1. **Cosa sono le Classi?**

Le classi son delle particolari strutture dati ci permettono di definire nuovi tipi di dato astratti.

Una classe è generalmente formata da una parte privata e una parte pubblica:

* Nella parte privata generalmente troviamo i membri dato, le funzioni di utilità e il mascheramento degli operatori.
* Nella parte pubblica solitamente troviamo le funzioni membro (le quali sono le uniche, insieme alle funzione friend a potere accedere alla parte privata)

Le classi occupano solitamente due file: il file di intestazione (.h) e il file di implementazione (.cpp): nel file di intestazione troviamo la definizione della classe e la dichiarazione delle funzioni membro (le quali però potrebbero anche essere definite inline).

Nel file di implementazione troviamo la definizione delle funzioni membro.

1. **Coso sono i costruttori?**

I costruttori sono delle particolari funzioni membro, il cui nome è lo stesso della Classe.

I costruttori servono a mettere in uno stato consistente (inizializzare) le istanze della classe.

Esiste il costruttore predefinito della classe, ma possiamo anche ridefinire noi i costruttori, i quali possono essere di diverso tipo:

* Costruttore con argomenti
* Costruttore senza argomenti (o di default)
* Costruttore di conversione (vedi n° 4)

I costruttori non possono essere chiamati esplicitamente, ma verranno chiamati automaticamente dal compiler, seguendo delle regole di chiamata:

* + 1. Per gli oggetti statici, all’inizio del programma;
    2. Per gli oggetti automatici, quando viene incontrata la dichiarazione;
    3. Per gli oggetti dinamici quando viene incontrato l’operatore new
    4. Per gli oggetti membro dato di un altro oggetto, quando viene chiamato il costruttore di quest’ultimo.

Attenzione: non è lecito definire un oggetto di tipo classe senza inizializzarlo se è presente solo ed esclusivamente il costruttore con parametri.

1. **Cosa sono i distruttori?**

Il Distruttore è una particolare funzione membro, il cui nome è ~Classe e che non ha parametri o valori di ritorno, che si occupa di deallocare la memoria utilizzata dagli oggetti classe.

È fondamentale ridefinirli quando una classe utilizza la memoria dinamica.

Come i costruttori, non possiamo chiamare esplicitamente i distruttori, i quali vengono chiamati dal compilatore, attraverso delle regole di chiamata.

* + 1. Per gli oggetti statici, alla fine del programma
    2. Per gli oggetti automatici, alla fine del blocco nel quale sono definiti
    3. Per gli oggetti dinamici, quando viene incontrato l’operatore delete
    4. Per gli oggetti membro di altri oggetti, quando questi ultimi vengono distrutti.

Nota: per gli oggetti membro dato di altri oggetti: questi vengono costruiti nell’ordine di “apparizione” e distrutti nell’ordine inverso.

1. **Costruttori e operatori di Conversione**

Il costruttore di conversione è quel particolare Costruttore che ha come parametro un unico argomento di tipo dato fondamentale.

L’operatore di conversione è quell’operatore che converte un tipo dato classe in un tipo dato fondamentale e la sua sintassi è la seguente:

operator type();

Si omette il valore di ritorno in quanto è implicito nel nome.

1. **Operator**

Gli operator sono delle particolari funzioni (globali o membro) il cui identificatore è ***operator*** seguito dall’operatore che si vuole ridefinire.

Il linguaggio C++ ci permette di ridefinire gli operatori che operano sugli oggetti di tipo classe.

NON è lecito ridefinire operatori che operano tra tipi dati fondamentali (es: la somma tra interi) e non è possibile definire nuovi operatori.

Gli operatori possono essere definiti come funzioni friend o funzioni membro:

* Operatori unari: un parametro se friend, nessuno se funzione membro
* Operatori binari: due parametri se friend, uno se funzione membro

Se ridefiniamo un operatore come funzione membro, l’operando di sinistra sarà sempre di tipo classe: per mantenere la simmetria degli operatori, si utilizzano le funzioni friend.

È compito del programmatore mantenere le equivalenze tra operatori (es: ++x e x+=1).

Si possono ridefinire tutti gli operatori eccetto:

* Risolutore di visibilità ::
* Selettore di membro .
* Selettore di membro attraverso puntatore a membro .\*

Gli operatori di assegnamento (e derivati), di indicizzazione e di chiamata a funzione devono essere sempre ridefiniti come funzioni membro!

1. **Costruttore di copia**

Il costruttore di copia predefinito esegue una ricopiatura membro a membro dei membri dato tra i due oggetti.

