네오플 도어락 부수기

1. 자기소개

안녕하십니까 역배에 올인하는 남자 이 용 석 이라고 합니다.

이런 수식어를 붙인 이유는 친구들과의 내기 때문입니다.

3년전 네오플이라는 꿈이 생겼을때

친구들은 너나 할 것 없이 못간다에 손목을 베팅했지만 고등학생때까지 운동을하며 얻은

승부욕과 끈기를 바탕으로 3년동안 꾸준히 노력하는 모습을 보고

이제는 저를 응원해주고 제게 생소했던 취업준비를 도와주는 좋은 친구들이 되었습니다.

하지만 오늘 친구들의 손목을 얻기 위해 최선을 다해보도록 하겠습니다.

감사합니다.

1. 게임프로그래머가 되고 싶은 이유

재수를 할 때 좋은 성적이라는 목표는 있었지만 구체적인 목표가 없어서

고민하던 시기에 고등학생때 같이 운동을 했던 형의 추천으로 게임공학과를 알게되었습니다.

형과 대화를 하며 "내가 좋아하는 게임을 내가 개발해본다?"라는 생각이 너무 매력적으로 다가왔고 크레이지 아케이드를 모작한 작품을 보며 "와 나도 저런거 만들어보고싶다"라고 생각해서

게임공학과에 진학했고 처음 팔라독 모작 개발에 참여해보니 상상대로 신기하기도 하고 성취감을 많이 느껴서 게임 프로그래머의 길을 선택했습니다.

1. 서버프로그래머가 되고 싶은 이유

싱글플레이 위주의 게임을 개발하다가 3학년때 처음 서버프로그래밍을 접하게 되었는데

친구들과 같이 게임을 하는 것을 좋아하는 제가 다른사람이 접속해서 같이 게임을 할 수 있는 기능을 구현하는 것이 너무 신기했고 재밌었습니다.

그래서 서버 개발자의 길을 택하게 되었습니다.

1. 프로그래밍할 때 나의 단점은 무엇인가?

작업을 하다가 흐름이 끊기고 다음에 다시 작업을 시작할 때 이전의 생각들을 잊어버리는 것이 가장 큰 단점이였습니다.

* 어떻게 해결하셨나요?

메모하는 습관을 들이기 시작했습니다.

작업 종료 30분 전에 이 때까지 했던 생각들에 대해 포스트잇에 정리해서 붙이기 시작했고 팀원들도 이런 습관이 좋아보였는지 화이트보드를 사서 벽에 붙여놓고 작업내용을 메모하면서 해결했습니다.

커밋 회수도 많아지고 개발 속도가 훨씬 빨라졌기 때문에 좋은 습관인 것 같습니다.

1. 사회생활은 어떤 편인가요?

사회생활은 항장 잘하는 편이라고 생각합니다.

* 왜 그렇게 생각하시나요?

사람들과 어울리는 것을 좋아하는 성격이라 학교생활 할 때는 학생회도 오래하며 원만한 관계를 유지했었습니다.

그리고 저는 어릴 때부터 알바를 많이 했는데 일하는 곳의 사장님들이 항상 저를 이쁘게 봐주셨기 때문에 사회생활은 잘하는 편이라고 생각합니다.

* 사례가 있나요?

최근에 학교근처 카페에서 일을 하며 만난 매니저님이 가장 생각이 많이나는데

남들이 꺼려하는 일도 웃으며 열심히 하는 모습을 좋게 보고있다고 하시며 카페 브랜드가 커지면 같이 일하자는 제안을 받았었던 기억이 가장 임펙트가 크게 다가왔던 것 같습니다.

1. 협업 시 가장 힘들었던 점

작업량의 차이로 발생하는 문제들을 해결하는 과정이 힘들었습니다.

저희는 사무실을 대여해서 같이 작업했기 때문에 서로의 작업량에 대해 잘 알고있었는데

작업량이 적던 친구가 거짓말로 작업을 회피하려는 모습을 보인적이 있었습니다.

거짓말에 너무 화가났지만 무작정 들이박는 것은 팀 분위기에 좋지않다고 생각해서

“쉬고 싶은 마음은 알겠지만 할 건 하고 쉬었으면 좋겠다, 내 작업이 끝나면 너꺼 도와줄

테니 속도좀 내보자” 라며 대화로 풀어가려 노력했고 이런 경험을 하면서 책임감을 많이 배웠

습니다.

졸업작품 때는 힘든 점이였지만 앞으로 더 큰 프로젝트를 진행할 때는 장점이 될것이라 생각합니다.

