294967296` different numbers. As a signed type, it can represent numbers from `-2^31` or `-2147483648` up to `2^31 - 1` or `2147483647`.

The type `int` can accept sign modifiers as `signed` or `unsigned` to change the behavior of the first bit to act as a sign or not.

The type `int` can accept size modifiers as `short` (2 bytes) or `long long` (8 bytes) to change the size and representation capacity of the container. Type declaration `short` and `short int` result in the same container size of 2 bytes. In the same way a `long long` or `long long int` reserves the same size of 8 bytes for the container.

The type `long` or `long int` usually gives the same size of `int` as 4 bytes. Historical fact or myth: This abnormality, comes from the evolution of the definition of `int`: in the past, 2 bytes were enough for the majority of the scenarios in the 16 bits processors, but it frequently reached the limits of the container and it overflowed. So they changed the standard definition of a integer from being 2 bytes to 4 bytes, and created the short modifier. In this scenario the long int lost the reason to exist.

Here goes a list of valid integer types and its probable size(it depends on the implementation, cpu architecture and operation system): - Size of 2 bytes: `short int`, `short`, `signed short int`, `signed short`, `unsigned short int`, `unsigned short`, - Size of 4 bytes: `signed`, `unsigned`, `int`, `signed int`, `unsigned int`, `long int`, `long`, `signed long int`, `signed long`, `unsigned long int`, `unsigned long`, - Size of 8 bytes: `long long int`, `long long`, `signed long long int`, `signed long long`, `unsigned long long int`, `unsigned long long`.

OPINION: I highly recommend the usage of these types instead, to ensure determinism and consistency between compilers, operating systems and cpu architectures.

FLOAT POINTING

There are 3 basic types of floating point containers: `float` (4 bytes) and `double` (8 bytes) and `long double` (16 bytes) to represent fractional numeric types.

The standard IEEE754 specifies how a floating point number is stored in the form of bits inside the container. The container holds 3 basic information to simulate the behavior of a fractional type inside a binary type: signal, exponent and fraction.

> **Note**
>
> This standard was very open to implementation definition in the past, and this is one of the root causes of non-determinism physics simulation. This is the main problem you cannot guarantee the same operation with the same pair of numbers will consistently give the same result across different types of processors and compilers, thus making the physics of a multiplayer game consistency hardly achievable. Many deterministic physics engines tend to not use this standard at all, and implement those behaviors via software on top of integers instead. There are 2 approaches to solve the floating-point determinism: softfloat that implement all the IEEE754 specifications via software, or implement some kind of fixed-point arithmetic on top of integers.

BOOLEANS

`bool` is a special type that has the container size of 1 byte but the compiler can optimize and pack up to 8 `bool`s in one byte if they are declared in sequence.

ENUMS

An enumeration is a type that consists of a set of named integral constants. It can be defined using the `enum` keyword:

```
enum Color {
  Red,
  Green,
  Blue
};
```

This defines a new type called `Color`, which has three possible values: `Red`, `Green`, and `Blue`. By default, the values of these constants are `0`, `1`, and `2`, respectively. However, you can specify your own values:

```
enum Color {
  Red = 5,
  Green,  // 6
  Blue    // 7
};
```

You can then use the enumeration type just like any other type:

```
Color favoriteColor = Red;
```

Enumerations can also have their underlying type explicitly specified:

```
enum class Color : char {
  Red,
  Green,
  Blue
};
```

Here, the underlying type of the enumeration is `char`, so the constants `Red`, `Green`, and `Blue` will be stored as characters. The `enum class` syntax is known as a "scoped" enumeration, and it is recommended over the traditional `enum` syntax because it helps prevent naming conflicts. See the CppCoreGuidelines to understand better why you should prefer using this.

Special derived type: string

`string` is a derived type and in order to use it, string should be included in the beginning of the file or in the header. `char` are the basic unit of a `string` and is used to store words as a sequence of chars.

In C++, a string is a sequence of characters that is stored in an object of the `std::string` class. The `std::string` class is part of the C++ Standard Library and provides a variety of functions and operators for manipulating strings.

`void type`

When `void` type specifier is used in functions, it indicates that a function does not return a value.

It can also be used as a placeholder for a pointer to a memory location to indicate that the pointer is "universal" and can point to data of any type, but this can be arguably a bad pattern, and should be used exceptionally when interchanging types with c-style API.

We are going to cover this again when covering pointers and functions.

4.19.2 Variable Naming

Variable names are called identifiers. In C++, you can use any combination of letters, digits, and underscores to name a variable, it should follow some rules: - Variables can have numbers, en any position, except the first character, so the name does not begin with a digit. Ex. `point2` and `vector2d` are allowed, but `9life` isn't; - Variable names are case-sensitive, so "myVar" and "myvar" are considered to be different variables; - Can have `_` in any position of the identifier. Ex. `_myname` and `user_name` are allowed; - It is not a reserved keyword;

Keep in mind that it is a good practice to choose descriptive and meaningful names for your variables, as this can make your code easier to read and understand. Avoid using abbreviations or acronyms that may not be familiar to others who may read your code.

It is also important to note that C++ has some naming conventions that are commonly followed by programmers. For example, it is common to use camelCase or snake_case to separate words in a variable name, and to use all lowercase letters for variables that are local to a function and all uppercase letters for constants.

4.19.3 Variable declaration

Variable declaration in C++ follows this pattern.

```
TYPENAME VARIABLENAME;
```

`TYPENAME` can be the name of any predefined type. See Variable Types for the types. `VARIABLENAME` can be anything as long it follow the naming rules. See Variable Naming for the naming rules.

> **Note**
>
> A given variable name can only be declared once in the same context / scope. If you try to redeclare the same variable, the compiler will accuse an error.

> **Note**
>
> You can redeclare the same variable name in different scopes. If one scope is parent of the other, the current will be used and will shadow the content of the one from outer scope. We are going to cover this more when we are covering multi-file projects and functions.

Examples:

```
int a;       // integer variable
float pi;    // floating-point variable
char c;      // character variable
bool d;      // boolean variable
string name; // string variable
```

> **Note**
>
> We are going to cover later in this course other complex types in other modules such as arrays, pointers and references.

4.20 Variable assignment

`=` operator means that whatever the container have will be overwritten by the result of the right side statement. You should read it not as `equal` but as `receives` to avoid misunderstanding. Reference

```
int a = 10;         // integer variable
float pi = 3.14;    // floating-point variable
```

```
char c = 'A';       // character variable
bool d = true;      // boolean variable
string name = "John Doe"; // string variable
```

Every variable, by default, is not initialized. It means that you have to set the content of it after declaring. If the variable is read before the assignment, its content is garbage, it will read whatever is set in the memory stack for the given container location. So the best approach is to always set a value when a variable is declared or be assured that every variable is never read before an assigment.

A `char` variable can be assigned by integer numbers or any characters between single quotes.

```
char c;
c = 'A'; // the content is 65 and the representation is A. see ascii table.
c = 98; // the content is 98 and the representation is b. see ascii table.
```

A `bool` is by default either `true` or `false`, but it can be assigned by numeric value following this rule: - if the value is `0`, then the value stored by the variable is `false` (`0`); - if the value is anything different than `0`, the value stored is `true` (`1`);

To convert a `string` to a `int`, you have to use a function stoi(for `int`), stol(for `long`) or stoll(for `long long`) because both types are not compatibles.

To convert a `string` to a `float`, you have to use a function stof(for `float`), stod(for `double`), or stold(for `long double`) because both types are not compatibles.

4.21 Literals

Literals are values that are expressed freely in the code. Every numeric type can be appended with suffixes to specify explicitly the type to avoid undefined behaviors or compiler defined behaviors such as implicit cast or container size.

4.21.1 Integer literals

There are 4 types of integer literals. - decimal-literal: never starts with digit `0` and followed by any decimal digit; - octal-literal: starts with `0` digit and followed by any octal digit; - hex-literal: starts with `0x` or `0X` and followed by any hexadecimal digit; - binary-literal: starts with `0b` or `0B` and followed by any binary digit;

```
// all of these variables holds the same value, 42, but using different bases.
// the right side of the = are literals
int deci = 42;
int octa = 052;
int hexa = 0x2a;
int bina = 0b101010;
```

Suffixes: - `no suffix` provided: it will use the first smallest signed integer container that can hold the data starting from `int`; - `u` or `U`: it will use the first smallest unsigned integer container that can hold the data starting from `unsigned int`; - `l` or `L`: it will use the first smallest signed integer container that can hold the data starting from `long`; - `lu` or `LU`: it will use the first smallest unsigned integer container that can hold the data starting from `unsigned long`; - `ll` or `LL`: it will use the long long signed integer container `long long`; - `llu` or `LLU`: it will use the long long unsigned integer container `unsigned long long`;

```
unsigned long long l1 = 15731685574866854135ull;
```

Reference

4.21.2 Float point literals

There are 3 suffixes in floating point decimals. - `no suffix` means the container is a double; - `f` suffix means it is a float container; - `l` suffix means it is a long double container;

A floating point literal can be defined by 3 ways: - digit-sequence decimal-exponent suffix(optional). - `1e2` means its a `double` with the value of `1*10^2` or `100`; - `1e-2f` means its a `float` with the value of `1*10^-2` or `0.01`; - digit-sequence . decimal-exponent(optional) suffix(optional).
- `2.` means it is a `double` with value of `2`; - `2.f` means it is a `float` with value of `2`; - `2.1l` means it is a `long double` with value

of `2.1`; - digit-sequence(optional) . digit-sequence decimal-exponent(optional) suffix(optional) - `3.1415f` means it is a `float` with value of `3.1415`; - `.1` means it is a `double` with value of `0.1`; - `0.1e1L` means it is a `long double` with value of `1`;

Reference

4.22 Arithmetic Operations

In C++, you can perform common arithmetic operations is statements using the following operators Reference:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Modulus (remainder): `%`

There are two special cases called unary increment / decrement operators that may occur in before(prefixed) or after(postfixed) the variable name reference. If prefixed it is executed first and then return the result, if postfixed, it returns the current value and then execute the operation: - Increment: `++`; - Decrement: `--`;

There are shorthand assignment operators reference that reassign the value of the variable after executing the arithmetic operation with the right side of the operator with the old value of the variable: - Addition: `+=` - Subtraction: `-=` - Multiplication: `*=` - Division: `/=` - Modulus (remainder): `%=`

Here is an example of how to use these operators in a C++ program:

```
#include <iostream>

int main() {
  int a = 5;
  int b = 2;

  std::cout << a + b << std::endl; // Outputs 7
  std::cout << a - b << std::endl; // Outputs 3
  std::cout << a * b << std::endl; // Outputs 10
  std::cout << a / b << std::endl; // Outputs 2
  std::cout << a % b << std::endl; // Outputs 1
  a++;
  std::cout << a << std::endl; // Outputs 6
  a--;
  std::cout << a << std::endl; // Outputs 5

  std::cout << a++ << std::endl; // Outputs 5 because it first returns the current value and then increments.
  std::cout << a << std::endl; // Outputs 6

  std::cout << --a << std::endl; // Outputs 5 because it first decrements the value and then return it already changed;
  std::cout << a << std::endl; // Outputs 5

  b *= 2; // it is a short version of b = b * 2;
  std::cout << b << std::endl; // Outputs 4

  b /= 2; // it is a short version of b = b / 2;
  std::cout << b << std::endl; // Outputs 2

  return 0;
}
```

Note that the division operator (`/`) performs integer division if both operands are integers. If either operand is a floating-point type, the division will be performed as floating-point division. So `5/2` is `2` because both are integers, se we use integer division, but `5/2.` is `2.5` because the second one is a `double` literal.

Also, the modulus operator (`%`) returns the remainder of an integer division. For example, `7 % 3` is equal to `1`, because `3` goes into `7` two times with a remainder of `1`.

4.22.1 Implicit cast

Implicit casting, also known as type coercion, is the process of converting a value of one data type to another data type without the need for an explicit cast operator. In C++, this can occur when an expression involves operands of different data types and the compiler automatically converts one of the operands to the data type of the other in order to perform the operation.

For example:

```
int a = 1;
double b = 1.5;

int c = a + b; // c is automatically converted to a double before the addition
```

In this example, the value of `b` is a double, while the value of `a` is an `int`. When the addition operator is used, the compiler will automatically convert a to a `double` before performing the addition. The result of the expression is a `double`, so `c` is also automatically converted to a `double` before being assigned the result of the expression.

Implicit casting can also occur when assigning a value to a variable of a different data type. For example:

```
int a = 2;
double b = a; // a is automatically converted to a double before the assignment
```

In this case, the value of `a` is an int, but it is being assigned to a double variable. The compiler will automatically convert the value of `a` to a `double` before making the assignment.

It's important to be aware of implicit casting, because it can sometimes lead to unexpected results or loss of precision if not handled properly. In some cases, it may be necessary to use an explicit cast operator to explicitly convert a value to a specific data type.

4.22.2 Explicit cast

In C++, you can use an explicit cast operator to explicitly convert a value of one data type to another. The general syntax for an explicit cast are:

```
// ref: https://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages/C%2B%2B/Code/Statements/Variables/Type_Casting
(TYPENAME) value; // regular c-style. do not use this extensively
static_cast<TYPENAME>(value); // c++ style conversion, arguably it is the preferred style. use this if you know what you are doing.
TYPENAME(value); // functional initialization, slower but safer. might not work for every case. use this if you are unsure or want to be safe.
TYPENAME{value}; // initialization style, faster, convenient, concise and arguably safer because it triggers warnings. use this for the general case.
```

For example:

```
int a = 7;
double b = (double) a; // a is explicitly converted to a double
```

In this example, the value of `a` is an `int`, but it is being explicitly converted to a `double` using the explicit cast operator. The result of the cast is then assigned to the `double` variable `b`.

Explicit casts can be useful in situations where you want to ensure that a value is converted to a specific data type, regardless of the data types of the operands in an expression. However, it's important to be aware that explicit casts can also lead to unexpected results or loss of precision if not used carefully. This behaviour is called *narrowing*.

C-style:

```
int a = 20001;
char b = (char) a; // b is assigned the ASCII value for the character '!'
```

In this case, the value of `a` is an `int`, but it is being explicitly converted to a `char` using the explicit cast operator. However, the range of values that can be represented by a `char` is much smaller than the range of values that can be represented by an `int`, so the value of `a` is outside the range that can be represented by a `char`. As a result, `b` is assigned the ASCII value for the character `1`, which is not the same as the original value of `a`. The value `!` is `33` in ASCII table, and `33` is the result of the `20001 % 256` where `256` is the number of elements the `char` can represent. In this case, what happened was a bug that is hard to track called `int overflow`.

4.23 `auto` keyword

`auto` keyword is mostly a syntax sugar to automatically infer the data type. It is used to avoid writing the full declaration of complex types when it is easily inferred. `auto` is not a dynamic type, once it is inferred, it cannot be changed later like in other dynamic typed languages such as javascript.

```
auto i = 0; // automatically inferred as an integer type;
auto f = 0.0f; // automatically inferred as a float type;

i = "word"; // this won't work, because it was already inferred as an integer and integer container cannot hold string
```

4.24 Formatting

There are many functions to help you format the output in the way it is expected, here goes a selection of the most useful ones I can think. Yon can find more functions and manipulators here and here.

To set a fixed precision for floating point numbers in C++, you can use the `std::setprecision` manipulator from the `iomanip` header, along with the `std::fixed` manipulator.

Here's an example of how to use these manipulators to output a floating point number with a fixed precision of 3 decimal places:

```
#include <iostream>
#include <iomanip>

int main() {
  double num = 3.14159265;

  std::cout << std::fixed << std::setprecision(3) << num << std::endl;
  // Output: 3.142
  return 0;
}
```

You can also use the `std::setw` manipulator to set the minimum field width for the output, which can be useful for aligning the decimal points in a table of numbers.