Questo viene chiamato quando:

* + 1. Creiamo un oggetto di tipo classe e lo inizializziamo attraverso un altro oggetto della stessa classe preesistente.
    2. Passiamo per valore un oggetto di tipo classe.
    3. Una funzione restituisce per valore un oggetto di tipo classe.

È importante ridefinirlo quando la classe utilizza la memoria dinamica.

La sintassi della sua dichiarazione è:

Classe(const Classe&)

Passiamo un riferimento perché al momento della definizione, per le regole di chiamata del costruttore di copia, cercheremo di chiamare infinite volte il costruttore di copia che stiamo definendo.

Costante perché non vogliamo modificare erroneamente l’operando di destra.

1. **Operatore di assegnamento**

Di default esegue una ricopiatura membro a membro, per questo motivo è importante ridefinirlo quando la classe utilizza la memoria dinamica.

L’operatore di assegnamento deve:

* Controllare che l’operando di destra sia diverso dall’operando di sinistra (anti-aliasing)
* (se la quantità di memoria occupata dall’operando di sinistra è diversa da quello di destra) deallocare la memoria dell’operando di sinistra e allocare memoria della stessa dimensione dell’operando di destra.
* Ricopiare membro a membro.

La sua dichiarazione è:

Class& operator=(const Class&)

1. **Operatore di incremento (prefisso-postfisso)** *[Vale la stessa cosa per il decremento]*

L’operatore di incremento unario prefisso restituisce un lvalue, e per questo è possibile concatenare più incrementi.

L’operatore postfisso, restituisce un rvalue e non è possibile concatenare più incrementi.

Questi operatori possono essere ridefiniti anche per le classi.

Le dichiarazioni sono rispettivamente:

Classe& operator++();

Classe operator++(int);

Dove il primo, come si vede restituisce un riferimento, il secondo un valore.

Il primo è senza argomenti, il secondo invece ha un argomento fittizzio, se non avesse argomento, non si potrebbe distinguere la differenza, dato che il valore d ritorno non costituisce motivo di overloading di funzioni.

1. **Funzioni friend**

Le funzioni friend, sono le uniche oltre alle funzioni membro, a potere accedere alla parte privata della classe.

Le funzioni friend devono essere dichiarate nella classe e definite come funzioni globali.

Possono essere friend di una classe non solo le funzioni, ma anche altre classi o funzioni di altre classi.

1. **Proprietà degli operatori**

Le proprietà degli operatori sono 4:

* + 1. Posizione: prefissa, infissa, postfissa.
    2. Priorità: l’ordine in cui più operatori in una stessa espressione devono essere eseguiti
    3. Arietà: numero di oggetti su cui può agire un operatore, questo numero varia tra 1 e 2, eccetto per l’operatore condizionale, l’unico operatore ternario
    4. Associatività: a parità di priorità, quali operatori devono essere utilizzati per primi? L’associatività può essere a destra o a sinistra.

1. **Cosa sono i puntatori?**

Il tipo dato puntatore è un tipo di dato derivato dai tipi dato fondamentale.

Una variabile di tipo puntatore a... occupa sempre 4 byte e ha come rvalue, l’ lvalue di un altro oggetto del tipo a cui quel puntatore si riferisce, nel senso: se abbiamo una variabile di tipo puntatore ad intero, questa conterrà l’indirizzo di memoria di una variabile di tipo intero.

Per i puntatori è definito l’operatore di dereferenziazione, il quale ci permette di accedere al valore destro della variabile puntata dal puntatore.

Per questo motivo è importante inizializzare i puntatori (a NULL o con l’indirizzo di un’altra variabile), poiché se non inizializzato il puntatore, interpreta il valore destro già presente in quella cella di memoria come un indirizzo e punta a una cella di memoria che non conosciamo ed utilizzando l’operatore di dereferenziazione potremmo modificare involontariamente altre variabili del nostro programma o potremmo modificare celle di memoria che non sono di nostra competenza.

Quest’errore inoltre, essendo un errore logico e non sintattico, non verrebbe segnalato dal compilatore.

Gli operatori di dereferenziazione si possono concatenare.

Nel caso cercassimo di dereferenziare un puntatore a NULL si verificherebbe una Segmentation Fault.

Per i puntatori è definita un’aritmetica, la quale ci permette, attraverso somme e sottrazioni di accedere alle celle di memoria vicine a quella puntata dal puntatore: quest’aritmetica è molto importante in quanto è alla base degli Array.