기술 면접 C/C++

* L-value 와 R-value

L-value는 표현식 이후에도 사라지지 않고 지속되는객체

R-value는 표현식 이후에 사라지는 임시 객체 입니다.

* Auto 자료형

초기화 값에 따라 데이터 타입을 정해주는 키워드입니다.

생성 시 변수를 초기활 할 때만 작동하며, 초기화 값을 사용하지 않고 생성된 변수는 기능을 사용하지 못합니다.

그리고 매개변수와 함께 사용할 수 없습니다.

* Volatile 자료형

해당 변수의 컴파일러 최적화를 제한하는 용도로 사용됩니다.

* 어떤 경우 사용되나요?

일반적으로 멀티스레드 환경에서 많이 사용되는 걸로 알고있지만 메모리 접근 순서를 보장하지 않아서 cpu가 실행시간에 메모리접근을 재배치할 수 있기 때문에 사용하면 안된다고 알고있습니다.

* Atomic 자료형

메모리 접근 순서를 보장하여 DataRace 문제를 해결할 수 있는 타입이다. 느리다는 단점

* 스마트 포인터

프로그래머가 메모리를 해제 하지 않았을 경우를 대비하여 사용이 끝난 메모리를 안전하게 자동으로 해제해 주는 역할을 하며 Unique\_ptr, Shared\_ptr, weak\_ptr이 있습니다.

Unique\_ptr은 하나의 스마트포인터만이 특정객체를 소유할 수 있도록 합니다.

그렇기 때문에 복사나 대입과 같은 공유를 허용하지 않는 스마트 포인터 입니다.

Shared\_ptr은 참조 카운팅을 기반으로 동작하는 스마트 포인터이며 하나의 객체를 다른 Shared\_ptr과 공유할 수 있습니다.

하지만 두객체가 서로를 참조하여 참조 카운트가 0이 되지못해 스스로 메모리 해제가 불가능한 상황을 말하는 순환참조가 발생할 수 있다는 단점이 있습니다.

이를 해결한 것이 Weak\_ptr인데 sharted\_ptr의 객체만 참조할 뿐, shared\_ptr의 참조 카운트를 증가시키지 않는 방식입니다.

* 물어보면 대답

Shared\_ptr에 강한 참조 카운트와 약한 참조 카운트가 있는데 객체 생명 주기에 관여하는 강한 참조 카운트를 올리지 않을 뿐 약한 참조 카운트는 증가시킵니다.

* Constexpr 과 const 차이점

가장 큰 차이점은 const는 런타임 상수이고, constexpr은 컴파일 시간 상수식입니다.

그렇기 때문에 Constexpr은 잘못 사용시 컴파일 타임에 오류가 발생한다는 장점이 있습니다.

* Define 과 const 차이점

Define은 타입지정을 하지 않아도 되고 디버깅시에 확인이 불가능 하며

Const는 디버깅시에 확인이 가능하다는 차이가 있습니다.

* 함수호출 규약인 Cdecl 과 stdcall, fastcall

Cdecl은 함수 호출용으로 스택에 쌓은 파라미터를 호출한 함수에서 제거하고,

Stdcall은 호출 당한 함수 내부에서 스택 포인터 연산을 합니다.

즉, 스택 포인터의 연산 위치의 차이가 존재합니다.

* 호출된 쪽에서 스택을 정리하는 all은 가변인자를 허용하지 않고

호출자가 정리하는 cdcel은 가변인자를 허용합니다.

* 클래스와 구조체 차이점

기본접근지정자의 차이가 존재합니다.

구조체는 기본접근 지정자가 public이고 class에서는 private이라는 차이점이 있습니다.

* 템플릿

매개변수의 타입에 따라 함수나 클래스를 생성하는 매커니즘을 의미합니다.

템플릿을 사용하면 타입마다 별도의 함수나 클래스를 만들지 않고, 여러 타입에서 동작할 수 있는 함수나 클래스를 작성할 수 있습니다.

* 오버로딩과 오버라이딩의 차이점

오버로딩은 함수명이 같고 배개 변수의 타입이나 개수가 다른 함수를 허용하는 것 이고

오버라이딩은 부모 클래스에서 정의한 함수를 자식 클래스에서 재정의하는 것을 허용하는 것입니다.

* Malloc, calloc, new 차이점

Malloc은 라이브러리가 제공하는 함수이지만 new는 언어가 제공하는 연산자 입니다.

Malloc은 기본적으로 사이즈를 매개변수로 받고, 반환타입이 void\*형 이므로 sizeof와 캐스트연산자의 도움을 받아야 하지만 new는 타입을 지정하면 됩니다.