For example:

```
#include <iostream>
#include <iomanip>

int main() {
  double num1 = 3.14159265;
  double num2 = 123.456789;

  std::cout << std::fixed << std::setprecision(3) << std::setw(8) << num1 << std::endl;
  std::cout << std::fixed << std::setprecision(3) << std::setw(8) << num2 << std::endl;
  // Output:
  //   3.142
  // 123.457
  return 0;
}
```

Note that these manipulators only affect the output stream, and do not modify the values of the floating point variables themselves. If you want to store the numbers with a fixed precision, you will need to use a different method such as rounding or truncating the numbers.

To align text to the right or left in C++, you can use the `setw` manipulator in the `iomanip` header and the `right` or `left` flag. More details here

Here is an example:

```
#include <iostream>
#include <iomanip>

int main() {
  std::cout << std::right << std::setw(10) << "Apple" << std::endl;
  std::cout << std::left << std::setw(10) << "Banana" << std::endl;
  return 0;
}
```

Both will print inside a virtual column with the size of 10 chars. This will output the following:

```
     Apple
Banana
```

4.25 Optional Exercises

Do all exercises up to this topic here.

In order to get into coding, the easiest way to learn is by solving coding challenges. It is like learning any new language, you have to be exposed and involved. Do not do only the homeworks, otherwise you are going to fail. Another metaphor is: the homework is the like a competition that you have to run to prove that you are trained, but in order to train, you have to do small runs and do small steps first, so you have to train yourself ot least 2x per week.

The best way to train yourself in coding and solving problems in my opinion is this:

1. Sort Beecrowd questions from the most solved to the least solved questions here is the link of the list already filtered.
2. Start solving the questions from the top to the bottom. Chose one from de the beginning, it would be one of the easiest;
3. If you are feeling comfortable and being able to solve more than 3 per hour, you are allowed to skip some of the questions. It is just like in a gym, when you get used with the load, you increase it. Otherwise continue training slowly.

4.26 Homework

banknotes and coins - Here you will use formatting, modulus, casting, arithmetic operations, compound assignment. You don't need to use if-else.

Hint. Follow this only if dont find your way of solving it. You can read the number as a double, multiply by 100 and then do a sequence of modulus and division operations.

```
double input; // declare the container to store the input
cin >> input; // read the input

long long cents = static_cast<long long>(input * 100); // number of cents. Note: if you just use float, you will face issues.

long long notes100 = cents/10000; // get the number of notes of 100 dollar (100 units of 100 cents)
cents %= 10000; // remove the amount of 100 dollars
```

Another good way of solving it avoiding casting is reading the number as string and removing the point. Never use `float` for money

```
string input; // declare the container to store the input
cin >> input; // read the input

// given every input will have the dot, we should remove it. remove the dot `.`
input = input.erase(str.find('.'), 1);

// not it is safe to use int, because no bit is lost in floating casting and nobody have more than MAX_INT cents.
int cents = stoll(input); // number of cents.

long long notes100 = cents/10000; // get the number of notes of 100 dollar (100 units of 100 cents)
cents %= 10000; // update the remaining cents by removing the amount of 100 dollars in cents units
```

4.27 Troubleshooting

If you have problems here, start a discussion. Nhis is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me. Or visit the tutoring service.

August 30, 2023

December 20, 2022

GitHub 

4.27.1 Comments

4.28 Conditionals, Switch, Boolean Operations

Estimated time to read: 23 minutes

- Boolean Operations
- Bitwise Operations
- Conditionals
- Switch

4.29 Boolean Operations

In C++, the boolean operators are used to perform logical operations on boolean values (values that can only be true or false).

4.29.1 AND

And operators can be represented by `&&` (most common syntax) or `and` (C++20 and up - alternative operator representation). This operator represents the logical `AND` operation. It returns `true` if both operands are `true`, and `false` otherwise. - It needs only if one `false` element to make the result be `false`; - It needs all elements to be `true` in order the result be `true`;

p	q	p and q
true	true	true
true	false	false
false	true	false
false	folse	false

For example:

```
bool x = true;
bool y = false;
bool z = x && y; // z is assigned the value false
```

4.29.2 OR

Or operators can be represented by `||` (most common syntax) or `or` (C++20 and up - - alternative operator representation). This operator represents the logical `OR` operation. It returns `true` if one operands are `true`, and `false` if all are `false`. - It needs only if one `true` element to make the result be `true`; - It needs all elements to be `false` in order the result be `false`;

p	q	p or q
true	true	true
true	false	true
false	true	true
false	folse	false

For example:

```
bool x = true;
bool y = false;
bool z = x || y; // z is assigned the value true
```

4.29.3 NOT

Not operator can be represented by `!` (most common syntax) or `not` (C++20 and up - alternative operator representation). This operator represents the logical `NOT` operation. It returns `true` if operand after it is `false`, and `true` otherwise._

p	not p
true	false
false	true

For example:

```
bool x = true;
bool y = !x; // y is assigned the value false
```

4.30 Bitwise operations

In C++, the bitwise operators are used to perform operations on the individual bits of an integer value.

4.30.1 AND

Bitwise `and` can be represented by `&` or `bitand` (C++20 and up - alternative operator representation: This operator performs the bitwise AND operation. It compares each bit of the first operand to the corresponding bit of the second operand, and if both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0. For example:

```
int x = 5; // binary representation is 0101
int y = 3; // binary representation is 0011
int z = x & y; // z is assigned the value 1, which is binary 0001
```

4.30.2 OR

Bitwise `or` can be represented by `|` or `bitor` (C++20 and up - alternative operator representation: This operator performs the bitwise OR operation. It compares each bit of the first operand to the corresponding bit of the second operand, and if either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0. For example:

```
int x = 5; // binary representation is 0101
int y = 3; // binary representation is 0011
int z = x | y; // z is assigned the value 7, which is binary 0111
```

4.30.3 XOR

Bitwise `xor` can be represented by `^` or `bitxor` (C++20 and up - alternative operator representation: This operator performs the bitwise XOR (exclusive OR) operation. It compares each bit of the first operand to the corresponding bit of the second operand, and if the bits are different, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

```
int x = 5; // binary representation is 0101
int y = 3; // binary representation is 0011
int z = x ^ y; // z is assigned the value 6, which is binary 0110
```

Bitwise xor is a type of binary sum without carry bit.

4.30.4 NOT

Bitwise `not` can be represented by `~` or `bitnot` (C++20 and up - alternative operator representation: This operator performs the bitwise NOT (negation) operation. It inverts each bit of the operand (changes 1 to 0 and 0 to 1). For example:

```
int x = 5; // binary representation is 0101
int y = ~x; // y is assigned the value -6, which is binary 11111010. See complement of two for more details.
```

4.30.5 SHIFT

In C++, the shift operators are used to shift the bits of a binary number to the left or right. Pay attention to not mix with the same ones used to strings, in that case they are called stream operators. There are two shift operators:

1. `<<`: This operator shifts the bits of the left operand to the left by the number of positions specified by the right operand. For example:

```
int x = 2; // binary representation is 10
x = x << 1; // shifts the bits of x one position to the left and assigns the result to x
// x now contains 4, which is binary 100
```

1. `>>`: This operator shifts the bits of the left operand to the right by the number of positions specified by the right operand. For example:

```
int x = 4; // binary representation is 100
x = x >> 1; // shifts the bits of x one position to the right and assigns the result to x
// x now contains 2, which is binary 10
```

The shift operators are often used to perform operations more efficiently than can be done with other operators. They can also be used to extract or insert specific bits from or into a value.

4.31 Conditionals

Conditionals are used to branch and execute different blocks of code based on whether a certain condition is true or false. There are several types of conditionals, including:

4.31.1 if clause

`if` statements: These execute a block of code if a certain condition is `true`. If statements usually uses comparison operators or any result that can be transformed as boolean - any number different than `0` is considered `true`, only `0` is considered `false`.

Comparison operator is used to compare the value of two operands. The operands can be variables, expressions, or constants. The comparison operator returns a Boolean value of true or false, depending on the result of the comparison. There are several comparison operators available:

- `==`: returns `true` if the operands are equal;
- `!=`: returns `true` if the operands are not equal;
- `>`: returns `true` if the left operand is greater than the right operand;
- `<`: returns `true` if the left operand is less than the right operand;
- `>=`: returns `true` if the left operand is greater than or equal to the right operand;
- `<=`: returns `true` if the left operand is less than or equal to the right operand;

For example:

```
if (x > y) {
  // code to execute if x is greater than y
}
```

If it appears without scope `{}`, the condition will applied only to the next statement. For example

```
if (x > y)
  doSomething(); // only happens if x > y is evaluated as true
otherThing(); // this will always occur.
```

Inline conditional:

```
if (x > y) doSomething(); // only happens if x > y is evaluated as true
```

```
if (x > y) {doSomething();} // only happens if x > y is evaluated as true
```

A common source of error is adding a `;` after the condition. In this case, the compiler will understand that it is an empty statement and always execute the next statement.

```
if (x > y); // note the inline empty statement here finished with a `;`
  doSomething(); // this will always happen
```

Note

It is preferred to always create scopes with `{}`, but there is no need to have them if you have only one statement that will happen for that condition.

4.31.2 if-else clause

All the explanations from `if` applies here but now we have a fallback case.

`if-else` statements: These execute a block of code if a certain condition is true, and a different block of code if the condition is false. For example:

```
if (x > y) {
  // code to execute if x is greater than y
} else {
  // code to execute if x is not greater than y
}
```

All the explanations about scope on the `if` clause described before, can be applied to the `else`.

4.31.3 Ternary Operator

The ternary operator is also known as the conditional operator. It is used to evaluate a condition and return one value if the condition is true and another value if the condition is false. The syntax for the ternary operator is:

```
condition ? value_if_true : value_if_false
```

For example:

```
int a = 5;
int b = 10;
int min = (a < b) ? a : b;  // min will be assigned the value of a, since a is less than b
```

Here, the condition `a < b` is evaluated to be true, so the value of a is returned. If the condition had been `false`, the value of b would have been returned instead.

The ternary operator can be used as a shorthand for an if-else statement. For example, the code above could be written as:

```
int a = 5;
int b = 10;
int min;
if (a < b) {
  min = a;
} else {
  min = b;
}
```

4.31.4 Switch

`switch` statement allows you to execute a block of code based on the value of a variable or expression. The switch statement is often used as an alternative to a series of if statements, as it can make the code more concise and easier to read. Here is the basic syntax for a switch statement in C++:

```
switch (expression) {
  case value1:
    // code to be executed if expression == value1
    break;
  case value2:
    // code to be executed if expression == value2
    break;
```

```
  // ...
  default:
    // code to be executed if expression is not equal to any of the values
}
```

The expression is evaluated once, and the value is compared to the values in each case statement. If a match is found, the code associated with that case is executed. The break statement is used to exit the switch statement and prevent the code in subsequent cases from being executed. The default case is optional, and is executed if none of the other cases match the value of the expression.

Here is an example of a switch statement that checks the value of a variable x and executes different code depending on the value of x:

```
int x = 2;

switch (x) {
  case 1:
    cout << "x is 1" << endl;
    break;
  case 2:
    cout << "x is 2" << endl;
    break;
  case 3:
    cout << "x is 3" << endl;
    break;
  default:
    cout << "x is not 1, 2, or 3" << endl;
}
```

In this example, the output would be "x is 2", as the value of x is 2.

Note

It's important to note that C++ uses strict type checking, so you need to be careful about the types of variables you use in your conditionals. For example, you can't compare a string to an integer using the `==` operator.

Switch fallthrough

In C++, the `break` statement is used to exit a `switch` statement and prevent the code in subsequent cases from being executed. However, sometimes you may want to allow the code in multiple cases to be executed if certain conditions are met. This is known as a "fallthrough" in C++.

To allow a `switch` statement to fall through to the next case, you can omit the break statement at the end of the case's code block. The code in the next case will then be executed, and the switch statement will continue to execute until a break statement is encountered or the end of the switch is reached.

Here is an example of a switch statement with a fallthrough:

```
int x = 2;

switch (x) {
  case 1:
    cout << "x is 1" << endl;
  case 2:
    cout << "x is 2" << endl;
  case 3:
    cout << "x is 3" << endl;
  default:
    cout << "x is not 1, 2, or 3" << endl;
}
```

In this example, the output would be "x is 2" and "x is 3", as the break statement is omitted in the `case 2` block and the code in the `case 3` block is executed as a result.

It is generally considered good practice to include a `break` statement at the end of each case in a `switch` statement to avoid unintended fallthrough. However, there may be cases where a fallthrough is desired behavior. In such cases, it is important to document the intended fallthrough in the code to make it clear to other programmers.

4.32 Homework

- Do all exercises up to this topic here.
- Coordinates of a Point. In this activity, you will have to code a way to find the quadrant of a given coordinate.

4.33 Outcomes

It is expected for you to be able to solve all questions before this one `1041` on beecrowd. Sort Beecrowd questions from the most solved to the least solved questions here in the link. If you don't, see Troubleshooting. Don`t let your study pile up, this homework is just a small test, it is expected from you to do other questions on Beecrowd or any other tool such as leetcode.

4.34 Troubleshooting

If you have problems here, start a discussion. Nhis is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me. Or visit the tutoring service.

August 30, 2023

January 2, 2023

GitHub 

4.34.1 Comments

4.35 Loops, for, while and goto

Estimated time to read: 16 minutes

A loop is a control flow statement that allows you to repeat a block of code.

4.36 while loop

This loop is used when you want to execute a block of code an unknown number of times, as long as a certain condition is true. It has the following syntax:

Syntax:

```
while (condition) {
    // code block to be executed
}
```

Example:

```
int nums = 10;
while (nums>=0) {
    cout << nums << endl;
    nums--;
}
```

If the block is only one statement, it can be expressed without `{}` s.

Syntax:

```
while (condition)
    // statement goes here
```

Example:

```
int nums = 10;
while (nums>=0)
    cout << nums-- << endl;
```

4.37 do-while loop

This is similar to the `while` loop, but it is guaranteed to execute at least once.

Syntax:

```
do {
    // code block to be executed
} while (condition);
```

Example:

```
int x = 0;
do{
    cout << x << endl;
    x++;
} while(x<10);
```

If the block is only one statement, it can be expressed without `{}` s.

Syntax:

```
do
    // single statement goes here
while (condition);
```

Example:

```
int x = 0;
do
    cout << x++ << endl;
while (x<=10);
```

4.38 for loop

This loop is used when you know in advance how many times you want to execute a block of code.

- The initialization part is executed only once, at the beginning of the loop. It is used to initialize any loop variables.
- The condition is evaluated at the beginning of each iteration of the loop. If the condition is true, the code block inside the loop is executed. If the condition is false, the loop is terminated.
- The increment part is executed at the end of each iteration of the loop. It is used to update the loop variables.

Syntax:

```
for (initialization; condition; step_iteration) {
    // code block to be executed
}
```

Example:

```
for(int i=10; i<=0; i--){
    cout << i << endl;
}
```

If the block is only one statement, it can be expressed without `{}`s.

Syntax:

```
for (initialization; condition; step_iteration)
    // single statement goes here
```

Example:

```
for(int i=10; i<=0; i--)
    cout << i << endl;
```

4.39 range based loops

A range-based loop is a loop that iterates over a range of elements. The declaration type should follow the same type of the elements in the range.

Syntax:

```
for (declaration : range) {
    // code block to be executed
}
```

or

```
for (declaration : range)
    // single statement
```

To avoid explaining arrays and vectors now, assume `v` as an iterable container that can hold multiple elements. I am going to use auto here to avoid explaining this topic any further.

```
auto v = {1, 2, 3, 4, 5}; // an automatically inferred iterable container with multiple elements
for (int x : v) {
    cout << x << " ";
}
```

It is possible to automatically generate ranges

```
#include <ranges>
#include <iostream>
using namespace std;
int main() {
    // goes from 0 to 9. in iota, the first element is inclusive and the last one is exclusive.
    for (int i : views::iota(0, 10))
        cout << i << ' ';
}
```

4.40 Loop Control Statements

4.40.1 `break`

`break` keyword defines a way to break the current loop and end it immediately.

```
// check if it is prime
int num;
cin >> num; // read the number to be checked if is prime or not
bool isPrime = true;
for(int i=2; i<num; i++){
    if(num%i==0){ // check if i divides num
        isPrime = false;
        break; // this will break the loop and prevent further precessing
    }
}
```

4.40.2 `continue`

`continue` keyword is used to skip the following statements of the loop and move to the next iteration.

```
// print all even numbers
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 1)
        continue;
    cout << i << " "; // this statement is skipped if odd numbers
}
```

4.40.3 `goto`

You should avoid `goto` keyword. PERIOD. The only acceptable usage is to break multiple nested loops at the same time. But even in this case, is better to use `return` statement and `functions` that you're going to see later in this course.

