**CONTAINERS**

E’ tipo arraylist

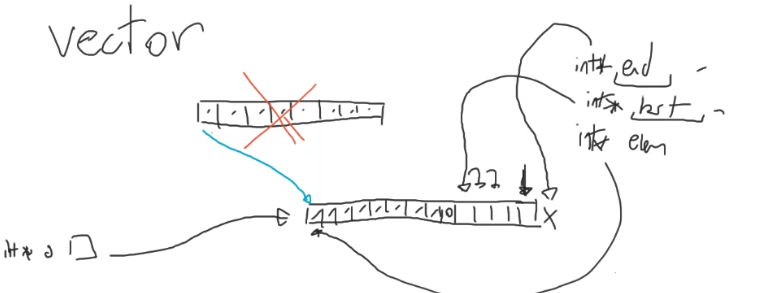
Prima: devo ingrandire un array, devo:

* Farne uno nuovo nell’heap
* Copiare
* Eliminare il vecchio
* Far puntare il puntatore del vecchio al nuovo

**NOTA:**  in ListvsVector.cpp ci sono i vari confronti

Classe **Vector:**

Semplificano lavori sugli array



I 3 puntatori indicano:

* Inizio
* Ultimo elemento diverso da 0 o null
* Indirizzo di memoria dopo l’ultimo elemento

Il vector tratta tutte ste cose in automatico, inoltre utilizza lui copy constructor ecc

**NB:** fare #include<vector>

Utilizzo:

std::vector<NomeClasse> = {{nome, cognome},{nome2, cognome2} };

auto it = vec.begin(); //restituisce l’iteratore in posizione 0

(è random access, con il Vector l’iteratore può fare tutto)

vec.end(); //restituisce quello dopo il contenitore/vector

it = vec.inser(it, 200); //Inserisce 200 nella posizione di it

**NB:** it è un iteratore, tipo puntatore o indice, serve a scorrere le robe (è di tipo std::

**NB2:** gli iteratori che “puntavano” elementi successivi a quello inserito rimangono fermi, quindi non puntano più gli stessi elementi (quelli prima vanno bene, ma smettono di andare bene se si deve aggiungere uno spazio al vector)

Non posso fare due insert di fila, prima devo riposizionare it (maybe?)

It = vec.begin();

it = vec.insert(it+2, 2, 200); //Inserisce due 200 nelle posizioni a partire da it + 2 posizioni

guerrueru.push\_back(oggettoGuerriero); // lo aggiunge alla fine

**NB:**  per il polimorfismo servono comunque i puntatori, si può fare

Vector<unique\_ptr<Guerriero>> puntatoriGuerrieri;

puntatoriGuerrieri.push\_back(unique\_ptr<Guerriero> oggettoGuerriero);

**NB:** posso accedere al vector anche facendo puntatoreGuerrieri[2] ma se vado su altri indirizzi non mi da errori (come con gli array)

Posso invece fare puntatoreGuerrieri.at(2) fa un controllo ed interrompe il programma se si va fuori

Vec.emplace (it, 5); //se deve ingrandire l’array invece di copiare distruggere ecc crea su quell’array e basta

//Minuto 20 e qualcosa importante, riascoltare

\_\_\_\_\_

**LIST:**

avversario di Vector

std::list<int> listaInt = {9, 2, 5}

nelle liste gli iteratori sono super importanti, anche se li ha solo bidirectional (possono solo fare avanti e indietro, ma si può usare std::advance per andare in indici specifici)

erase(it1, it2); //elimina gli elementi da it1 a it2 (escluso)

lIntIt = erase(lIntIt); //elimina l’elemento ma lo sa e lIntIt rimane valido

Super utile:



Grafo?

**LIST O VECTOR? MINUTO 42 PER RIASCOLTARE (è pure scritto nel file lol)**

Dipende da quale accesso mi conviene, se randomico o bidirezionale

Se elimino/aggiungo sempre alla fine conviene la LIST che è bidirezionale

**ITERATOR:**

**NB:** tutte le funzioni possono essere usate sia da Vector che da List??

Vector e List hanno un iterator diverso, anche se i comportamenti di base sono gli stessi.

//ASCOLTARE MINUTO 50 PER REVERSE ITERATOR (iteratore ma l’inizio diventa la fine e viceversa?)

**MAPPE**

Assegna valori a chiavi (entrambe possono essere oggetti custom)

Vengono utilizzate per gli alberi binari (che sono ordinati è OP, si risparmia un sacco di tempo dato che ad ogni nodo si sa dove andare)

Std::map<TipoChiave, TipoValore> nomeMap;

Esempio

Std::map<string, int> mappa { {“Primo”, 1}, {“Secondo”, 2}

};

//1 ora 8 minuti mappe come container (complicato da leggere)

Iteratore->first; //valore chiave

Iteratore->second; //valore del valore xd

**NB:** la freccia dereferenzia anche (\*iteratore).first; è equivalente

Se il find mi fa andare l’iteratore dove sta l’end, allora non ha trovato l’elemento ed è arrivato fino in fondo

**UNORDERED MAPS**

Key->hash

I valori hash (che vengono calcolati in base alla chiave) stanno abbastanza a caso nella memoria (ad ognuno corrisponde ad un oggetto)

Però sono elencati in una tabella

Bucket: insieme degli oggetti con la stessa hash

**NB:** molto rapido se si deve trovare un oggetto a partire dalla key dato che si calcola l’hash e ci va direttamente, senza scorrere nulla.

//1 ora e 20 circa rivedersi sta roba bruh

Size\_t: valore numerico in sequenza di bit

//1 ora e 30 le cose diventano molto confuse

//1 ora e 34 spiegazione operazioni bit