그리고 New는 자동으로 생성자를 호출하여 초기화를 해 줄 수 있지만 malloc은 불가능합니다.

* Calloc

Calloc은 malloc과 비슷한 기능을 가지고 있지만 calloc은 malloc과 다르개 할당된 공간의 값을 모두 0으로 바꿉니다. 그러므로 초기화할 필요가 있을 경우 calloc을 사용하면 됩니다.

* 언제 malloc을 사용하고 new를 사용하나요?

재할당이 빈번하게 일어나는 경우 동적할당된 메모리 크기를 변경시켜주는 reallloc이 가능한 malloc을 사용하는 것이 옳다고 생각합니다.

* Realloc의 단점은 없나요?

새로운 영역을 할당 받더라도 기존의 내용을 잃어버리는 일은 없지만 할당한 메모리에 여러 개의 포인터가 공유하고 있다면 사용해서는 안됩니다.

왜냐하면 메모리가 부족해 확장을 하지 못하고 새로운 공간으로 옮긴다면 nullptr을 반환하고 원래 공간에 남아있는 주소를 잃어버리게 되기 때문입니다.(댕글링 포인터)

* 초기화 리스트

생성자가 길어지는 것을 줄일 수 있고 대입 연산자를 호출하지 않아 성능적으로 우수합니다.

그리고 const나 reference 변수와 같이 초기값이 필요한 변수를 초기화 할 수 있는 좋은 방법입니다.

* 가상함수

Virtual 키워드를 붙여 자식클래스에서 부모 클래스의 멤버 함수를 재정의할 수 있는 함수이며

자신을 호출하는 객체의 동적타입에 따라 실제 호출할 함수가 결정 됩니다.

* 가상 함수 테이블

가상함수를 가지는 클래스에 생성되며 가상 함수 주소들이 배열 형태로 존재하는 테이블이다.

동작 시 호출할 함수의 목록을 가상함수 테이블에 미리 저장하고 실행 중에 객체와 해당 객체의 함수 주소를 찾는 방법으로 동작합니다.

* 절차지향형 프로그래밍(c) vs 객체지향형 프로그래밍(c++)

절차지향 프로그래밍이란 순차적인 처리가 중요시되며 프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법입니다.

컴퓨터의 처리구조와 유사해 실행속도가 빠르다는 장점이 있지만 유지보수가 어렵고 실행 순서가 바뀌면 동일한 결과를 보장하기 어렵다는 단점이 있습니다.

객체지향 프로그래밍이란 데이터를 객체 단위로 나누어 그 객체들 가느이 상호작용을 통해 로직을 구성하는 프로그래밍 방식입니다.

장점으로는 상속을 통한 확장으로 코드 재사용에 용이하고 유지보수 시에 수정할 객체 내부의 매서드만 수정하면 된다는 장점이 있지만 느린 처리속도와 객체가 많으면 용량이 커진다는 단점이 있습니다.

* Call by Value 와 Call by Reference 차이

Call by value는 인자로 받은 값을 복사하여 처리 하기 때문에 원래 값이 보존된다는 장점이 있지만 복사를 하기 때문에 메모리 사용량이 늘어난다는 단점이 있고

Call by Reference는 복사하지 않고 직접 참조하기 때문에 빠르지만 원래 값이 영향을 받을 수 있다는 단점이 있다.

* Call by pointer

Call by Reference와 같은 방식이지만 nullptr 검사를 해야한다는 단점이 있다.

* 객체지향 언어 – OOP 특징

1. 추상화

* 객체들의 공통적인 속성과 기능을 중심으로 추상화 한다.

1. 캡슐화

* 실제로 구현되는 부분을 외부에 드러나지 않도록 정보를 은닉할 수 있다.
* 객체가 독립적으로 역할을 할 수 있도록 데이터와 기능을 하나로 묶어 관리한다.
* 데이터를 보이지 않고 외부와 상호작용을 할 때는 메소드를 이용하여 통신을 한다.

1. 상속성

* 부모 클래스가 자식 클래스에게 속성과 기능을 물려주어 속성과 기능을 동일하게 사용하는 것
* 기존 코드를 재활용 해서 사용함으로써 객체지향 방법의 중요한 기능 중 하나에 속한다.