The `goto` keyword allows you to transfer control to a labeled statement elsewhere in your code.

Example on how to create a loop using labels and goto. You can create a loop just using labels(anchors) and goto keywords. But this syntax is hard to debug and read. Avoid it at all costs:

```
#include <iostream>
using namespace std;
int main() {
    int i=0;
    start: // this a label named as start.
    cout << i << endl;
    i++;
    if(i<10)
        goto start; // jump back to start
    else
        goto finish; // jump to finish
    finish: // this a label named as finish.
    return 0;
}
```

Example on how to jump over and skip statements:

```
#include <iostream>

int main() {
    int x = 10;

    goto jump_over_this;  // control jumps to the label below

    x = 20;  // this line of code is skipped
```

```
    jump_over_this:  // label for goto statement
    std::cout << x << std::endl;  // outputs 10

    return 0;
}
```

Example of an arguably acceptable use of `goto`. Here you can see the usage of a way to break both loops at the same time. If you use `break`, you will only break the inner loop. In this situation it is better to break your code into functions to reduce complexity and nesting.

```
for (int i = 0; i < imax; ++i)
    for (int j = 0; j < jmax; ++j) {
        if (i + j > elem_max) goto finished;
        // ...
    }
finished:
// ...
```

4.41 Loop nesting

You can nest loops by placing one loop inside another. The inner loop will be executed completely for each iteration of the outer loop. Here is an example of nesting a for loop inside another for loop:

```
for (int i = 0; i < 10; i++) {
  for (int j = 0; j < 5; j++) {
    cout << "i: " << i << " j: " << j << endl;
  }
}
```

4.42 Infinite loops

A infinite loop is when the code loops indefinitely without having a way out. Here goes some examples:

```
while(true)
    cout << "Hello World!" << endl;
```

```
for(;;)
    cout << "Hello World!" << endl;
```

```
int i = 0;
while(i<10); // note the ';' here, it will run indefinitely an empty statement because it won't reach the scope.
{
    cout << i << endl;
    i++;
}
```

4.43 Accumulator Pattern

The accumulator pattern is a way to accumulate values in a loop. Here is an example of how to use it:

```
int fact = 1; // accumulator variable
for(int i=2; i<5; i++){
    fact *= i; // multiply the accumulator by the current value of i
}
// fact = 1*1*2*3*4 = 24
cout << fact << endl;
```

4.44 Search pattern

The search pattern is a way to search for a value in a loop, the most common implementation is a boolean flag. Here is an example of how to use it:

```
int num;
cin >> num; // read the number to be checked if is prime or not
bool isPrime = true; // flag to indicate if the number is prime or not
for(int i=2; i<num; i++){
    if(num%i==0){ // check if i divides num
```

```
        isPrime = false;
        break; // this will break the loop and prevent further precessing
    }
}
cout << num << " is " << (isPrime ? "" : "not ") << "prime" << endl;
// (isPrime ? "" : "not ") is the ternary operator, it is a shorthand for if-else
```

4.45 Debugging

Debugging is the act of instrumentalize your code in order to track problems and fix them.

The most naive way of doing it is by printing variables random texts to find the problem. Don't do it. Use debugger tools instead. Each IDE has his its ows set of tools, if you are using CLion, use this tutorial.

4.46 Automated tests

There are lots of methodologies to guarantee your code is correct and solve the problem it is supposed to solve. The one that stand out is Automated tests.

When you are using beecrowd, leetcode, hackerrank or any other tool to solve problems to learn how to code, a problem is posted to be solved and they test your code solution against a set of expected outputs. This is automated testing. You can generate custom automated tests for your code and cover all cases that you can imagine before you start coding the solution. This is a good practice and is documented in the industry as Test Driven Development.

4.47 Homework

Do all exercises up to this topic here.

In this activity, you will have to solve Fibonacci sequence. You should implement using loops, and variables. Do not use arrays nor closed-form formulas.

- Easy Fibonacci

Optional Readings on Fibonacci Sequence;

Hint: Create two variables, one to store the current value and the previous value. For each iteration step, calculate the sum of both and store and put into a temp variable. Copy the current into the previous and set the current with the temporary you calculated before.

4.48 Outcomes

It is expected for you to be able to solve all questions before this one `1151` on beecrowd. Sort Beecrowd questions from the most solved to the least solved questions here in the link. If you don't, see Troubleshooting. Don`t let your study pile up, this homework is just a small test, it is expected from you to do other questions on Beecrowd or any other tool such as leetcode.

4.49 Troubleshooting

If you have problems here, start a discussion. Nhis is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me. Or visit the tutoring service.

August 30, 2023

January 2, 2023

GitHub 

4.49.1 Comments

4.50 Base Conversion, Functions, Pointers, Parameter Passing

Estimated time to read: 33 minutes

4.51 Base conversion

Data containers use binary coding to store data where every digit can be 0 or 1, this is called base 2, but there are different types of binary encodings and representation, the most common integer representation is Complement of two for representing positive and negative numbers and for floats is IEEE754. Given that, it is relevant to learn how to convert the most used common bases in computer science in order to code more efficiently.

Most common bases are: - Base 2 - Binary. Digits can go from 0 to 1. `{0, 1}`; - Base 8 - Octal. Digits can go from 0 to 7. `{0, 1, 2, 3, 4, 5, 6, 7}`; - Base 10 - Decimal. Digits can go from 0 to 9. `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`; - Base 16 - Hexadecimal. Digits can go from 0 to 9 and then from A to F. `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}`;

4.51.1 Converting from Decimal to any base

There are several methods for performing base conversion, but one common method is to use the repeated division and remainder method. To convert a number from base 10 to another base `b`, you can divide the number by `b` and record the remainder. Repeat this process with the quotient obtained from the previous division until the quotient becomes zero. The remainders obtained during the process will be the digits of the result in the new base, with the last remainder being the least significant digit.

For example, to convert the decimal number 75 to base 2 (binary), we can follow these steps:

```
75 ÷ 2 = 37 remainder 1
37 ÷ 2 = 18 remainder 1
18 ÷ 2 = 9 remainder 0
9 ÷ 2 = 4 remainder 1
4 ÷ 2 = 2 remainder 0
2 ÷ 2 = 1 remainder 0
1 ÷ 2 = 0 remainder 1
```

The remainders obtained during the process (1, 1, 0, 1, 0, 0, 1) are the digits of the result in base 2, with the last remainder (1) being the least significant digit. Therefore, the number 75 in base 10 is equal to 1001011 in base 2.

4.51.2 Converting from any base to decimal

The most common way to convert from any base to decimal is to follow the formula:

$$d_{n-1}*b^{n-1} + d_{n-2}*b^{n-2} + ... + d_1*b^1 + d_0*b^0$$

Where d_x represents the digit at the corresponding position x in the number, n is the number of digits in the number, and b is the base of the number.

For example, to convert the number 1001011 (base 2) to base 10, we can use the following formula:

(1 * 2^6) + (0 * 2^5) + (0 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) = 75

Therefore, the number 1001011 in base 2 is equal to 75 in base 10.

4.52 Functions

A function is a block of code that performs a specific task. It is mostly used to isolate specific reusable functionality from the rest of the code. It has a name, a return type, and a list of parameters. Functions can be called from other parts of the program to execute the task. Here is an example of a simple C++ function that takes two integers as input and returns their sum.

```
int add(int x, int y) {
  int sum = x + y;
```

```
  return sum;
}
```

To call the function, you would use its name followed by the arguments in parentheses:

```
int a = 2, b = 3;
int c = add(a, b); // c will be equal to 5
```

Functions can also be declared before they are defined, in which case they are called "prototypes." This allows you to use the function before it is defined, which can be useful if you want to define the function after it is used. For example:

```
int add(int x, int y);

int main() {
  int a = 2, b = 3;
  int c = add(a, b);
  return 0;
}

int add(int x, int y) {
  int sum = x + y;
  return sum;
}
```

4.53 Reference Declaration

> **Note**
>
> This content only covers an introduction to the topic.

The `&` is used to refer memory address of the variable. When used in the declaration, it is the Lvalue reference declarator. It is an alias to an already-existing, variable, object or function. Read more here.

When used as an prefix operator before the name of a variable, it will return the memory address where the variable is allocated.

Example:

```
string s;

// the variable r has the same memory address of s
// the declaration requires initialization
string& r = s;

s = "Hello";

cout << &s << endl; // prints the variable memory address location. in my machine: "0x7ffc53631cd0"
cout << &r << endl; // prints the same variable memory address location. in my machine: "0x7ffc53631cd0"

cout << s << endl; // prints "Hello"
cout << r << endl; // prints "Hello"

// update the content
r += " world!";

cout << s << endl; // prints "Hello world!"
cout << r << endl; // prints "Hello world!"
```

4.54 Pointer Declaration

> **Note**
>
> This content only covers an introduction to the topic.

The `*` is used to declare a variable that holds the address of a memory position. A pointer is an integer number that points to a memory location of a container of a given type. Read more here.

```
string* r = nullptr; // it is not required do initialize, but it is a good practice to always initialize a pointer pointing to null address (0).
string s = "Hello";
```

```
r = &s; // the variable r stores the memory address of s

cout << s << endl; // prints the content of the variable s. "Hello"
cout << &s << endl; // prints the address of the variable s. in my machine "0x7fffdda021b0"

cout << r << endl;  // prints the numeric value of the address the pointer points, in this case it is "0x7fffdda021b0".
cout << &r << endl; // prints the address of the variable r. it is a different address than s, in my machine "0x7fffdda021d0".
cout << *r << endl; // prints the content of the container that is pointing, it prints "Hello".

string other = "world";
r = &s; // r now points to another variable

cout << *r << endl; // prints the content of the container that is pointing, it prints "world"
```

4.55 `void` type

We covered briefly the `void` type when we covered data types. There are 2 main usages of `void`

`void` is used to specify that some function dont return anything to the caller.

voidFunction.cpp

```
// this function does not need to return anything
// optionally you can use an empty `return` keyword without variable to break the flow early
void doSomething() {
    // function body goes here
    return; // this line is optional, it can be used inside conditional do break early the function flow
}
```

`void*` is used as a placeholder to store a pointer to anything in memory. Use this with extreme caution, because you can easily mess with it and lose track of the type or the conversion. The most common use are: - Access the raw content of a variable in memory; - Low-level raw memory allocation; - Placeholder to act as a pointer to anything;

rawpointer.cpp

```
#include <iostream>
#include <iomanip>
#include <bitset>
using namespace std;
int main()
{
    // declare our data
    float f = 2.0f;
    // point without type that points to the memory location of `f`
    void* p = &f;
    // (int*) casts the void* to int*, so it can be understandable
    // * in front means that we want to fetch the content of what is pointing
    int i = *(int*)(p);
    cout << hex << i << endl; // prints 40000000
    std::bitset<32> bits(i);
    cout << bits << endl; // prints 01000000000000000000000000000000
    return 0;
}
```

4.56 Passing parameter to a function by value

Pass-by-value is when the parameter declaration follows the traditional variable declaration without `&`. A copy of the value is made and passed to the function. Any changes made to the parameter inside the function have don't change on the original value outside the function.

```
#include <iostream>
using namespace std;
void times2(int x) {
    x = x * 2;
    // the value x here is doubled. but it dont change the value outside the scope
}

int main()
{
    int y = 2;
    times2(y); // this dont change the value, it passes a copy to the function
    cout << y << endl;  // output: 2
    return 0;
}
```

4.57 Passing parameter to a function by reference

Pass-by-reference occurs when the function parameter uses the `&` in the parameter declaration. It will allow the function to modify the value of the parameter directly in the other scope, rather than making a copy of the value as it does with pass-by-value. The mechanism behind the variable passed is that it is an alias to the outer variable because it uses the same memory position.

```
#include <iostream>
using namespace std;
void times2(int &x) { // by using &, x has the same address the variable passed where the function is called
  x*=2; // it will change the variable in caller scope
}

int main() {
  int y = 2;
  times2(y);
  cout << y << endl;  // Outputs 4
  return 0;
}
```

4.58 Passing parameter to a function by pointer

Pass-by-pointer occurs when the function parameter uses the `*` in the parameter declaration. It will allow the function to modify the value of the parameter in the other scope via memory pointer, rather than making a copy of the value as it does with pass-by-value. The mechanism behind it is to pass the memory location of the outer variable as a parameter to the function.

```
#include <iostream>
using namespace std;
void times2(int *x) { // by using *, x has the same address the variable passed where the function is called
    // x holds the address of the outer variable
    // *x is the content of what x points.
  *x *= 2; // it will change the variable in caller scope
}

int main() {
  int y = 2;
  times2(&y); // the function expects a pointer, given pointer is an address, we pass the address of the variable here
  cout << y << endl;  // Outputs 4
  return 0;
}
```

4.59 Function overload

A function with a specific name can be overload with different not implicitly convertible parameters.

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

float average(float a, float b){
    return (a + b)/2;
}

float average(float a, float b, float c){
    return (a + b + c)/3;
}

int main(){
    cout << average(1, 2) << endl; // print 1.5
    cout << average(1, 2, 3) << endl; // print 2
    return 0;
}
```

4.60 Default parameter

Functions can have default parameters that should be used if the parameter is not provided, making it optional.

defaultparam.cpp

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

void greet(string username = "user") {
```

```
    cout << "Hello " << mes << endl;
}

int main() {
  // Prints "Hello user"
  greet(); // the default parameter user is used here

  // Prints "Hello John"
  greet("John");

  return 0;
}
```

4.61 Scopes

Scope is a region of the code where a identifier is accessible. A scope usually is specified by what is inside `{` and `}`. The global scope is the one that do not is inside any `{}`.

scope.cpp

```
#include <iostream>
#include <string>
using namespace std;
string h = "Hello"; // this variable is in the global scope
int main() {
  string w = " world"; // this variable belongs to the scope of the main function
  cout << h << w << endl; // both variables are visible and accessible
  return 0;
}
```

Multiple identifiers with same name can not be created in the same scope. But in a nested scope it is possible to shadow the outer one when declared in the inner scope.

variableShadowing.cpp

```
#include <iostream>
#include <string>
using namespace std;
string h = "Hello"; // this variable is in the global scope
int main() {
  cout << h; // will print "Hello"
  string h = " world"; // this will shadow the global variable with the same name h
  cout << h; // will print " world"
  return 0;
}
```

4.62 Lambda functions

In C++, an anonymous function is a function without a name. Anonymous functions are often referred to as lambda functions or just lambdas. They are useful for situations where you only need to use a function in one place, or when you don't want to give a name to a function for some other reason.

```
auto lambda = [](int x, int y) { return x + y; };
// auto lambda = [] (int x, int y) -> int { return x + y; }; // or you can specify the return type
int z = lambda(1, 2);  // z is now 3
```

In this case the only variables accessible by the lambda function scope are the ones passed as parameter `x` and `y`, and works just like a normal function, but it can be declared inside at any scope.

If you want to make a variable available to the lambda, you can pass it via captures, and it can be by-value or by-reference. To capture a variable by value, just pass the variable name inside the `[]`. To capture a variable by reference, you use the `&` operator followed by the variable name inside the `[]`. Here is an example of capturing a variable by value:

```
int x = 1;
auto lambda = [x] { return x + 1; };
```

The value of `x` is copied into the lambda function, and any changes to `x` inside the lambda function have no effect on the original variable.

Here is an example of capturing a variable by reference:

```
int x = 1;
auto lambda = [&x] { return x + 1; };
```

The lambda function has direct access to the original variable, and any changes to `x` inside the lambda function are reflected in the original variable.

You can also capture multiple variables by separating them with a comma. For example:

```
int x = 1, y = 2;
auto lambda = [x, &y] { x += 1; y += 1; return x + y; };
```

This defines a lambda function that captures `x` by-value and `y` by-reference. The lambda function can modify `y` but not `x`.