I puntatori possono essere costanti o (definendoli impropriamente) puntatori a costanti:

* I puntatori costanti, sono quei puntatori il cui valore destro non può essere modificato, ma può essere modificato il valore destro della variabile puntata.
* I puntatori a costanti non possono modificare il valore destro della variabile puntata, il quale però può essere modificato da altri alias, a patto che non sia una variabile costante.

1. **Cosa sono i riferimenti?**

I riferimenti sono “delle variabili fantasma” o più semplicemente un alias per la variabile a cui si riferiscono.

I riferimenti non possono essere riferiti a loro volta e devono essere inizializzati al momento della creazione.

Risultano molto utili per i passaggi di argomenti ala funzioni.

1. **Cosa sono le C-stringhe?**

Le c-stringhe sono array di caratteri il cui ultimo carattere è quello di terminazione ‘\0’; in questo modo le stringhe possono essere stampate o prese in ingresso.

A differenza degli Array, si può utilizzare l’operatore di assegnamento ma solo al momento dell’inizializzazione, che può avvenire anche attraverso un letterale.

1. **Array e Matrice**

Un array è una sequenza di componenti dello stesso tipo, memorizzati in celle di memoria contigue.

L’array è un tipo di dato derivato sul quale non sono definite le operazioni di assegnamento e di confronto.

Ci si può riferire ai singoli componenti degli array attraverso l’operatore di indicizzazione [].

Le matrici sono anch’esse un sequenza di componenti dello stesso tipo memorizzate in celle di memoria contigue, le matrici, avendo due dimensioni vengono memorizzate per righe.

Sia per gli array che per le matrici ci si può riferire ai singoli elementi attraverso i puntatori sfruttando l’aritmetica dei puntatori.

Questo perché esiste delle equivalenze:

&v[0]=v

V[i] = \*(v+i)

Pensando ad una matrice per come è memorizzata nella memoria, si può accedere ai singoli elementi anche in questo modo:

\*(v+i\*C+j) equivale a [v+i\*C+j]

dove C è il numero di colonne e i e j sono rispettivamente gli indici di riga e colonna.

In questo modo la matrice si dice linearizzata.

1. **Preprocessore**

È quel programma la cui uscita viene preso in ingresso dal compilatore.

Viene chiamato in causa ogni volta che nel programma è presente il carattere ‘#’.

Tra le varie direttive ricordiamo:

* #include che serve per includere nel nostro file, altri file, come ad esempio le librerie. Quando vogliamo includere una libreria scriviamo:

#include <libreria> se è presente nella directory default delle librerie

#include “libreria.h” se è nella stessa directory del .c++ (aggiungere il path se sono in directory diverse)

* #define per definire un simbolo e un valore associato, utilizzato ad esempio per definire una costante: sconsigliato poiché il preprocessore ogni volta che incontra quel carattere sostituisce il suo valore senza effettuare nessun controllo sul tipo.

Questa direttiva però è anche utilizzata per altri scopi.

* #ifdef #elif #else istruzioni condizionali del preprocessore, ci permettono di mandare al compilatore solo i “rami attivi” attivi dell’if.

1. **Spazio di nomi**

Insieme di dichiarazioni e definizioni tra parentesi graffe.

Per utilizzare uno spazio di nomi si può utilizzare l’operatore di risolutore di visibilità o la keyword using: quest’ultima potrebbe creare ambiguità (segnalate dal compiler) nel caso in cui lo stesso identificatore sia usato in due spazi di nomi distinti.

Esiste lo spazio di nomi senza nome il quale racchiude tutte le dichiarazioni e definizioni fuori dalle graffe (es: variabili globali).

1. **Unità di compilazione**

È quel file sorgente generato dal pre-processore e dato in ingresso al compilatore per creare il file .object.

Il linker si occuperà di collegare i vari file oggetto per creare l’eseguibile.

È importante che ci sia un solo file con il metodo main.

1. **Programmazione a moduli**

È un tipo di programmazione basato su concetto di information hiding.

In questo tipo di programmazione sono presenti più unità di compilazione che saranno poi collegate per creare l’eseguibile.

Solitamente sono presenti almeno un file .h e uno (o più) .cpp.

Questo tipo di programmazione è utilizzato ad esempio per creare le librerie.

La programmazione a moduli risulta molto utile poiché, dato che solitamente nel file .h sono presenti dichiarazioni delle funzioni, nel caso si dovesse cambiare l’implementazione di qualcuna di esse, si dovrebbe ricompilare solo il file di implementazione.