1. 다형성

동일한 요청에 의해 서로 다른 방식으로 응답하는 것

* 클래스 4대 속성

1. 멤버 변수의 기본접근지정자를 private으로 두어 외부로부터 원치 않는 데이터변겨을 막는 은닉화
2. 데이터화 함수를 하나로 묶어 사용하는 캡슐화
3. 부모 클래스의 기능을 자식클래스에게 물려주는 상속성
4. 각기 다른 객체가 동일한 메시지를 통해 각기 다른 기능을 수행하는 다형성

* 정적 라이브러리 lib / 동적 라이브러리 dll

정적 라이브러리는 컴파일 시 실행파일에 포함되고, 동적 라이브러리는 실행 시 실행 파일에 포함됩니다. 정적 라이브러리는 실행 파일에 함수의 코드가 복사되어 실행 파일의 크기가 커지지만 컴파일 후에는 lib 파일이 없어도 실행 가능하고, 동적 라이브러리는 실행 파일에 함수의 코드는 복사되지 않고 함수에 대한 정보만 가지고 있기 때문에 파일의 크기는 작다.

하지만 프로그램 실행 시 항상 dll 파일은 존재해야 한다.

* 디자인 패턴

1. MVC패턴

* Model/View/Controller로 역할을 구분해서 설계하는 디자인 패턴이며 역할에 따라 확시랗게 분리하여 유지보스를 용이하게 하고 프로그램의 확장성과 유연성을 높일수 있다는 장점이 있는 패턴입니다.

1. Mediator 패턴

여러 객체가 서로 상호작용이 많은 경우 유지보수나 확장에 불리합니다.

이런상황의 객체들을 캡슐화 함으로써 결합력을 낮추는 가능을 하는 패턴입니다.

하지만 특정 로직에 맞춰져있기 때문에 재사용이 힘들다는 단점이 있습니다.

* 포폴에 CMediator라는 클래스가 있는데 Mediator 패턴을 활용하여 구현한 것이 맞습니까?

사실 처음 구현하고 사용했을 때는 Mediator 패턴이라고 생각했는데 객체간의 상호작용이

많이 없다는 점에서 Mediator패턴을 사용했다기 보다는 유사한 기능을 하는 객체들을 관리하는 하나의 클래스를 만들었다. 라고 보는 것이 맞다고 생각합니다.

1. 싱글톤 패턴

최초 한번만 메모리를 할당하고 그 메모리에 인스턴스를 만들어 사용하는 디자인 패턴이다.

그렇기 때문에 메모리 낭비를 방지할 수 있고 전역이기 때문에 다른 클래스의 인스턴스들이 데이터를 공유하기 쉽다는 장점이 있지만

싱글톤 인스턴스가 너무 많은 일을 하거나 많은 데이터를 공유시킬 경우 다른 클래스간에 결합도가 높아지게 되어 유지보수가 어려워진다는 단점이있고

멀티스레드 환경에서 동기화 문제가 발생할 수 있다.

* 멀티스레드 환경에서 어떻게 해결했나요 ?

DeltaTime을 업데이트하는 클래스를 싱글톤으로 사용하였기 때문에 매 프레임 연산을 해줘야 했습니다.

그래서 lock을 사용하는 방법이 아니라 멀티스레드가 돌기전에 미리 인스턴스를 호출하여 만들어 버리는 방식으로 해결했습니다.

STL

* 시퀀스 컨테이너

데이터를 선형으로 저장하며 특별한 제약이나 규칙이없는 컨테이너이며 삽입된 요소의 순서가 그대로 유지됩니다.

* Vector

벡터는 동적으로 확장 축소가 가능하며 배열처럼 연속적인 메모리 공간에 저장하기 때문에 개별 원소에 대한 접근 속도가 빠르고 index로의 접근이 가능합니다.

하지만 확장 시 재할당 비용이 크기 때문에 적절한 크기의 reserve로 인한 메모리 확보가 중요합니다.

* 중간에서 삽입/삭제가 이뤄지면 왜 느린가요?

연속적인 배열 형태이기 때문에 중간에서 삽입삭제가 일어나면 한칸씩 당겨야 하는 상황이 발생하기 때문입니다.

* 확장시 비용이 왜 큰가요?

확장시에 기존 벡터의 용량에 1.5~2배 정도 확장된 메모리를 새로 할당하여 기존 벡터를 옮겨 담는 작업을 수행하기 때문입니다. 이를 위해 resize나 reserve를 사용합니다.

* Resize와 reserve는 뭔가요?

Reserve 는 메모리만 할당하고 초기화 하지 않으며 size에는 영향을 주지 않습니다.

Resize는 벡터에 적절한 수의 요소를 삽입,삭제하여 주어진 크기로 만드는 역할을 하며

사용시 초기화 합니다.

* Emplace\_back과 push\_back의 차이는 뭔가요?