Lambda captures are a useful feature of C++ that allow you to write more concise and expressive code. They can be especially useful when working with algorithms from the Standard Template Library (STL), where you often need to pass a function as an argument.

In order to capture everything automatically you can either capture by copy `[=]` or by reference `[&]`.

```
// capture everything via copy
int x = 1, y = 2;
auto lambda = [=] {
    // x += 1; // cannot be changed because it is read-only
    // y += 1; // cannot be changed because it is read-only
    return x + y;
};
int c = lambda(); // c will be 5, but x and y wont change their values
```

```
// capture everything via reference
int x = 1, y = 2;
auto lambda = [&] { x += 1; y += 1; return x + y; };
int c = lambda(); // c will be 5, x will be 2, and y will be 3.
```

For a more in depth understanding, go to Manual Reference or check this tutorial.

4.63 Multiple files

In bigger projects, it is useful to split your code in multiple files isolating intention and organizing your code. To do so, you can create a header file with the extension `.h` and a source file with the extension `.cpp`. The header file will contain the declarations of the functions and the source file will contain the definitions of the functions. The header file will be included in the source file and the source file will be compiled together with the main file.

main.cpp

```
#include <iostream>
#include "functions.h"
using namespace std;

int main() {
  cout << sum(1, 2) << endl;
  return 0;
}
```

functions.h

```
// Preprocessor directive (macro) to ensure that this header file is only included once
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

// Function declaration without body
int sum(int a, int b);

#endif
```

Alternatively, you can use `#pragma once` instead of `#ifndef`, `#define` end `#endif` to ensure that the header file is only included once. This is a non-standard preprocessor directive, but it is supported by most compilers. Ex.:

functions.h

```
// Preprocessor directive (macro) to ensure that this header file is only included once
#pragma once

// Function declaration without body
int sum(int a, int b);
```

functions.cpp

```
// include the header file that contains the function declaration
#include "functions.h"

// function definition with body
int sum(int a, int b) {
  return a + b;
}
```

4.64 Preprocessor directives and macros

In C++, the preprocessor is a text substitution tool. It runs before compiling the code. It scans a program for special commands called preprocessor directives, which begin with a `#` symbol. When it finds a preprocessor directive, it performs the specified text substitutions before the program is compiled.

The most common preprocessor directive is `#include`, which tells the preprocessor to include the contents of another file in the current file. The included file is called a header file, and commonly has a `.h` extension. For example:

```
#include <iostream>
```

Another extensively used macro is `#define`, which defines a macro. A macro is a symbolic name for a constant value or a small piece of code. For example:

```
#define PI 3.14159
```

It will replace all occurrences of `PI` with `3.14159` before compiling the code. But pay attention that is not recommended to use macros for constants, because they are not type safe and can cause unexpected behavior. It is recommended to declare `const` variable instead.

See more about some cases against macros here:

- https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#enum1-prefer-enumerations-over-macros
- https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es30-dont-use-macros-for-program-text-manipulation
- https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es31-dont-use-macros-for-constants-or-functions

Nowadays the best use case for macros are for conditional compilation or platform specification. For example:

```
#define DEBUG 1

int main() {
  #if DEBUG
    std::cout << "Debug mode" << std::endl;
  #else
    std::cout << "Release mode" << std::endl;
  #endif
}
```

Another example is to define the operating system:

```
#ifdef _WIN32
  #define OS "Windows"
#elif __APPLE__
  #define OS "MacOS"
#elif __linux__
  #define OS "Linux"
#else
  #define OS "Unknown"
#endif
```

```
int main() {
  std::cout << "OS: " << OS << std::endl;
}
```

4.65 Homework

- Do all exercises up to this topic here.
- Hexadecimal converter. In this activity, you will have to code a way to find the convert to hexadecimal without using any std library to do it for you. DON'T USE `std::hex`.

4.66 Outcomes

It is expected for you to be able to solve all questions before this one `1957` on beecrowd. Sort Beecrowd questions from the most solved to the least solved questions here in the link. If you don't, see Troubleshooting. Don`t let your study pile up, this homework is just a small test, it is expected from you to do other questions on Beecrowd or any other tool such as leetcode.

4.67 Troubleshooting

If you have problems here, start a discussion. Nhis is publicly visible and not FERPA compliant. Use discussions in Canvas if you are enrolled in a class with me. Or visit the tutoring service.

August 30, 2023

January 4, 2023

GitHub 

4.67.1 Comments

4.68 Streams and File IO

Estimated time to read: 12 minutes

At this point, you already are familiar with the `iostream` header. But we never discussed what it is properly. It is a basic stream and it has two static variable we already use: `cin` for reading variables from the console input and `cout` to output things to console, see details here. It is possible to interact with all streams via the `>>` and `<<` operators.

But C++ have 2 other relevant streams that we need to cover: `fstream` and `sstream`.

4.69 File streams

File streams are streams that target files instead of the terminal console. The `fstream` header describes the file streams and the ways you can interact with it.

The main differences between console and file streams are: - You have to target the filesystem path for files because we can manage different files at the same, but for console, you only have one, so you dont need to target which console we are streaming. In order to not mess each target, you have to declare a different variable to store the target and state. - Files are persistent, so if you write something to them, and try to read from it again, the that will be there saved.

Files are a kind of resource managed by the operation system. So every time you request something to be read or write, behind the scenes you are requesting something to the operating system, and it can be slow or subject by lock control. When you open a file to be read or write, the OS locks it to avoid problems. You can open a file to be read multiple times simultaneously, but you cannot write more than once. So to avoid problems, after reading or writing the file, you should close the file.

```
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

int main() {
  // Open the file
  // this file path is relative to the executable, so be assured it exists in the same folder the executable is placed
  // fin is the variablename and it is function initialized via a file path to target, but it can be any valid identifier
  // I am using fin as variable to follow the same metaphor `fin` as `file input` as we have with console input `cin`,
  ifstream fin("file.txt");

  // Check if the file is open
  // it is a good practice to check if the file is really there before doing anything
  if (!fin.is_open()) {
    cerr << "Error opening file" << endl;
    return 1; // quits the program with an error code
  }

  // Read the contents of the file line by line
  string line;
  // getline can target streams in general, so you can pass the file stream as a target
  while (getline(fin, line)) { // while the file have lines, read and store the content inside the line variable
    cout << line << endl; // output each string into the console
  }

  // Close the file
  fin.close();

  return 0;
}
```

4.70 String Stream

The `sstream` header describes string stream, which is a type of memory stream and is very useful to do string manipulation. For our intent, we aro going to focus 3 types of memory streams.

- `ostringstream`: works just like `cout` but the content will printed to a memory region.
- it is more efficient to build a complex string in this way than `cout`ing multiple times;
- `istringstream`: works just like `cin` but it will read from a memory area.
- it is safer to read from a closed memory area than, and you ran reset the reading pointer to re-read previous elements easier than with `cin`.

```
#include <iostream>
#include <sstream>
using namespace std;
int main() {
    ostringstream oss; // declare the output stream
    // print numbers from 0 to 100
    for(int i=0; i<=100; i++)
        oss << i << ' '; // store the data into memory
    cout << oss.str(); // convert the stream into a string to be printed all at once
}
```

```
#include <iostream>
#include <sstream>
using namespace std;
int main() {
    // read input
    string input;
    getline(cin, input);

    // initialize string stream with the content from a console line
    istringstream ss(input); // declare the stream to read from

    // extract input
    string name;
    string course;
    string grade;

    iss >> name >> course >> grade;
}
```

You can combine string stream and file stream to read a whole file and store into a single string.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main() {
  // Open the file
  ifstream file("file.txt");

  // Check if the file is open
  if (!file.is_open()) {
    cerr << "Error opening file" << endl;
    return 1;
  }

  // Read the contents of the file into a stringstream
  stringstream ss;
  ss << file.rdbuf(); // read the whole file buffer and stores it into a string stream

  // Close the file
  file.close();

  // Convert the stringstream into a string
  string contents = ss.str();

  cout << contents << endl; // prints the whole file at once

  return 0;
}
```

4.71 Homework

You have the job of creating a small program to read a file image in the format PGMA and inverse the colors as a negative image.

You can test your code with different images if you want. You can download more images here. But here goes 2 examples:

- Sample input easy: baboon.ascii.pgm - max intensity is not 255 and don't have comments.
- Sample input harder: lena.ascii.pgm - have comments, and the max intensity is different than 255.

You can test if your output file is correct using this tool. You can open this file via any text reader, use the online viewer, or use any app that reads pnm images.

4.71.1 Attention:

- To create the inverse image, you should read the file header and search for the maximum intensity. You should use this number as a base to inverse. In the Lena case, it is 245.
- You should pay attention that every line shouldn't be bigger than 70 chars;
- Pay attention that the line 2 might exists or not. And any comment found in the file should be skipped.

The user should input the filename to be read. So you should store it into a string variable. The output filename should be the same as the input but with '.inverse' concatenated in the end. Ex.: lena.pgm becomes lena.inverse.pgm; If you find this too complicated, just concatenate with .inverse.pgm would be acceptable. ex.: lena.pgm becomes lena.pgm.inverse.pgm

In order for your program to find the file to be read, you should provide the fullpath to the file or simply put the file in the same folder your executable is.

HINT: In order to find comments and ignore them do something like that:

```
string widthstr;
int width;
fin >> widthstr;
if(widthstr.at(0)=='#')
    getline(fin, widthstr); // ignore line
else
    width = stoi(widthstr); // covert string to integer
```

August 30, 2023

January 4, 2023

GitHub 

4.71.2 Comments

4.72 Arrays

Estimated time to read: 23 minutes

An array is a collection of similar data items, stored in contiguous memory locations. The items in an array can be of any built-in data type such as int, float, char, etc. An array is defined using a syntax similar to declaring a variable, but with square brackets indicating the size of the array.

Here's an example of declaring an array of integers with a size of 5:

```
int arr[5]; // declare an array of size 5 at the stack
```

The above declaration creates an array named arr of size 5, which means it can store 5 integers. The array elements are stored in contiguous memory locations, which means the next element is stored at the immediate next memory location. The first element of the array is stored at the 0^{th} index, the second element at the 1^{st} index, and so on up to 4. Between 0 an 4 all inclusive we have 5 elements.

This creates an array called "myArray" that can hold 5 integers. The first element of the array is accessed using the index 0, and the last element is accessed using the index 4. You can initialize the array elements during declaration by providing a comma-separated list of values enclosed in braces:

```
int myArr[5] = {10, 20, 30, 40, 50}; // initialize the array with 5 elements
```

In this case, the first element of the array will be 10, the second element will be 20, and so on.

You can also use loops to iterate over the elements of an array and perform operations on them. For example:

```
for (int i = 0; i < 5; i++) {
  myArray[i] *= 2;
}
```

This loop multiplies each element of the "myArray" by 2.

Arrays are a useful data structure in C++ because they allow you to store and manipulate collections of data in a structured way. However, they have some limitations, such as a fixed size that cannot be changed at runtime, and the potential for buffer overflow if you try to access elements beyond the end of the array.

4.73 Buffer overflow

A buffer overflow occurs when a program attempts to write more data to a fixed-size buffer than it can hold. This can happen when a program attempts to write more data to a buffer than the buffer can hold, or when a program attempts to read more data from a buffer than the buffer contains. This can happen when a program attempts to write more data to a buffer than the buffer can hold, or when a program attempts to read more data from a buffer than the buffer contains.

A buffer overflow can be caused by a number of different factors, including:

- A program that attempts to write more data to a buffer than the buffer can hold
- A program that attempts to read more data from a buffer than the buffer contains

Buffer overflow vulnerabilities are a common type of security vulnerability, as they can be exploited by malicious attackers to execute arbitrary code or gain unauthorized access to a system. To prevent buffer overflow vulnerabilities, it's important to carefully manage memory allocation and use bounds checking functions or techniques such as using safe C++ library functions like `std::vector` or `std::array`, and ensuring that input data is properly validated and sanitized.

4.74 Multi-dimensional arrays

A multi-dimensional array is an array of arrays. For example, a 2-dimensional array is an array of arrays, where each element of the array is itself an array. A 3-dimensional array is an array of 2-dimensional arrays, where each element of the array is itself a 2-dimensional array. And so on.

For example, to declare a two-dimensional array with 3 rows and 4 columns of integers, you would use the following code:

```
int arr[3][4]; // Declare a 2-dimensional array with 3 rows and 4 columns at the stack
```

You can access elements in a multidimensional array using multiple sets of square brackets. For example, to access the element at row 2 and column 3 of myArray, you would use the following code:

```
int element = myArray[1][2]; // Access the element at row 2 and column 3
```

In C++, you can have arrays with any number of dimensions, but keep in mind that as the number of dimensions increases, it becomes more difficult to manage and visualize the data.

4.75 Array dynamic allocation

In some cases, you dont know the size of the array at compile time. In this case, you can use dynamic memory allocation to allocate the array at runtime. This is done using the `new` operator, which allocates a block of memory on the heap and returns a pointer to the beginning of the block. For example, to allocate an array of 5 integers on the heap, you would use the following code:

```
int *arr = new int[5]; // Allocate a block of memory on the heap
```

The above code allocates a block of memory on the heap that is large enough to hold 5 integers. The `new` operator returns a pointer to the beginning of the block, which is assigned to the pointer variable `arr`. You can then use the pointer to access the elements of the array. You can access individual elements of the array using the array subscript notation:

```
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

When you are done using the array, you should free the memory using the `delete` operator. For example, to free the memory allocated to the array in the previous example, you would use the following code:

```
delete[] arr; // Free the memory by telling the operation system you are done with it
arr = nullptr; // Reset the pointer to null to avoid dangling pointers and other bugs
```

The `delete` operator takes a pointer to the beginning of the block of memory to free. The `[]` operator is used to indicate that the block of memory contains an array, and that the `delete` operator should free the entire array.

4.75.1 Dynamic allocation of multi-dimensional arrays

In the case of dynamically allocate memory for a multidimensional array, first you have to understand that in the same way you can have an array of arrays, you can have a pointer to a pointer. This is called a double pointer. So, if you want to allocate a 2-dimensional array dynamically, you can do it like this:

```
int lines, columns;
cin >> lines >> columns;
int **arr = new int*[lines]; // Allocate an array of pointers to pointers
for (int i = 0; i < lines; i++) {
  arr[i] = new int[columns]; // Allocate an array of integers for each pointer
}
// do stuff with the array
for (int i = 0; i < lines; i++) {
  delete[] arr[i]; // Free the memory for each array of integers
}
delete[] arr; // Free the memory for the array of pointers
```

4.76 Smart pointers to rescue

You probably noticed the number of bugs and vulnerabilities that can be caused by improper memory management. To help address that, C++ introduced smart pointers. The general purpose smart contract you will be mostly using is `shared_ptr` that in the end of the scope and when all references to it become 0 will automatically free the memory. The other smart pointers are `unique_ptr` and `weak_ptr` that are used in more advanced scenarios. But for now, we will focus on `shared_ptr`.

In C++11, smart pointers were introduced to help manage memory allocation and deallocation. Smart pointers are classes that wrap a pointer to a dynamically allocated object and provide additional features such as automatic memory management. The most commonly used smart pointers are `std::unique_ptr` and `std::shared_ptr`. The `std::unique_ptr` class is a smart pointer that owns and manages another object through a pointer and disposes of that object when the `std::unique_ptr` goes out of scope. The `std::shared_ptr` class is a smart pointer that retains shared ownership of an object through a pointer. Several `std::shared_ptr` objects may own the same object. The object is destroyed and its memory deallocated when either of the following happens:

- the last remaining `std::shared_ptr` owning the object is:
- destroyed
- is assigned another pointer via `operator=` or `reset()`
- is reset or released
- moved from
- is swapped with another `std::shared_ptr` using `swap()`
- the function `std::shared_ptr::swap()` is called with the last remaining `std::shared_ptr` owning the object as an argument
- the object is no longer reachable from the program (for example, when the program terminates)
- the program:
- throws an exception that is not caught within the same thread
- calls terminating calls such as `std::terminate()`, `std::abort()`, `std::exit()`, or `std::quick_exit()`

To create a dynamic array of int using shared pointers, you can use the std::shared_ptr class template. Here's an example:

```
#include <memory> // for std::shared_ptr
std::shared_ptr<int[]> arr(new int[5]);
```

This creates a shared pointer to an array of 5 integers. The `new int[5]` expression dynamically allocates memory for the array on the heap, and the shared pointer takes ownership of the memory. When the shared pointer goes out of scope, the memory is automatically freed.