Se invece si modifica il file .h, si devono ricompilare tutti i moduli che la includono.

Nella programmazione a moduli si utilizza il concetto di collegamento, il quale può essere interno o esterno:

* Hanno collegamento interno le var. costanti e le strutture; ovvero possono essere “viste” solo dall’unità di compilazione dove sono definite.
* Hanno collegamento esterno le variabili globali e le funzioni: per utilizzare le prime è necessario utilizzare la keyword extern, per le se seconde basta scrivere la dichiarazione.

Per non dover ricopiare ogni volta le dichiarazioni, queste solitamente si scrivono nel .h, nel quale saranno presenti anche le costanti e le strutture nell’eventualità che esse debbano essere condivise tra più file .cpp ( ovvero dai file che includeranno il file di intestazione).

Per rendere le variabili (o le funzioni) con collegamento interno, si utilizza la parola chiave static.

1. **Concetto di stream e ridefinizione degli operatori**

Lo stream è una struttura logica formata da una sequenza di bit.

Gli stream di ingresso e di uscita non sono standardizzati nel linguaggio, ma si possono effettuare includendo la libreria <iostream>.

Cin e cout sono, rispettivamente, istanze della classe istream e ostream.

Lo stream può essere rindirizzato, ad esempio su un file.

Nella sua struttura lo stream presenta un unsigned int, tra i vari bit dell’unsigned ricordiamo:

* Failbit: se è a 1, lo stream è in una situazione di errore recuperabile;
* Badbit: se a 1, lo stream è in una situazione irrecuperabile;
* Goodbit: se a 1, lo stream è in uno stato consistente;
* Eofbit: bit di fine file/ fine stream.

**Stream su file:**

per effettuarlo bisogna importare le librerie ifstream (per l’ingresso) e ofstream (per l’uscita) o ffstream (per entrambi).

Sia per la scrittura che per la lettura è importante specificare per quale motivo stiamo aprendo lo stream e chiudere lo stream, una volta finite tutte le operazioni.

Casi particolari stream di uscita su file:

* Quando apriamo un file (quando utilizziamo la libreria ffstream) dobbiamo specificare se lo apriamo in sovrascrittura o in scrittura in coda.
* Lo stream di scrittura potrebbe fallire per due motivi: non possediamo i diritti per poter scrivere in quel file, oppure, al momento della creazione di quel file la memoria è insufficiente.

Casi particolari stream di ingresso da file:

* Lo stream potrebbe andare in una situazione di errore recuperabile se tentassimo di leggere un tipo di dato diverso da quello effetivamente presente.
* Lo stream potrebbe andare in una situazione di errore irrecuperabile in caso di file non presente di assenza dei diritti di lettura su quel file.
* Così come per lo stream di ingresso da tastiera, esiste la funzione di libreria clear() che rimette lo stream in una condizione di good, così da poter riprendere a leggere da tastiera/file. Passano come parametro alla funzione ad esempio ios::failbit possiamo modificare lo stato del failbit.

Se volessimo ***stampare a video o leggere in ingresso tipi di dato*** classe dobbiamo ridefinire gli operatori della classe iostream.

Per questo ridefiniamo l’operatore di ingresso e di uscita come funzioni friend della classe di cui dobbiamo eseguire lo stream.

La sintassi delle dichiarazioni degli operatori sono:

ostream& operator<<(ostream&, const Class&);

istream& operator>>(istream&, const Class&);

La funzione ha come parametri un tipo dato ostream riferimento perché dobbiamo modificare quel riferimento e una sua copia (regole di chiamata costruttore di copia) ed un riferimento costante ad un oggetto della classe da stampare per non chiamare il costruttore di copia. La funzione restituisce il riferimento a quello stream, così da poter concatenare gli stream.

1. **Membri dato statici**

I membri dato statici sono dei membri dato che non si riferiscono alle singole istanze della classe, ma si riferiscono alla classe intera. Possono essere utilizzati ad esempio per contare quanti oggetti di quella classe ci sono in “vita” contemporaneamente.

Per riferirsi ai membri dati statici della classe si utilizza il risolutore di visibilità (a patto che essi siano pubblici).

Esistono anche la funzioni statiche: queste non possono accedere ai membri privati della classe e non puntatore this, in quanto si riferiscono alla classe e non ad un oggetto.

Anch’esse si chiamano attraverso risolutore di visibilità.

1. **Lista di inizializzazione**

La lista di inizializzazione si può definire solo per i costruttori.