Push\_back을 사용할 때는 vector에 값을 넣기 위해 바깥에서 임시 객체의 생성자 / 소멸자가 호출되고 vector내부에 복사되는 과정에서 또 객체가 한번 더 생성되지만

Emplace\_back은 내부에서 한번만 객체가 생성되므로 대부분의 경우 emplace\_back이 성능적으로 좋기 때문에 사용하였습니다.

* Vector array 차이

벡터는 순차 데이터 저장 컨테이너이고 배열은 인덱스 기반의 고정 크기 컨테이너입니다.

즉, Vector는 동적으로 크기가 변경가능하지만 배열은 불가능합니다.

* 벡터와 덱의 차이

덱은 늘어나야 할 크기만큼 할당되기 때문에 Vector에 비해 확장 비용이 작다는 장점이 있지만

테이블 전체가 연속된 메모리 공간이 아니므로 원소들 간의 포인터 연산이 불가능 하다는 단점이 있습니다.

* List란?

중간에 데이터 삽입이나 삭제가 자주 일어나고 데이터를 랜덤하게 접근하는 경우가 많지 않을 경우 유용하게 사용되는 컨테이너입니다.

컨테이너의 어느 위치에서도 삽입삭제가 상수시간이라는 장점이 있지만 인덱스에 의한 접근을 지원하지 않는다는 단점이 있습니다.

* Set이란?

고유한 key 값을 저장하는 연관 컨테이너이며 인덱스 접근을 허용하지 않지만 search와 삽입삭제에서 log n의 시간복잡도를 가집니다.

* Map이란?

고유한 key와 key의 Value값을 저장하는 연관 컨테이너이며 인덱스 접근을 허용하지 않지만 search와 삽입삭제에서 log n의 시간복잡도를 가집니다.

* Map이 내부적으로 어떤 구조로 되어있는지 아시나요?

균형이진탐색트리인 레드블랙트리로 구현되어 있습니다. 이진탐색 트리에서 최악의 경우 트리의 높이만큼 탐색 시간이 걸리는 상황을 해결하기 위해 균형잡힌 트리가 만들어지도록

5가지 조건을 가지고 있는데 각 노드는 빨간색, 검정색 중 하나이다.

루트노드는 검정색이여야 한다, 모든 리프 노드는 블랙이다(NIL노드), 레드노드의 자식은 블랙이다, 리프노드에서 루트 노드까지 가는 경로의 블랙 노드의 개수는 같다.

이 조건을 만족함으로써 트리의 높이를 log n에 바운드되도록해줍니다.

* Unoredered\_map map 차이점

맵은 균형 이진 트리를 기반으로 동작하며 언오더드는 해쉬 테이블 기반으로 동작합니다.

그래서 map은 key값에 따라 정렬되지만 언오더드는 key의 hash 값에 따라 저장되며 요소를 자동정렬하지 않습니다.

* Unordered\_map

해싱된 키를 가지고 배열의 인덱스로 접근하기 때문에 삽입, 삭제 검색이 매우 빠르다는 장점이 있지만 해시 충돌이 발생하면 성능이 떨어집니다.

* 해시충돌이 일어나는 이유는 뭔가요?

Key는 무한하지만 bucket의 사이즈는 유한하기도하고 key가 서로 다르더라도 동일 값을 반환할 수 있기 때문입니다.

* 해시 충돌을 최소화 하는 방법에 대해 아시나요 ?

버킷의 사이즈가 충분히 크거나, 버킷의 크기를 소수로 설계하는 것입니다.

잘은 모르지만 소수가 어떤 계산으로부터 나올 확률이 적기 때문으로 알고있습니다.

* 해시 충돌의 해결법에 대해서 아시나요?

체이닝과 개방주소법가 있습니다.

체이닝이란 value가 저장되는 공간을 가변 배열로 구현하여 충돌이 발생해 동일한 인덱스가 나오면 해당 인덱스에 추가로 이어서 저장하는 방법입니다.

개방주소법은 충돌 발생시 비어있는 인덱스에 value를 할당하는 방법입니다.

비어있는 인덱스를 탐색하는 방법으로는 다음인덱스를 확인하며 찾아가는 선형탐사, 선형탐사와 비슷하지만 폭을 이차식을통해 계산하는 제곱탐사, 또 다른 해시 함수를 사용해 증가 폭을 구하고 증가폭만큼 인덱스를 증가시키며 빈곳을 탐색하는 이중해싱이 있습니다.

* Queue

큐는 선입선출 방식의 자료 구조이며, 한쪽 끝에서 삽입 작업을, 다른 쪽 끝에서 삭제작업을 진행하는 구조입니다.