You can access individual elements of the array using the array subscript notation, just like with a regular C-style array:

```
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;
```

To deallocate the memory, you don't need to call delete[] explicitly, because the shared pointer takes care of it automatically. When the last shared pointer that points to the array goes out of scope or is explicitly reset, the memory is deallocated automatically:

```
arr.reset(); // deallocates the memory and reset the shared pointer to null to avoid dangling pointers and other bugs
```

Shared pointers provide a convenient and safe way to manage dynamic memory in C++, because they automatically handle memory allocation and deallocation, and help prevent memory leaks and dangling pointers.

Smart pointers are no silver bullet. They are not a replacement for proper memory management, but they can help you avoid common memory management bugs and vulnerabilities. For example, smart pointers can help you avoid memory leaks, dangling pointers, and double frees. They can also help you avoid buffer overflow vulnerabilities by providing bounds checking functions.

4.77 Passing arrays to functions

You can pass arrays to functions in C++ in the same way that you pass any other variable to a function. For example, to pass an array to a function, you would use the following code:

```
void printArray(int arr[], int size) // Pass the array by reference to avoid copying the entire array
{
    for (int i = 0; i < size; ++i)
        std::cout << arr[i] << ' ';
    std::cout << '\n';
}
```

Alternativelly you can pass the array as a pointer:

```
void printArray(int *arr, int size)
{
    for (int i = 0; i < size; ++i)
        std::cout << arr[i] << ' ';
    std::cout << '\n';
}
```

If you want to pass a two dimension array, you can do it in multiple ways:

```
void printArray(int rows, int columns, int **arr); // Pass the array as a pointer of pointers
```

This approach is problematic as you can see it in depth here. It does not check for types and it is not safe. You can also pass the array as a pointer to an array:

```
void printArray(int rows, int arr[][10]); // if you know the number of columns and it is fixed, in this case 10
```

```
void printArray(int rows, int (*arr)[10]); // if you know the number of columns and it is fixed, in this case 10
```

```
void printArray(int arr[10][10]); // if you know the number of rows and columns and they are fixed, in this case both 10
```

There is others ways to pass arrays to functions, such as **templates** but they are more advanced and **we will not cover them now**.

4.78 EXTRA: Standard Template Library (STL)

Those are the most common data structures that you will be using in C++. But it is outside the scope of this course to cover them in depth. So we will only give entry-points for you to learn more about them.

4.78.1 Arrays

If you are using fixed sized arrays, and want to be safe to avoid problems related to out of bounds, you should use the STL arrays. It is a template class that encapsulates fixed size arrays and adds protections for it. It is a safer alternative to C-style arrays. Read more about it here.

4.78.2 Vectors

Vectors are the safest way to deal with dynamic arrays in C++, the cpp core guideline even states that you should use it whenever you can. Vector is implemented in the standard template library and provide a lot of useful functions. Read more about them here.

4.79 Extra curiosities

Context on common bugs and vulnerabilities: - Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses - US Government enforces cyber security requirements - https://en.cppreference.com/w/cpp/language/array

August 30, 2023

January 4, 2023

GitHub 

4.79.1 Comments

4.80 Recursion

Estimated time to read: 7 minutes

Recursion is a method of solving problems where the solution depends on solutions to smaller instances of the same problem. It is a common technique used in computer science, and is one of the central ideas of functional programming. Let's explore recursion by looking at some examples.

You have to be aware that recursion isn't always the best solution for a problem. Sometimes it can be more efficient to use a loop and a producer-consumer strategy instead of recursion. But, in some cases, recursion is the more elegant solution.

When you call functions inside functions, the compiler will store the return point, value and variables on the stack, and it has limited size. Each time you call a function, it is added to the top of the stack. When the function returns, it is removed from the top of the stack. The last function to be called is the first to be returned. This is called the call stack. A common source of problems in programming is when the call stack gets too big. This is called a stack overflow. This is why you should be careful when using recursion.

4.80.1 Fibonacci numbers

The Fibonacci numbers are a sequence of numbers where each number is the sum of the two numbers before it. The constraints are: the first number is 0, the second number is 1, it only run on integers and it is not negative. The sequence looks like this:

```
int fibonacci(int n) {
    // base case
    if (n == 0 || n == 1)
        return n;
    else // recursive case
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

4.80.2 Factorial numbers

The factorial of a number is the product of all the numbers from 1 to that number. It only works for positive numbers greater than 1.

```
int factorial(int n) {
    // base case
    if (n <= 1)
        return 1;
    else // recursive case
        return n * factorial(n - 1);
}
```

4.81 Divide and Conquer

Divide and conquer is a method of solving problems by breaking them down into smaller subproblems. It is extensively used to reduce the complexity of some algorithms and increase readability.

4.81.1 Binary search

Imagine that you already have a sorted array of numbers and you want to find the location of a specific number in that array. You can use a binary search to find it. The binary search works by dividing the array in half and checking if the number you are looking for is in the first half or the second half. If it is in the first half, you repeat the process with the first half of the array. If it is in the second half, you repeat the process with the second half of the array. You keep doing this until you find the number or you know that it is not in the array.

```
// recursive binary search on a sorted array to return the position of a number
int binarySearch(int arr[], int start, int end, int number) {
    // base case
    if (start > end)
        return -1; // number not found
    else {
        // recursive case
```

```
        int mid = (start + end) / 2;
        // return the middle if wi find the number
        if (arr[mid] == number)
            return mid;
        // if the number is smaller than the middle, search in left side
        else if (arr[mid] > number)
            return binarySearch(arr, start, mid - 1, number);
        // if the number is bigger than the middle, search in right side
        else
            return binarySearch(arr, mid + 1, end, number);
    }
}
```

Binary search plays a fundamental role in Newton's method, which is a method to find and approximate the result of complex mathematical functions such as the square root of a number. Binary-sort is extensively used in sorting algorithms such as quick sort and merge sort.

4.81.2 Merge sort

Please refer to the Merge sort section in the sorting chapter.

August 30, 2023

December 27, 2022

GitHub 

4.81.3 Comments

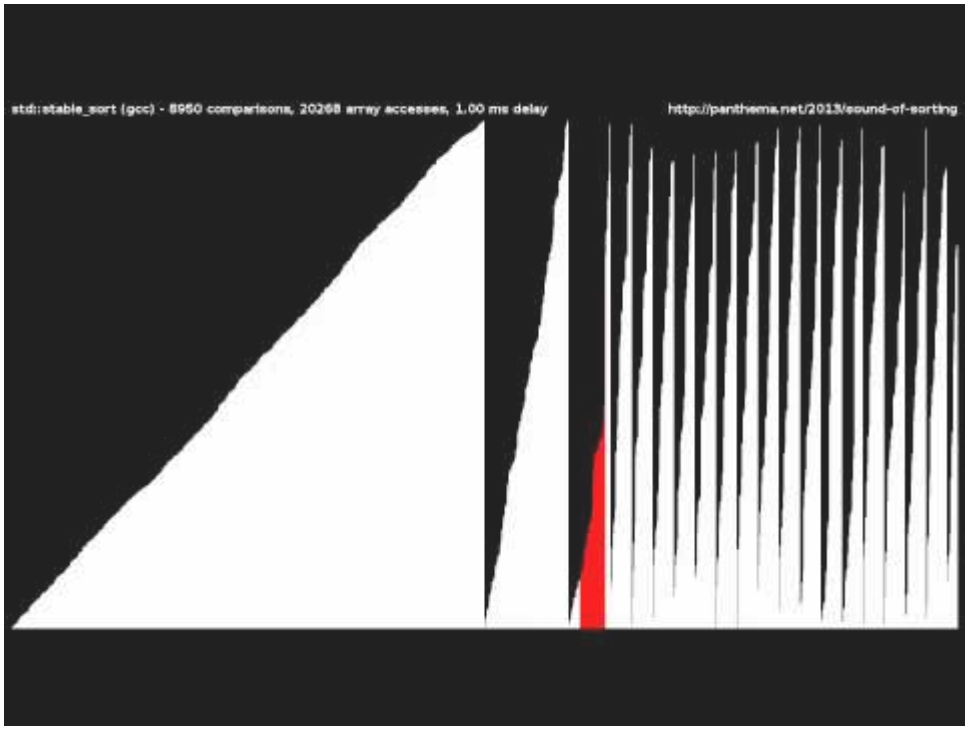
4.82 Sorting algorithms

Estimated time to read: 23 minutes

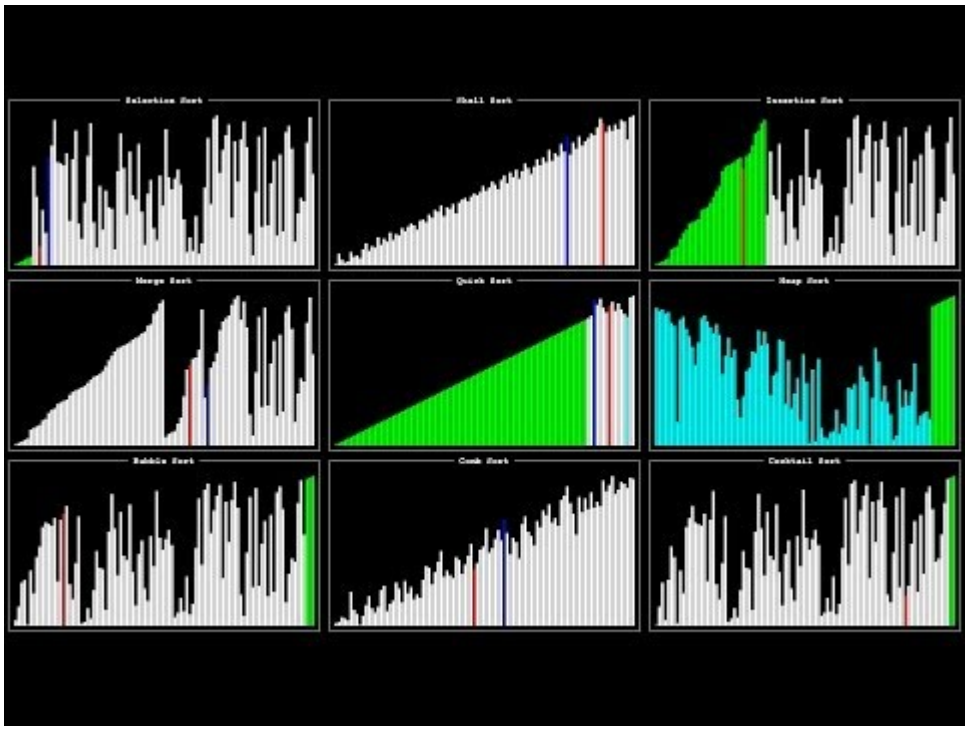
TODO: Note for my furune self: add complete example of how to use those algorithms

Sorting are algorithms that put elements of a list in a certain order. It is cruxial to understand the basics of sorting in order to start understanding more complex algorithms and why you have to pay attention to efficiency.

Before going deep, please watch this video:

SEIZURE WARNING!! 

and this one:



Explore the concepts interactively at visualgo.net.

Try to answer the following questions, before continuing:

- What are the slowest sorting algorithms?
- What are the fastest sorting algorithms?
- Con you infer the difference between a stable and unstable sorting algorithm?
- What is the difference between a comparison and a non-comparison sorting algorithm?
- What would be an in-place and a non-in-place sorting algorithm?
- What is the difference between a recursive and a non-recursive sorting algorithm?

4.82.1 The basics

Many of the algorithms will have to swap elements from the array, vector or list. In order to do that, we will need to create a function that swaps two elements. Here is the function:

```
// A function to swap two elements
void swap(int *xp, int *yp) {
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

The `*` operator used in the function signature means that the function will receive a pointer to an integer. So it will efectivelly change the content in another scope. The `*` operator is used to dereference a pointer, which means that it will return the value stored in the memory address pointed by the pointer. Given the declaration is `int *xp`, the `*xp` will return the value stored in the memory address pointed by `xp`.

Alternatively you could use the `&` operator to pass the reference to that variable in the similar fashion, but the usage wont be requiring the `*` before the variable name as follows:

```
// A function to swap two elements
void swap(int &xp, int &yp) {
    int temp = xp;
    xp = yp;
    yp = temp;
}
```

The result is the same, but the usage is different. The first one is more common in C++, while the second one is more common in C.

4.82.2 Bubble sort

Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

```
// A function to implement bubble sort
void bubbleSort(int arr[], int n) {
    // if the array has only one element, it is already sorted
    if(n<=1)
        return;

    int i, j;
    for (i = 0; i < n-1; i++)
        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}
```

As you can see, the algorithm is very simple, but it is not very efficient. It has a time complexity of O(n^2) and a space complexity of O(1).

One of the drawbacks of this algorithm is the sheer amount of swaps. In the worst scenario, it does n^2 swaps, which is a lot. If your machine have slow writes, it will be very slow.

4.82.3 Insertion sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands. You pick one card and insert it in the correct position in the sorted part of the list. You repeat this process until you have sorted the whole list. Here is the code:

```
// A function to implement insertion sort
void insertionSort(int arr[], int n) {
    // if the array has only one element, it is already sorted
    if(n<=1)
        return;

    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

It falls in the same category of algorithms that are very simple, but not very efficient. It has a time complexity of O(n^2) and a space complexity of O(1).

Although it have the same complexity as bubble sort, it is a little bit more efficient. It does less swaps than bubble sort, but it is still not very efficient. It will swap all numbers to the left of the current number, which is a lot of swaps.

4.82.4 Selection sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right. Here is the code:

```
// A function to implement selection sort
void selectionSort(int arr[], int n) {
    // if the array has only one element, it is already sorted
    if(n<=1)
```

```
        return;

    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++) {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

It is also a simple algorithm, but it is a little bit more efficient than the previous two. It has a time complexity of O(n^2) and a space complexity of O(1).

It does less swaps than the previous two algorithms, potentially n swaps, but it is still not very efficient. It selects for the current position, the smallest number to the right of it and swaps it with the current number. It does this for every number in the list, which fatally a lot of swaps.

4.82.5 Merge sort

Merge sort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. Here is the code:

```
// recursive merge sort
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

// merge function
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    // allocate memory for the sub arrays
    int *L = new int[n1];
    int *R = new int[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
```

```
        arr[k] = R[j];
        j++;
        k++;
    }

    // deallocate memory
    delete[] L;
    delete[] R;
}
```

It is a very efficient algorithm that needs extra memory to work. It has a time complexity of O(n*log(n)) and a space complexity of O(n). It is a very efficient algorithm, but it is not very simple. It is quite more complex than the previous algorithms. It is a divide and conquer algorithm, which means that it divides the problem in smaller problems and solves them. It divides the list in two halves, sorts them and then merges them. It does this recursively until it has a list of size 1, which is sorted. Then it merges the lists and returns the sorted list.