Questa serve per inizializzare i membri dato costanti o di tipo riferimento, o i membri dato di tipo classe.

La lista di inizializzazione si definisce al momento della definizione del costruttore, dopo i parametri formali e prima del corpo della funzione, attraverso i due punti ‘:’.

Inizializziamo le costanti e riferimenti lì poiché, per definizione di quei tipi di dato, non si possono assegnare valori, ad esempio all0interno del corpo del costruttore.

Facciamo la stessa cosa per i membri dato Classe così da chiamare implicitamente i costruttori di quella classe.

1. **Funzioni membro con attributo const**

Sono funzioni membro che non possono accedere in scrittura ai membri dati.

1. **Array di oggetti di tipo classe**

Si possono creare array di oggetti di tipo classe. Al momento dell’inizializzazione dell’array, i singoli componenti possono venir inizializzati con il costruttore con parametri, quelli che non vengono inizializzati in questo modo, vengono inizializzati dal costruttore di dafault.

Questa inizializzazione non è una vera e propria chiamata esplicita al costruttore!

Gli array di oggetti di tipo classe possono essere allocati anche nello heap: in questo caso, dato che l’operatore new è al massimo binario, i singoli elementi vengono inizializzati dal costruttore di default, per questo motivo è fondamentale che sia presente!

1. **Puntatore this**

È il puntatore predefinito che può essere utilizzato dalle funzioni membro per riferirsi all’oggetto su cui agiscono.

Non è presente nelle funzioni statiche!

1. **Grammatica del linguaggio**

Insieme di regole che definiscono un linguaggio, attraverso sintassi e semantica.

Sintassi: insieme di regole per la composizione delle frasi.

Semantica: significato delle frasi.

1. **Classi di memorizzazione**

Esistono tre classi di memorizzazione:

* + 1. Statica: le variabili statiche vengono allocate all’inizio del programma e deallocate alla fine.
    2. Automatiche: le variabili automatiche vengono allocate automaticamente quando si incontro la loro dichiarazione all’interno del blocco. Vengono deallocate all’uscita dal blocco.
    3. Dinamica: le variabili dinamiche vengono allocate quando si incontra l’operatore new e deallocate quando si incontra l’operatore delete.

Note: la dimensione delle variabili statiche e automatiche è conosciuta a tempo di compilazione, quella delle variabili dinamiche viene conosciuta in run-time.

L’operatore new è un operatore al massimo binario, che restituisce un lvalue.

1. **Cortocircuito**

La tecnica del cortocircuito riguarda gli operatori logici AND (&&) e OR (||).

Quando valutiamo una condizione (ad esempio in un if o in un ciclo) con l’operatore logico AND, se la prima condizione è falsa, la seconda non viene valutata.

Se invece abbiamo un OR, se la prima condizione è vera, la seconda non viene valutata.

Questo ci permette di evitare errori logici, come ad esempio dereferenziare un puntatore a NULL mentre scorriamo una lista.

1. **Effetti collaterali**

Detti anche Side-Effects, si hanno quando una funzione modifica elementi esterni alla funzione. Questo può avvenire solo se abbiamo a che fare con riferimenti e puntatori.

* **Nota su puntatori e funzioni:** dato che i puntatori possono essere valori di ritorno di una funzione, se il puntatore che restituiamo, punta ad una variabile locale alla funzione, commettiamo un errore logico, in quanto alla fine dell’esecuzione della funzione la variabile verrà deallocata e il puntatore punterà ad una zona di memoria il cui valore non sarà più quello della variabile locale alla funzione.

1. **Cosa sono le liste**

Una lista è una struttura dati i cui elementi, tutti dello stesso tipo, sono collegati tra loro in una sorta di catena potenzialmente infinita.

Gli elementi della lista sono formati da:

* Una parte informativa: le informazioni sull’elemento (int, struct, ecc.)
* Una parte puntatore: che punta all’elemento successivo della lista.

In una lista è importante che il primo elemento, la testa, punti al primo elemento e che l’ultimo elemento abbia il puntatore a NULL.

Questo perché altrimenti, non sarebbe possibile svolgere alcuna operazione sulle liste.

Memorizziamo le liste nello Heap, altrimenti i nostri discorsi non potrebbero “reggersi in piedi”.

L’accesso ai singoli elementi della lista è sequenziale, quindi per accedere all’n-esimo elemento, si devono scorrere tutti i precedenti.