4.82.6 Quick sort

Quick sort is a divide and conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. Here is the code:

```
// recursive quick sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// partition function
int partition (int arr[], int low, int high) {
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++) {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot) {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

It is a very efficient algorithm that don't needs extra memory, which means it is in-place. In average, it can be as fast as mergesort with time complexity of O(n*log(n)), but in the worst case it can be as slow as O(n^2). But it is a better choice if you are not allowed to use extra memory. It is a divide and conquer algorithm, which means that it divides the problem in smaller problems and solves them. It selects a pivot and partitions the list around the pivot. It does this recursively until it has a list of size 1, which is sorted. Then it merges the lists and returns the sorted list.

4.82.7 Counting sort

Counting sort is a specialized algorithm for sorting numbers. It only works well if you have a small range of numbers. It counts the number of occurrences of each number and then uses the count to place the numbers in the right position. Here is the code:

```
// counting sort
void countingSort(int arr[], int n) {
    // if the array has only one element, it is already sorted
    if(n<=1)
        return;

    int max=arr[0];
    int min[0];

    // find the max and min number
    for(int i=0; i<n; i++) {
        if(arr[i]>max) {
            max=arr[i];
        }
        if(arr[i]<min) {
```

```
            min=arr[i];
        }
    }

    // allocate memory for the count array
    int *count = new int[max-min+1];

    // initialize the count array
    for(int i=0; i<max-min+1; i++) {
        count[i]=0;
    }

    // count the number of occurrences of each number
    for(int i=0; i<n; i++) {
        count[arr[i]-min]++;
    }

    // place the numbers in the right position
    int j=0;
    for(int i=0; i<max-min+1; i++) {
        while(count[i]>0) {
            arr[j]=i+min;
            j++;
            count[i]--;
        }
    }

    // deallocate memory
    delete[] count;
}
```

Counting sort is a very efficient sorting algorithm which do not rely on comparisons. It has a time complexity of O(n+k) where k is the range of numbers. Space complexity is O(k) which means it is not an in-place sorting algorithm. It is a very efficient algorithm, but it is not very simple. It counts the number of occurrences of each number and then uses the count to place the numbers in the right position.

4.82.8 Radix sort

Radix sort is a specialized algorithm for sorting numbers. It only works well if you have a small range of numbers. It sorts the numbers by their digits. Here is the code:

```
// Radix sort
void radixSort(int arr[], int n) {
    // if the array has only one element, return
    if(n<=1)
        return;

    // initialize the max number as the first number.
    int max=arr[0];

    // find the max number
    for(int i=0; i<n; i++) {
        if(arr[i]>max) {
            max=arr[i];
        }
    }

    // allocate memory for the count array
    int *count = new int[10]; // 10 digits

    // allocate memory for the output array
    int *output = new int[n];

    // do counting sort for every digit
    for(int exp=1; max/exp>0; exp*=10) {
        // initialize the count array
        for(int i=0; i<10; i++) {
            count[i]=0;
        }

        // count the number of occurrences of each number
        for(int i=0; i<n; i++) {
            count[(arr[i]/exp)%10]++;
        }

        // change count[i] so that count[i] now contains actual position of this digit in output[]
        for(int i=1; i<10; i++) {
            count[i]+=count[i-1];
        }

        // build the output array
        for(int i=n-1; i>=0; i--) {
            output[count[(arr[i]/exp)%10]-1]=arr[i];
            count[(arr[i]/exp)%10]--;
        }

        // copy the output array to the input array
```

```
        for(int i=0; i<n; i++) {
            arr[i]=output[i];
        }
    }
}
```

Radix sort is just a counting sort that is applied to every digit. It has a time complexity of O(n*k) where k is the number of digits.

4.82.9 Conclusion

This is the first time we will talk about efficiency, and for now on, you will start evaluating and taking care about your algorithms' efficiency. You will learn more about efficiency in the next semester and course when we cover data structures.

August 30, 2023

December 27, 2022

GitHub 

4.82.10 Comments

5. Adv. Programming

5.1 Advanced Programming

Estimated time to read: 6 minutes

This course builds on the content from Introduction to Programming. Students study the Object Oriented Programming (OOP) Paradigm with topics such as objects, classes, encapsulation, abstraction, modularity, inheritance, and polymorphism. Students examine and use structures such as arrays, structs, classes, and linked lists to model complex information. Pointers and dynamic memory allocation are covered, as well as principles such as overloading and overriding. Students work to solve problems by selecting implementation options from competing alternatives.

Warning

This is a work in progress, and the schedule is subject to change. Every change will be communicated in class. Use the github repo as the source of truth for the schedule and materials. The materials provided in canvas are just a copy for archiving purposes and might be outdated.

5.1.1 Schedule

Relevant dates for the Fall 2023 semester:

- 09-13 Oct 2023 - Midterms Week
- 20-24 Nov 2023 - Thanksgiving Break
- 11-15 Dec 2023 - Finals Week

🔰 Introduction

- Week 1. 2023/08/28
- Topic: Introduction and tooling
- Formal Assignment:
- Interactive Assignment:

🤖 Structs and Classes

- Week 2. 2023/09/04
- Topic: Stcucts and Classes. Objects, classes, member functions, constructors, destructors.
- Formal Assignment:
- Interactive Assignment:

🔳 Pointers

- Week 3. 2023/09/11
- Topic: More about OOP. Private member functions, object passing, object composition, structs and unions
- Formal Assignment:
- Interactive Assignment:

🔢 Pointer operations

- Week 4. 2023/09/18
- Topic: Pointers - address operator, pointer variables, arrays and pointers, pointer math, pointers as function parameters and return types, dynamic memory allocation
- Formal Assignment:
- Interactive Assignment:

🔢 Pointers continued

- Week 5. 2023/09/25
- Topic: Pointers continued, this pointer, constant member functions, static members, friends, member-wise assignment, copy constructors
- Formal Assignment:
- Interactive Assignment:

:octicons-number-32:{ .lg .middle } Operators / Review

- Week 6. 2023/10/02
- Topic: Operator overloading, type conversion operators, convert constructors, aggregation and composition, namespaces
- Formal Assignment:
- Interactive Assignment:

:octicons-number-40:{ .lg .middle } Midterms

- Week 7. 2023/10/09
- Topic: Midterms / Review
- Formal Assignment:
- Interactive Assignment:

:octicons-number-48:{ .lg .middle } Vectors and Arrays

- Week 8. 2023/10/16
- Topic: Vectors and arrays of objects. Linked lists, linked list operations
- Formal Assignment:
- Interactive Assignment:

:octicons-number-56:{ .lg .middle } Thanksgiving break

- Week 9. 2023/10/23
- Topic: Thanksgiving Break - No Class
- Formal Assignment:
- Interactive Assignment:

:octicons-number-64:{ .lg .middle } Inheritance

- Week 10. 2023/10/30
- Topic: Inheritance, protected members, constructors/destructors
- Formal Assignment:
- Interactive Assignment:

:octicons-number-72:{ .lg .middle } Inheritance continued

- Week 11. 2023/11/06
- Topic: Inheritance hierarchies, polymorphism and virtual member functions, abstract base classes and pure virtual functions
- Formal Assignment:
- Interactive Assignment:

:octicons-number-80:{ .lg .middle } Exceptions

- Week 12. 2023/11/13
- Topic: Exceptions, macros, function and class templates, STL and STL containers, iterators
- Formal Assignment:
- Interactive Assignment:

:octicons-number-88:{ .lg .middle } Project introduction / Work sessions

- Week 13. 2023/11/20
- Topic: Project introduction. (Extra): Stack and Queue
- Formal Assignment:
- Interactive Assignment:

:octicons-number-96:{ .lg .middle } Project work

- Week 14. 2023/11/27
- Topic: Project work
- Formal Assignment:
- Interactive Assignment:

:octicons-number-104:{ .lg .middle } Project Presentations

- Week 15. 2023/12/04
- Topic: Project work
- Formal Assignment:
- Interactive Assignment:

:octicons-number-112:{ .lg .middle } Finals

- Week 16. 2023/12/11
- Topic: Finals
- Formal Assignment:
- Interactive Assignment:

August 30, 2023

April 24, 2023

GitHub 

5.1.2 Comments

6. Portfolio

6.1 Portfolio

Estimated time to read: 2 minutes

Main resources for the course.

6.1.1 Schedule

Week	Date	Topic
1	2023/01/16	Introduction
2	2023/01/23	Case Studies
3	2023/01/30	Game Developer Portfolio Structure
4	2023/02/06	Communication & Audience
5	2023/02/13	Strategy & Analytics
6	2023/02/20	Demo Reels
7	2023/02/27	Frontend
8	2023/03/06	Content Management System
9	2023/03/13	BREAK
10	2023/03/20	Final Project & Coding Interviews
11	2023/03/27	Hosting and Domain
12	2023/04/03	Dynamic Content & Blogs
13	2023/04/10	Promoting
14	2023/04/17	Cover Letters
15	2023/04/24	Traditional CVs
16	2023/05/01	FINALS

https://www.champlain.edu/career-success/career-collaborative/resources-for-students/career-resources/computer-science-and-innovation

August 30, 2023

December 20, 2022

GitHub 

6.1.2 Comments

6.2 Introduction

Estimated time to read: 14 minutes

A game developer portfolio is a collection of materials that showcase a game developer's skills, experience, and accomplishments. It is typically used by game developers to demonstrate their abilities to potential employers, clients, or partners, and may include a variety of materials such as:

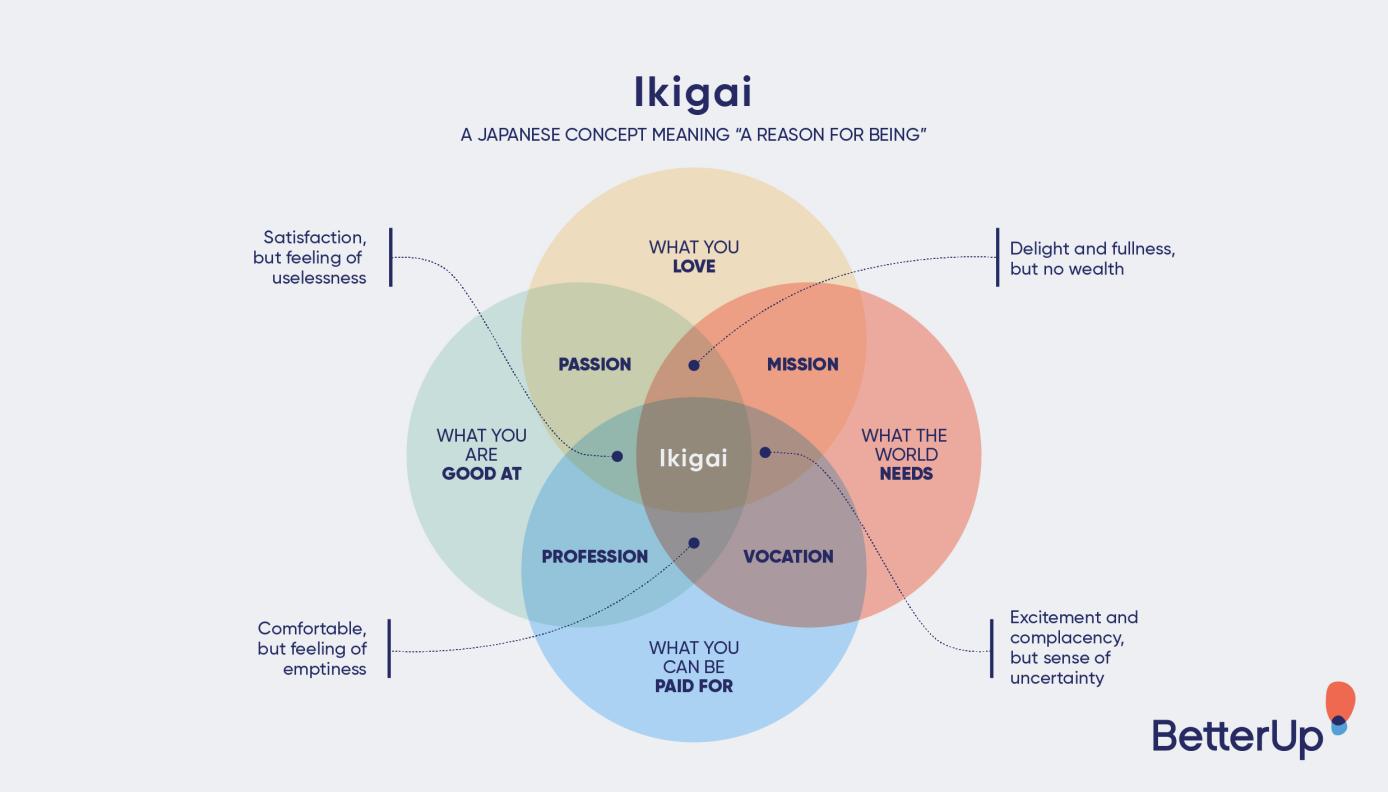
- A resume or CV: This should highlight your education, work experience, and skills relevant to game development.
- Examples of your work: This can include demos, prototypes, or completed games that you have developed or contributed to. It's a good idea to include links to any online versions of your work, as well as screenshots or video trailers.
- A portfolio website: Many game developers choose to create a website specifically for their portfolio, which can include additional information about their skills and experience, as well as links to their work.
- Blogs, articles, or other writing: If you have written about game development or related topics, you may want to include these in your portfolio to show your knowledge and expertise.
- Testimonials or references: Including positive feedback from clients or colleagues can help to demonstrate the quality of your work.

Overall, a game developer portfolio should be designed to demonstrate your abilities and accomplishments in a clear and concise way, and should be tailored to the specific needs and goals of the person or organization you are presenting it to.

Building a portfolio is not only about you, it is about making the life easier of the ones interested on you by giving insights if they should hire you, follow you or anything else. In order to make people understand you, you have to know yourself better.

6.2.1 Who are you, what you excel and what do you enjoy doing?

In your portfolio, you will have to express yourself in a way that others can understand who you are, and it can be challenging for some. In order do help you discover **who you are, what you excel, and what do you really enjoy doing**. I will be briefly vague here to point some emotional support and reasoning to help you answer the question. **If you are clear about that, please skip this entire section**. Here goes a small amount of advices I wish I have heard when I was young.



Ikigai

Note

The above image links to a very good reference to understand the drives that you should be aware while taking decisions on your future career. Visit it.

You are a complex being and hard to define. I know. It is hard to put yourself in a frame or box, but this process is relevant to make the life of the others to evaluate if they want more you or not. If for some reason a person is reading your portfolio, it means that you are ahead of the others, so you must respect their time and goals while they are reading your content.

What you do, do not define what you are, you can even work with something you dont love as long it is part of a bigger plan. Given that, you have to know how to differentiate yourself from your work while respecting your feelings. The sweet spot is when you mix who you are with what you do, and you have nice feelings about it. But this can be hard to achieve and require maturity to mix things. If you dont have a clear understand of those aspects of yourself, you will be subjected to be exploited by bad companies and managers.

It is totally fine try to excel some job you are not passionate. You just have to find means to make your time doing it as enjoyable as possible. In the end of the journey it will slowly become something you can be proud of, and you will become a different person than the one you are now. Understanding this kind of mentality will help you endure more and be more resilient to problems.

Keep track of your progress towards your goal. First of all, **have a clear goal**, so you can build a path to it. Otherwise, any path would sound just like any other apathetic path. Having a clear goal will make your path shine and easy to choose. It will help you in difficult moments where you feel uncomfortable by being just a small piece of a machinery. You will be able to act as part of machine while you need to achieve your goal as a necessary step.

Focus on always keep track on your evolution on your journey to excellence. Don't compare too much yourself to the others, everyone is facing a different journey and everyone took different paths in their career that probably you didn't have the option to chose in the past. But you cannot be uncritical either, you have to analyse your progress and check if your current path is

making you life good, you have to take a decision to change the plan or even the goal with the new information you learned through the current path you are pursuing.

In other point of view, you wont start your career as senior developer, so you have to build your own path. Making mistakes is part of the process, and that is the reason you will be gradually exposed to big things. You should accept yourself, don't push too hard, and do some basic stuff. Just accept the challenges of doing something not fancy, but relevant to build your career.

6.2.2 Define and state your mission and goal

- Are you a generalist or a specialist type?
- What position you are looking for?
- What kind of person you want to become?

6.2.3 Gather information

In order to build a good portfolio, you will need to gather information about yourself and your work. In the process you will discover yourself. It will feels like looking to a mirror for the first time.

If you didnt published yot your projects on itchio, github, or any other platform, now it is a good moment for doing it. Pay attention that if you are going to share your code publicly, you have to avoid sharing content that do not belong to you. In other words, avoid copyright infringements.

6.2.4 Proof of your accomplishments

It is a good practice to always take screenshots, use web archive or any means to prove what you are stating. Some games got lost in time, they die or become unavailable in the long term.

> **Personal advice**
>
> In my case, we developed a very successful game in the past, and because of some problems with investors and judicial dispute, we had to shut down the game. But it was one of the most successful games in that year, it was nominated to Unity Awards and it was the most downloaded racing game. The only things that I can showcase now are print-screens, recorded videos and web-archive pages. So it is something that can make you survive the questions.

6.2.5 Videos, photos, or lightweight web builds

A good way to express your work is to show it in a form of videos, or photos. If your game is small enough to be embedded, or you can strip the most relevant part of it and built for web(webgl, wasm etc), try to publish the relevant part of it online, but do not over-do it, because it will take too much time to craft a good interaction.

6.3 Homework

1. Define your domain name;
   - I usually search domains here and buy on wherever is cheaper, usually here
2. Find a good portfolio to follow;
3. Design the scaffold / wireframe of what you want to show;
4. Gather the data you want to show;
5. Think on catchphrases and call to actions.

August 30, 2023

December 21, 2022

GitHub 

6.3.1 Comments

6.4 Case Study

6.4.1 Case Study

Estimated time to read: 9 minutes

Index

This class will be focused in planning, portfolio evaluation, github processes, ci/cd and in-class activities.

6.4.2 Activity 1

Start setting up your Github pages. We are going to use github pages mostly for two intentions: Webpage hosting for your portfolio and Demo project hosting.

Webpage

For your webpage, you can develop something from ground up using your preferred web framework and we are going to show you how to do it, but the fastest way is to just follow any template. Here goes a bunch of open sourced developer portfolios you can fork and modify for your intent. https://github.com/topics/developer-portfolio?l=html . Try to take a look on them and check if you want to fork any of them. So in this activity you will have to fork and try to run a clone of a portfolio you like just to got into some action and discover how things work.

1. Find a developer portfolio on github
2. Fork it
3. Clone in your machine
4. Make some changes
5. Build it
6. Deploy it to gh-pages either via automated ci/cd or via publishing a build from a empty branch or the main one

Demo reels

For project demo, game, or whatever interaction you want to allow the user to do, I built some boilerplates for you. Later on, you will be able to embed those webgl/html5 builds into your portfolio, so it is a good moment for you to start doing it now. As extras, optionally you can add badges for your repo from here: https://shields.io/

SDL2

In order to showcase your ability to build something from ground up, this repo holds a boilerplate with C++, SDL2, IMGUI, SDL2IMAGE, SDL2TTF, SDL2MIXER, CI/CD automation for automatic deployment: https://github.com/InfiniBrains/SDL2-CPM-CMake-Example

1. fork it
2. go to the repo settings, actions, general, in the bottom, enable workflow permission, read and write, save
3. run github action at least once
4. enable actions and automatic page deployment from a branch gh-pages

AI + SDL2

If you enjoy AI programming and want to test yourself, you can try forking this repo and implement what is inside the examples folder https://github.com/InfiniBrains/mobagen

1. fork it
2. run github action at least once
3. enable actions and automatic page deployment from a branch gh-pages

UNITY

If you want to showcase your ability with Untiy, you can follow this boilerplate to have an automatic build set up. https://github.com/InfiniBrains/UnityBoilerplate

1. Fork it
2. run github action for getting an unit licence at least once
3. grab the generated file, and upload it to https://license.unity3d.com/manual
4. get the signed licence and copy the text content to your clipboard
5. go to your repo settings, security, secrets and variables, actions and setup a new repository secret with the name 'UNITY_LICENSE' and the content from your clipboard
6. go to the repo settings, actions, general, in the bottom, enable workflow permission, read and write, save
7. run the main action
8. enable actions and automatic page deployment from a branch gh-pages
9. edit webgl template with your logo or image

6.4.3 Activity 2

This class is totally up to you. Here goes what you should do in class and finish at home. The idea is for you to feel a whole process on how to create merge and pull requests to a public repo.

1. Search a good portfolio published online
2. use Twitter, LinkedIn, Google to search for good game developer portfolios;
3. another good query on google would be "awesome developer portfolio", or "curated list of developer portfolios" try it!. Example: https://github.com/emmabostian/developer-portfolios
4. You can use this time to search a good and open sourced portfolio to fork and start your own based on other. https://github.com/topics/developer-portfolio
5. Fork this repo
6. Create a markdown file in this folder with a meaningful name about the benchmarked repository.
7. Follow this example
8. The file name should be the website domain name followed by .md
9. If another student is aiming to evaluate the same portfolio, just edit the file adding your evaluation to the text.
10. Your file should contain:
11. A summary
12. The portfolio evaluated
13. The date the evaluation happened
14. Print-screens uploaded to image hosting services such as imgur or others
15. What things you judge as good and you are aiming to follow and target
16. What things you judge that needs attention and should be improved
17. Why you would hire the owner of the portfolio
18. General considerations
19. Edit this file on github to link your work here if you want to showcase it here.
20. Be Kind and constructive
21. Send a push request

6.4.4 Considerations

- The portfolios evaluated here are just opinions

6.4.5 Evaluated Portfolios

- Example

August 30, 2023

January 5, 2023

GitHub 

Comments

6.4.6 Index

Estimated time to read: 2 minutes

6.4.7 Assessment 1

Summary

- The date the evaluation happened
- The portfolio evaluated
- Briefing

Strength

What things you judge as good and you are aiming to follow and target. Add images as reference using print-screens uploaded to image hosting services such as imgur or others;

Improvements

- What things you judge that needs attention or should be improved?
- What questions you would ask this person?

Best fit

- Why you would hire the owner of the portfolio?
- For what kind of task?
- What position?
- How do you see this person interacting with others?

General considerations

- Just add some final consideration for the portfolio owner;
- If possible, send a message to one of its communication channels informing your assessment;

6.4.8 Assessment 2

The other student willing to do multiple assessment for the same portfolio, just create an entry in the index following the same structure and same the assessment differently in this case, we put number 2. And use the same structure on the 1.

August 30, 2023

January 5, 2023

GitHub 

Comments

6.5 Game Developer Portfolio Structure

Estimated time to read: 10 minutes

Create a single page app containing most of these features listed here.

6.5.1 Head

Chose carefully what to you use as a head of your page. It is the first thing a person reads. It can be an impactful message, a headline, personal statement, background video or very limited interactive section.

> **Note**
>
> Avoid bravado. You can be bold without being naive. Let the bravado statements for when you become a senior. If you write bravados right in the begining of your portifolio and you are still a junior, you are just communicating that you will be hard to work with. A senior developer reading your portfolio is more interested in developers eager to learn, humble, and looking for guidance so they will have a easier life hiring you.

6.5.2 About

This is a summary obout yourself, be brief and achievement oriented. What and who you are. Contact info via social medias. State your working status and target. If you are a narrative centred person, you can create something fancy here, but dont over-do, less is more!

6.5.3 Showcase

- Projects
- Ability and versatility
- Community Contributions

You can showcase your personal work, a job you make for a client(if authorized).

> **Projects**
>
> It is a good practice to showcase only the best works you made. You might find interesting to add more than 5, but there are chances of your reader clicking exactly on the worst one and have a bad first impression of you. In your showcase section, avoid showcasing bad work. Invite some of your friends to help you select the best ones to showcase.

> **Ability**
>
> Avoid using percentage graphs to showcase your proficiency on specific tech stack or tool. The main reason is: how do you grade of your ability as 80%, 100% or 30%? Worse than that, how can the reader be sure of that? If you want to do that, it is better to apply for certificates, there are plenty on linkedin or specialized sites.

6.5.4 Achievements

- List key achievements and skills. Dont use any kind of grading
- Education
- Testimonials or anything to prove your skills and capacity

6.5.5 Project Details

- You should create a way to explain more about what is showcased. Ex.: redirect the user to the project description page, or open a modal

6.5.6 Blog

- Featured posts/content and call to action to read your ongoing content production
- Explain your process in designing a game or piece of software
- Explain some interesting details you learn or describe your knowledge explorations.

6.5.7 Contact

Explicitly state what people should expect if they contact you and what they can expect from your return. Ex.: If you aim to be a freelance, state your offer and ask for them to briefly state the job activity, time frame and the rate they are willing to pay. If you are looking for a full-time position, the most common way is to just share your email, so they can contact you.

Another option is to list all of your social medias, but dont overuse this. Nowadays we have a bunch of them, so if you list all of them, there is chances, you are not active there and the link will guide the reader to a empty and haunted house and they will not engage.

6.6 Homework

1. [Optional]Create a wireframe draft of what you are going to do on figma, miro or any other tool you find relevant. If you already have a Portfolio page already set, try to think on how do you make it be responsive on mobile devices.
2. Scaffold your project in github. Describe in the README.md what you are going to do in your demo. You might try to convince a colleague to work with you. You can do more than one demo if you want. Some examples:
3. Follow some of mobagen AI examples from here https://github.com/InfiniBrains/mobagen check some of them running here https://infinibrains.github.io/mobagen/
4. Raytrace demo for cloud and Human body. ref https://www.youtube.com/watch?v=Qj_tK_mdRcA
5. Networking game using the phone as a controller and your portfolio demo as the viewer. https://blockrage.pgs-soft.com/
6. Catch the cat demo for AI. https://llerrah.com/cattrap.htm
7. Create unity editor plugin to dynamically generate meshes via splines based on sample locations. https://www.youtube.com/watch?v=f5Q7Z2KxILE is a game with 50million downloads that used this technique to generate scenarios. Here you can watch a better explanation. https://www.youtube.com/watch?v=saAQNRSYU9k
8. Embed a shader toy behind your portfolio page. https://shadertoyunofficial.wordpress.com/2018/02/17/embedding-shadertoys-in-website/
9. A nice collection of fun ideas to code for your portfolio https://mrdoob.com/
10. Board game simulation. Ex: chess https://www.chessprogramming.org/ or https://github.com/JuUnland/Chess/ or this https://www.youtube.com/watch?v=WKs685H6uOQ
11. [Optional] Simulate a real test by creating a console based space invader in 48h.

- Efficiently draw (print on console)
- Efficient collision check

6.7 Most requested topics to be covered in this class

- 13 #AI #flocking #BoardGames #MinMax #chess #procedural
- 5 #gameplay
- 4 #Rendering #vfx #grass #leaf #water #graphics #shader

- 4 #UI #UX
- 4 #networking
- 4 #interview #questions
- 3 #XR #VR
- 3 #tools #unityeditor
- 3 #backend #devops
- 2 #physics
- 2 #PortifolioPage #website #responsive
- 2 #CV
- 2 #github #githubpages
- 2 #writing #narrative
- 2 #itch
- 1 #soundtrack #audio
- 1 #LinkedIn
- 1 #security

August 30, 2023

December 21, 2022

GitHub 

6.7.1 Comments

6.8 Communication

Estimated time to read: 10 minutes

Having a well-written and organized portfolio is important for any game developer, as it can help them stand out from the competition and demonstrate their skills and experience to potential employers. A good portfolio should clearly communicate the developer's strengths and accomplishments, and should be tailored to the specific needs and expectations of the audience.

Effective communication is crucial in building a strong game developer portfolio, as it allows the developer to clearly convey their skills and experiences to potential employers. A portfolio that is well-written and easy to understand will be more effective at convincing an employer to hire the developer, while a poorly written or poorly organized portfolio may have the opposite effect.

6.8.1 Audience

In general your portfolio will be read by:

- Human Resources
- Software Developers

Human Resources

If you are applying for a big tech company, chances are your submission won't be read by a tech person the first human triage. So in order to pass this first filter, you have to be generic and precise. They are often very busy evaluating multiple applications, and probably they will spend 30-60 seconds before making the decision about moving forward in the process or not. Your portfolio will need to catch their attention and communicate clearly your fit, passion and ability in a short time frame.

Software Developer Managers

In contrast with HR, developer managers probably will not be shocked with any fancy stuff(such as full page pre-loaders) you add to your portfolio, so be concise and straight to the point, because most of them already know all the contents. From all of your portfolio readers, they are one of the most critique of your job.