1. **Operazioni sulle liste**

Sulle liste è possibile effettuare diverse operazioni:

* Inserimento in testa: questo inserimento non può mai fallire, viene creato un puntatore a struttura, il cui puntatore è la testa. La testa diventerà l’elemento aggiunto.
* Inserimento in coda: si utilizza la tecnica del doppio puntatore per scorrere la lista, si riutilizza il puntatore a NULL per creare un nuovo elemento, si controlla se lista è vuota oppure no:
  + Se non è vuota, l’ultimo elemento punterà al nuovo, il quale avrà come puntatore NULL;
  + Se è vuota, diventa un inserimento in testa.
* Estrazione in testa: l’estrazione in testa può fallire se la lista è vuota. Altrimenti, un puntatore di appoggio punterà al primo elemento, mentre la testa punterà all’elemento successivo.

Si elimina il puntatore del primo elemento al successivo e si elimina l’elemento.

* Estrazione dalla Coda: si scorre la lista con il doppio puntatore finchè il puntatore davanti non arriva all’ultimo elemento, a quel punto si controlla se il primo puntatore sia diverso dalla testa:
  + Se è diverso, il puntatore che precede diventerà l’ultimo e si elimina l’elemento.
  + Se è uguale, si elimina l’elemento e la testa diventa NULL.
* Estrazione di un elemento dato dalla lista.
* Stampa della lista.

1. **Cos’è una struct e una union**

La struct è una collezione di informazioni su un determinato oggetto.

La struct è un tipo di dato aggregato. Non sono definite le operazioni aritmetico logiche sulle struct.

Sulle struct sono definiti l’operatore di selettore di membro e l’operatore “freccia”:

* L’operatore selettore di membro (.) ci permette di accedere ai campi dato della struct.
* L’operatore -> implica una dereferenziazione e una selezione di membro, usato quindi quando abbiamo un puntatore a struct.

Possono essere passati come parametri alle funzioni e possono essere valori di ritorno delle funzioni.

Le Union si comportano come le Struct, la differenza tra le due è che nella prima, i campi dato condividono le celle di memoria, nella seconda ognuno ha le proprie celle di memoria.

Perché quindi utilizzare le Union, dato che scrivendo su un campo dato, modifichiamo anche l’altro (sovrascrivendolo totalmente o parzialmente)? Per risparmiare memoria e quando sappiamo che un elemento può essere di un tipo o di un altro ma non entrambi contemporaneamente.

1. **Algoritmi ordinamento di un vettore**

Gli algoritmi di ordinamento di un vettore sono:

* Selection Sort, viene cercato il valore minimo e viene scambiato con la prima posizione, alla seconda iterazione si parte dalla seconda cella, in quando la prima è già ordinata. Questa operazione viene effettuata per n volte quante sono le celle del vettore. Il selection sort non è in grado di sapere se il vettore che gli viene passato è già ordinato.
* Bubble Sort, vengono confrontati i valori adiacenti e viene spostato verso il fondo il valore più alto, fino a sistemare nell’ultima cella il massimo tra i valori presenti.

Alla seconda iterazione i confronti saranno effettuati fino alla cella n-1-esima.

Attraverso una variabile flag, è possibile sapere se ad un certo p-esimo passaggio il vettore risulta già ordinato.

1. **Algoritmi ricerca in un vettore**

Algoritmo di ricerca lineare: viene cercato il valore richiesto, fino a quando non si incontra.

Algoritmo di ricerca binaria: si parte dal centro e si controlla se il valore corrisponde a quella cella è il valore cercato o se è un valore maggiore o minore. Supponendo che sia un valore minore, allora la ricerca riprende nella seconda metà dalla posizione k+1 alla posizione n-esima, ripartendo sempre dal centro tra queste due posizioni, fino a quando non si trova il valore cercato ( se presente)

IMPORTANTE: QUESTO ALGORITMO FUNZIONA SOLO SE IL VETTORE è ORDINATO!

1. **Rappresentazione dei numeri Reali**

I numeri reali sono rappresentati dal tipo fondamentale Double nel linguaggio c++.

Sono rappresentati secondo lo standard IEEE 754, nel quale è previsto un bit per la rappresentazione del segno, 8 per l’esponente (rappresentato in bias) e 23 bit per la mantissa, rappresentata attraverso l’algoritmo div e mod: il primo bit a uno non viene rappresentato perché implicito, dato che si utilizza la notazione esponenziale.

1. **Rappresentazione dei numeri interi**

I numeri interi (int) vengono rappresentati in complemento a due.