In another hand, usually developers do not look for programming language fit, frameworks or tools you use. They are more interested if you will be able to learn and execute the job in a meaningful time. So try to express yourself in a way that showcase your ability to solve problems, no matter what problem is, they are mostly curious on how to solve complex problem by framing the problem in another way or how to be innovative.

6.8.2 What they look for

The following metrics can be evaluated by reading your portfolio, interviews or tests. The most common evaluation metrics they made are: - Position Fit - They are going to search if your portfolio showcase experience in the same area of what they are looking for the specific position they received. Usually they will look for specific keywords for the requirements list; - Company fit - They take count on your expressed ideas, stated goals, tone, alignment and supporting projects to evaluate your future evolution inside the company; ex. https://wa.aws.amazon.com/wat.pillars.wa-pillars.en.html - Passion - Passionate developers tend to express projects they are proud of. The description of the projects are mostly achievement-based. Ex.: more than X million downloads. This example showcases that you were part of something huge, and it is easily understandable. - There is a high correlation on high performant people that they usually shine in side-projects or even hobbies. So they look for it. Ex.: Google encourages employees to devote 20% of their time to hobbies or skill-building. - Competence - They need to evaluate if you are really able to solve the problem properly, in a meaningful time, and in a team. You have to describe which tools, tech stack and how you glue everything in order to solve the problem. Be assured you are correctly expressing yourself here, because it is one of the central part that is not taken superficially. - Innovation and Curiosity - Innovative developers solve problems out of the box. It doesn't matter how complex the problem is, but if you solve in a innovative way, reframe it or do any magic to solve it, chances are to have good points here; - Good companies incentives research and test new stuff. So they usually like to see your deliverables with new bleeding-edge technology tools. - Proactiveness - Usually the more proactive developers tend to have more

leadership positions. So if you want to give the readers a glimpse of your ability in this area, a good place to showcase that is in project description section. Express problems that arise and how do you manage that before it become a real problem.
- Learner - It is good to be always tuned with the current evolution of the technology, so try to keep the education section always updated with some courses or publish blog posts about some new tech. - Thoughtfulness - Risk management - Thinking big

6.9 Homework

- Write the content for each one of your projects and all other sections of your portfolio.
- 1st round of Portfolio Feedbacks.
- Research in a game company you like the resources on their hiring process. Take notes on how they hire and which positions you want to apply. Here goes some examples on how to get ready for interviews:
- Epic: https://www.epicgames.com/site/en-US/earlycareers/career-paths
- AWS: https://aws.amazon.com/careers/how-we-hire/
- General interview questions https://www.mockquestions.com/
- Top interview questions https://leetcode.com/problem-list/top-interview-questions/. Some examples:
- https://leetcode.com/problems/two-sum/ - the most asked question
- https://leetcode.com/problems/kth-largest-element-in-an-array/ - heap
- https://leetcode.com/problems/median-of-two-sorted-arrays/
- https://leetcode.com/problems/minimum-space-wasted-from-packaging/ - bin packing, try to solve for 2d image pack for textures
- Watch this video https://www.youtube.com/watch?v=Kte-t1pQQ3I and start contributing into an open source project or community

August 30, 2023

December 27, 2022

GitHub 

6.9.1 Comments

6.10 Frontend for your portfolio

Estimated time to read: 3 minutes

Here goes a curated templates for a quick start: - https://github.com/techfolios/template - the easiest one - https://github.com/rammcodes/Dopefolio - straight to the point developer portfolio - https://github.com/ashutosh1919/masterPortfolio - animated with a strong opening - https://smaranjitghose.github.io/awesome-portfolio-websites a good compilation on how to build and deploy your portfolio with a good pre-made template

But for this class, we are going to follow this template, sofork this boilerplate if you want a more robust webapp experience.

6.11 Frontend frameworks

There are many frontend frameworks floating around, but in order to speed up your learning curve on how to deploy a fully customized webpage, I am going to use this combination of technologies:

- React for building website;
- Vite for tooling;
- Tailwindcss for styling;

Some examples with this stack:

- https://reactjsexample.com/a-portfolio-page-using-react-js-and-tailwind-css/
- https://github.com/InfiniBrains/reactjs-vite-tailwindcss-boilerplate

Watch this video to get a fast entry to this stack Here goes an introductory video about this combination.

August 30, 2023

December 27, 2022

GitHub 

6.11.1 Comments

6.12 Final project

Estimated time to read: 4 minutes

Your portfolio should be a hosted webpage and a open repository on github.

You should follow a portfolio structure, to build a website and host it publicly. It should have a nice style, a good communication is the key to execute and analyse your strategy in order to capture insights. You can optionally increment your portfolio via dynamic content such as blogs or whatever you find relevant. Another extra step would be to create a generic cover letter to express your intentions and goals more personally. Note that some game companies still require CVs To boost your visualization, you can promote.

Minimum steps: 1. Have a domain or at least a meaningful github username/organization; 2. Create a github repository; 3. Push your frontend to the repo; 4. Enable github pages; 5. Create a CI/CD to build and deploy to gh pages; 6. Point your domain to gh-pages if you have one;

It is expected to have something to showcase, so it is expected to have at least 3 projects to showcase. It is preferable to showcase something that could be testable(webgl builds) or watchable in a lightweight manner.

If you are willing to showcase your ability in Unity, I recommend you to try GameCI and github pages. If you want to showcase your game engine abilities with C++, I recommend you using CMake, SDL2 and emscripten to build and deploy for github pages.

If you want to start something from scratch you can use this repo to start have a SDL2 project with all libraries already set. It builds and publish a Github page via Github actions automatically, you can check it running here. It features CMake tooling, IMGUI for debug interfaces, SDL2, SDL2_ttf, SDL_image, SDL_mixer,

6.13 2023

Here goes a list of portfolios

August 30, 2023

January 12, 2023

GitHub 

6.13.1 Comments

6.14 Hosting

Estimated time to read: 9 minutes

There are many hosting options and solutions to match each need. Lets cover some options here.

6.15 Options low code

- Google sites - My preference

Other notable options: - Godaddy - Wordpress - Wix - Squarespace

The problem with those are they require payments to be fully functional, so if you want to go deep and have mor freedom, we are going to cover other options.

6.16 Static HTML with Static Data

If what you want to serve is static hosting, your content is only frontend and do not require backend, you can use github pages, google firebase, S3 bucket hosting or many others. This is the easiest approach. - In this scenario you will be able to store only pre-generated html and static files; - This is useful even if you use blogs that changes rarely, you would have to redeploy your page for every change.

6.17 Static HTML with Dynamic Data

If your html is static and need backend services that are rarely called, you can go with cloud functions, my suggestions here are google cloud run and aws amplify or even firebase functions. If you use nextjs website, check vercel or netlify hosting services. - The deploys are easy; - It can be very expensive if you hit high traffic, but it will remain free if you dont hit the free tiers; - You will have to pay attention to your database management;

6.18 Dynamic HTML with Dynamic Data

If your website generate content dynamically such as Wordpress blogs or any custom made combination with next or anything. - There is many "cheap hosting" solutions that are mostly bad performant(it can reach more than 10s to answer a request). You have to avoid them to make your user enjoy the visit; - Management can go as hard as possible, but the results can be awesome; - It can be really expensive;

6.19 CDN and DNS Management

I highly recommend you to use Cloudflare as you DNS nameserver, so you can cache your website results for faster loading. But you can use your own nameserver provider by your domain name registrar.

DNS stands for Domain Name System, which is a system that translates domain names into IP addresses. When you type a domain name into your web browser, such as "www.example.com," your computer sends a request to a DNS server to resolve the domain name into an IP address, such as "192.0.2.1." The IP address is then used to establish a connection with the web server that hosts the website you are trying to access.

DNS plays a crucial role in hosting because it enables users to access websites using domain names instead of IP addresses. This makes it easier for users to remember and find websites. DNS also allows websites to change servers or IP addresses without affecting the user experience, as long as the DNS records are updated properly.

In hosting, DNS is important because it determines which server is responsible for hosting a particular website. DNS records can be configured to point to different servers depending on factors such as geographic location, server load, and failover. Hosting providers typically offer DNS management tools to help users configure and manage their DNS records.

6.20 Homework

The goal is to have as website up and running for your portfolio.

Here goes my preferable way for hosting anything. With that you can host microservices, game services, serve API, static and dynamic websites and much more. It can be tricky but lets setup it now.

- Oracle cloud - Free forever - Virtual Machine with 4vCPU, 24GB ram, 200GB storage. https://www.youtube.com/watch?v=NKc3k7xceT8 watch up to 5:38 time
- Coolify - Your private Software as a Service (SAAS) manager - https://youtu.be/Jg6SWqyvYys?t=125 starts from minute 2:00
- CI/CD - to your remote machine https://www.youtube.com/watch?v=Uj7F3hdgmEo
- Cloudflare DNS - set your domain to point to your DNS - https://www.youtube.com/watch?v=XQKkb84EjNQ
- Install Wordpress via coolify interface(new resource, new service) and use your own DNS. Or host your page statically https://www.youtube.com/watch?v=CfdPyASUSkI&

Talk with me if you dont have a domain and want to use my infrastructure temporarily.

I am assuming you wont have a huge traffic, but you have a complex combination of services. In the complex cases and if you want to make your life easier and cheaper,my suggestion for hosting would be oracle cloud with arm cpu. They offer for free a virtual machine with 200gb storage, 4vcpus, 24gb ram for free at this date of 2022/12 tutorial. In this scenario, I recommend using https://coolify.io/ as your deployment management system, just pay attention that this machine is running in an arm cpu. With this combination, you can manage everything easily in one place for free. This is not ideal, because you wont have backups, but it is good enough for most scenarios.

If you have plenty of money or your website have high traffic, I recommend you to use Kubernetes to orchestrate every microservice.

August 30, 2023

March 1, 2023

GitHub 

6.20.1 Comments

6.21 Dynamic Content

Estimated time to read: 1 minute

August 30, 2023

December 27, 2022

GitHub 

6.21.1 Comments

6.22 How to promote yourself and your work

Estimated time to read: 13 minutes

For most of us, game developers, the most important thing is to make games. But, in order to make games, we need to promote ourselves and our work. In this section, we will learn how to do that.

6.22.1 Defining the target to be promoted

Before we start promoting, we need to define what we want to promote. The main difference between promoting ourselves or our work is the tone, the message and the medium being promoted. So we can build a successful strategy.

In ether path you chose, consider the following questions:

- What is the target audience?
- What is the target platform?
- What is the target medium?
- What is the target message and content?
- What is the target call to action?
- What is the target result?
- How to measure the success?
- How to improve the promotion?

6.22.2 Defining the Audience

Before creating and running a promotion campaigns, we need to define the audience. The audience is the group of people we want to reach with our promotion and it can defined by the following:

- Recruiters, HR, and hiring managers;
- Other game developers, especially those who are in the same field as you;
- Game players;
- Journalists, writers and critics;
- Investors;
- Communities;

6.22.3 About Platforms

To reach specific audiences, we need to be in the same platform they are. For example: - Game players: Steam, Twitch, YouTube, itchio, GameJolt, Discord; - Journalists: Twitter, LinkedIn; - Investors: AngelList, Ycombinator, LinkedIn, Crunchbase; - Communities: Reddit, Discord, Facebook, Twitter; - Recruiters: mostly Linkedin.

Social media is a great way to promote yourself as a game developer. You can use it to share your work, your thoughts, your ideas, and your opinions. You can also use it to connect with other developers and learn from them.

Here goes my opinion about the most important platforms:

- Twitter: it is the best and easy way to communicate with anyone in the world. The distance between to reach anyone is zero. And it has an awesome tagging structure. It is a great way to share your thoughts and ideas and ask for feedbacks. It is a great way to connect with other developers and learn from them. You can also use it to share your work and promote your games.
- LinkedIn: It is a great way to connect with recruiters and hiring managers. Usually you will see other developers publishing their thoughts and ideas, so try to post relevant comments on their publications to get noticed and improve your visibility.
- Facebook: It is mostly a general purpose social media. You can use it to connect with your friends and family, and collect feedbacks for your content. It is a great way to promote your finished games.
- Instagram and Tiktok: Are more focused in fast, small and visual content. You can use it to promote your games and your work, but it is not the best way to share your thoughts and ideas.
- Reddit: This one is the best for collecting feedbacks from other developers about your content, but the reach is limited.
- Discord: The best tool be in touch with communities, you can build your own community for your game and be in direct contact with yours consumers. Another good use is to be in direct contact with other developers and learn from them.
- YouTube and Twitch: The best way to share your work and promote your games.
- Medium and Blogs in general: The best way to share your thoughts and ideas and ask for feedbacks. You can use it in conjunction with other platforms to catch the general attention and bring them to your content.

6.22.4 Mediums

The mediums are: Social media posts, Blog posts, Email, Podcasts, Videos, Events, Conferences, Meetups, Workshops, Webinars, Webcasts, and more.

For each type of medium, we need to plan the content, the frequency, and the duration. We have very nice tools to help us with that, like Buffer, Hootsuite and many others.

6.22.5 Message and tone

The message is the main idea we want to communicate. The tone is the way we want to communicate it. You have to match the tone with the message in the given platform to reach the right audience. So plan ahead how you want to communicate your message and what tone you want to use.

When planning the message, it is good to plan the emotions we want to trigger in the audience. For example, if we want to promote our game, we can use the following emotions: Excitement, Joy, Curiosity, and Fun. If we want to promote yourself by doing something interesting, you can use the following emotions: Curiosity, Fun, Surprise and Pride.

6.22.6 Call to action

The call to action is the action we want your audience to take. It can be: Download the game, Read my Resume, be part of by community, Take a look on my Repository, Buy the game, Play the game, Follow me, Subscribe, Share, Like, Comment, and more.

6.22.7 Results

Whatever is your goal, you need to define the results you want to achieve so you should track and measure your progress. You can use tools like Google Analytics mostly for web content, Google Firebase for apps and games and many other.

Here some ideas of results you can track: Number of downloads, page views, number of people reaching you, number of followers, number of subscribers, number of likes, number of comments, number of shares, number of retweets, number of reposts and more.

6.22.8 Improving the promotion

If you really want to go deep in this rabbit hole, I highly recommend you to create performance measurements such as KPI dashboard to track your progress and improve your promotion. You can use tools like Google Data Studio or Tableau. With the

KPI dashboard, you can track your progress and improve your promotion. You can also use it to track your competitors and learn from them.

Another good strategy is to A/B test your promotion. You can use tools like Google Optimize to create different versions of your promotion and test which one is the best. You can also use it to test different messages, tones, and call to actions. I cannot stress enough how important it is to test your promotion, the most successful companies in the world do it. Zynga even quoted once "We are not in the business of making games, we are in the business of testing games" and "We are a data warehouse maskerated as a game company". So being data-driven and customer-centric is the key to success.

6.23 Homework

- Create a promotion strategy for yourself and your work.
- What would be your first content and medium to promote?
- What is the message and the tone?
- Define your call to action.
- How do you measure your results?
- How would you plan to improve your promotion?

6.24 Conclusion

I hope you enjoyed this content. If you have any questions, please create an issue in this repository. If you want to contribute, please create a pull request. If you want to support me, please share this content with your friends and colleagues. If you want to support me financially, please consider buying me a coffee or a very fancy wine.

August 30, 2023

December 27, 2022

GitHub 

6.24.1 Comments

6.25 How to write an Awesome Cover Letter

Estimated time to read: 13 minutes

6.25.1 What is a cover letter?

A cover letter is a document that is sent together with your resume. It is a way to introduce yourself to the company, explain why you're applying for the job, and why you're a good fit for the position. You should also explain why you're interested in the company, and why you want to work for them.

Nowadays writing a Cover Letter seems to be a lost art. Most of the time, people just send their resume and that's it. But, if you want to stand out from the crowd, you should write a cover letter.

In a cover letter you can be more personal to sell yourself more effectively. The core of it is to link your skills and history to what they do and need. Now lets see how to write a cover letter.

6.25.2 Strategies to write a cover letter

There are many strategies to write a cover letter. But the main idea is to be personal and try to sell yourself more effectively. Here are some strategies to write a cover letter:

- **Be clear, concise and specific**. You should try to be clear and try to sell yourself more effectively. Don't waste their time with long and boring paragraphs. You can do that by linking your skills and history to what they do and need. You can also try to show your personality and your passion for the job;
- **Be personal, enthusiastic and professional**: You should try to be personal setting the best tone that matches your style and the company, just don't exaggerate. You can also try to show your personality and your passion for the job. But, you should also be professional and try to be polite and respectful. If you're unsure about the company culture, you can do that by using a formal language and a professional tone;

6.25.3 Knowing your audience

Usually, game companies are interested in people who are passionate about games. But there are some core differences between what profiles AAA game studios and Indie Studios seek for. AAA usually follow the path of the specialist, while Indie Studios usually, the generalist. So try to match this style of writing in your cover letter.

Another relevant aspect is the company culture. You should try to match the tone of your cover letter to the company culture. If you're unsure about the company culture, you can do that by using a formal language and a professional tone. Or try to connect with some employees of the company and ask them about the company culture.

Research about the company. Try to find out what they do, what they are looking for, and what they are interested in. You can do that by reading their website, their blog, and their social media. They tend to prefer people that have culture, passion and goals aligned with theirs. So try to show that you are passionate about their products and their goals.

Play their games, and use their products. An awesome icebreaker can be yourself telling about some funny bug or how you enjoyed the game connecting it to your life. It would be awesome if you can show that you are a fan of their products to the point to even create mods or fan art.

6.25.4 Write interesting content

You should try to write memorable sentences to maintain your reader engaged. One strategy is to start the paragraphs with a short and powerful sentence that summarizes the the topic you are about to write. Arguably, you can also try to use a powerful quote to start your cover letter.

Your first sentence plays a huge role in your cover letter, it should be meaningful to you and to the reader. Chances are, they wont be reading the whole cover letter, so you should try to make the first sentence as interesting as possible. Try to be catchy and try to make them want to read more, but take care not to exaggerate.

Sometimes your content is really relevant to you but it might not be that relevant to the company or the job. Sometimes we get too excited and we want to tell everything about ourselves and how passionate we are, by try telling all the things you ever did. But you should try to be clear and concise. Just add some breadcrumbs for the reader ask you in the interview about the things you didn't mention in the cover letter.

6.25.5 Strengths

You should try to highlight your strengths. You can do that by using a list of your skills and achievements. They will try to extrapolate the value you brought to the previous companies you worked for to themselves. So try show that you are a good fit for the job by giving success stories about your acchievents. Some examples:

- **AAA centred**: I published a game on Steam with 100k downloads while a student. I acted as the main developer and tech lead, responsible for creating tools for level designers and AI system for the game. Besides that, I played a fundamental role to cut the scope of the game to make it possible to be released on time and consequently the sanity of the team;
- **Indie**: I am fearless. I am not afraid to fail or take risks. This behavior pressures me to have a good plan and to be prepared for the worst. Once we tried a very ambitious feature that we thought would be awesome, we tracked the adoption of it, just to discover that nobody used it. We learned from it and we tried again with a smaller scope and it worked. We released the game on time and we were happy with the result;

Pay attention that some companies might not like to see that you are a risk taker. So try to be careful with that, and ask some employees of the company and ask them about the company culture.

6.25.6 Closure

You should try to close your cover letter with summary, thank them for their consideration and time, and add a call to action. You can also try to add a call to action to connect with you on social media or to visit your website, or just say that you are in hopes to talk with them in person soon.

6.25.7 Create a Template

You should try to create a template for your cover letter. A way of doing it is to add replaceable tags for the company name, the job title, and the date. Try to mark those tags in some colorful way, so you can easily find them and replace them. You can also try to add some comments to help you remember what to write in each tag.

Another strategy to templating your cover letter is to create one template for every type of company. For example, you can create a template for AAA game studios, another for Indie game studios, and another for game companies. You can also create a template for each type of job. For example, you can create a template for a game designer, another for a gameplay developer, and another for a UI/frontend developer.

But if you pursue this path, you have to pay attention to the examples and products/games that you use in your cover letter. You will have to change them to match the company you are applying for.

6.26 Homework

Write a Cover Letter for a game company.

August 30, 2023

December 27, 2022

GitHub 

6.26.1 Comments

6.27 CV

Estimated time to read: 1 minute

August 30, 2023

December 27, 2022

GitHub 

6.27.1 Comments