Di Ilaria Anfuso.

- ALGORITMI DI ORDINAMENTO
  Gli algoritmi di ordinamento servono ad ordinare una serie di numeri in ordine crescente (o decrescente).
  - BubbleSort: scambia di posto gli elementi array[i] e array[i + 1] se il secondo è minore del primo. Lavora per coppie di elementi.
  - SelectionSort→ ricerca il più piccolo valore dell'array e lo scambia con il primo elemento. Ricerca il successivo più piccolo e lo scambia per il secondo, ecc..
  - InsertionSort→ Ogni elemento dell'array viene spostato in un sotto-array ordinato. Il primo elemento forma il primo sottoarray. Il secondo elemento viene aggiunto a questo sottoarray e se è più piccolo del primo elemento viene messo per primo, e così via.

- RICERCA DICOTOMICA
  Avviene quando bisogna cercare un numero in una grande serie di numeri. Si divide la serie a metà e si evidenza il valore centrale. Se questo è maggiore del numero da cercare allora andremo a considerare la prima metà, altrimenti la seconda. Si ripete il procedimento ripetendo la divisione a metà, in questo caso saranno due metà di metà. La cosa utile della ricerca dicotomica è che andremo a considerare volta per volta solo metà della serie, guadagnando in efficienza e velocità.

- NAMESPACE
  Introducono un ulteriore livello di scope, il più alto. Sono contenitori di nomi.

- CLASSE ASTRATTA
  Una classe è astratta quando contiene dei metodi virtuali puri. Un metodo è virtuale puro quando non viene implementato. Nelle classi derivate dalla classe astratta è obbligatorio implementare quel metodo. Una classe viene detta INTERFACCIA se contiene solo metodi virtuali puri.

- COSTRUTTORE DI COPIA
  Un costruttore di copia permette di creare un oggetto inizializzandolo con i dati di un altro oggetto già esistente della stessa classe.
  Il costruttore di copia di default copia gli attributi membro a membro.
  E' necessario per gli oggetti che allocano memoria dinamica. Presenta un unico parametro formale che è un riferimento a oggetto costante dello stesso tipo della classe.
  X(const X &);
  viene chiamato nelle inizializzazioni e non negli assegnamenti.

- ARRAY, MATRICI
  Insieme di locazioni di memoria consecutive dello stesso tipo.

Il nome di un array è un puntatore costante al primo elemento dell'array. Le matrici sono array bidimensionali. Una matrice è un array di puntatori ad array.

- FUNZIONE RICORSIVA
  Una funzione ricorsiva è una funzione che richiama se stessa. Esempi sono il fattoriale, numero di FIbonacci e la torre di Hanoi. Ogni funzione ricorsiva è caratterizzata dal caso base(condizione di terminazione) e passo induttivo(chiamata ricorsiva). E' bene evitare le funzioni ricorsive a causa della loro poca efficienza. Qualsiasi funzione ricorsiva può essere scritta in forma non ricorsiva.

- PUNTATORI
  Una variabile puntatore è una variabile che contiene un indirizzo di memoria.
  Il carattere * è chiamato operatore di dereferenziazione e serve a definire puntatori. Il carattere & è chiamato operatore di referenziazione e serve ad estrarre l'indirizzo di una variabile, scritto in esadecimale. Il puntatore può essere modificato mediante riassegnamento di altri indirizzi di memoria. Se un puntatore viene incrementato ad esempio, esso punterà alla locazione di memoria successiva.

- PUNTATORI COSTANTI E PUNTATORI A COSTANTI
  Se un puntatore punta a una costante, il valore della locazione di memoria a cui punta non può essere modificata.
  *const int *ptr= &var;*
  Un puntatore costante è una variabile costante e non può subire riassegnamenti; punta sempre alla variabile a cui è stato inizializzato.
  *int * const ptr= &var;*
  Un puntatore costante ad una variabile costante è una variabile costante che non può essere riassegnata e non può modificare la variabile a cui punta.
  *const int * const ptr= &var;*

- PUNTATORI DOPPI
  Chiamati anche puntatori a puntatori, costituiscono il secondo passo nell'indirizzamento multi-livello della memoria.Usato per passare per "indirizzo" un parametro puntatore.

- PUNTATORE THIS
  Permette di identificare al momento della chiamata, l'oggetto di destinazione di una funzione membro.
  Nelle classi di riferisce alla variabile private.

- STACK
  (Pila). Porzione di memoria riservata allo stoccaggio dei dati utili all'esecuzione delle istruzioni contenute nel corpo delle funzioni. Vengono inseriti e prelevati dati in base al meccanismo LIFO(last in first out). Push→ deposito. Pop→ prelievo.

- PASSAGGIO PER VALORE O PER INDIRIZZO O PER RIFERIMENTO

Quando avviene un **passaggio per valore** di un parametro di una funzione il valore attuale del dato viene copiato sul record di attivazione dello stack. Dato che avviene la copia, il valore originario non viene modificato.
Per passare un parametro per **indirizzo** usiamo i puntatori.Quando chiamiamo il metodo usiamo metodo(&a). La segnatura del metodo sarà metodo(int *a). Il puntatore può operare modifiche mediante l'indirizzo usando l'operatore di dereferenziazione * (*a). Il passaggio di un array come parametro avviene sempre per indirizzo. Il codice di un metodo può modificare il valore della variabile usata come parametro attuale mediante operatore di referenziazione &.
Nel passaggio **per riferimento** il parametro formale diviene alias della variabile attuale. Ogni modifica del parametro formale all'interno del metodo si riflette nella variabile. Chiamo il metodo(b) e la segnatura è metodo(int &a).

- FUNZIONI INLINE
  Se definiamo un metodo *inline doublef()*, il compilatore sostituisce ad ogni chiamata di funzione il codice del corpo di f(). E' una soluzione efficiente.

- ALLOCAZIONE MEMORIA
  Variabili, array e matrici possono essere allocati staticamente o dinamicamente.
  - staticamente: Vengono salvati del segmento di memoria Data o Stack;
  - dinamicamente: Salvati nel segmento Heap(segmento di memoria adibito ai dati dinamici). Per dichiarare un oggetto dinamicamente bisogna usare il new che riserva un nuovo spazio di memoria. Dopo ogni new bisognerebbe usare il delete, che libera la zona di memoria. (memory leak)

  Un puntatore memorizzato nello Stack contiene l'indirizzo di memoria dove è memorizzato l'oggetto vero e proprio (nello Heap).

  L'**aliasing** è un effetto che si genera quando viene fatto puntare ad un puntatore, che puntava già a qualcosa, un'altra cosa. In questo modo la cosa a cui puntava prima viene persa e non è più possibile recuperarla. L'aliasing è una violazione del principio di incapsulamento.

- VARIABILE STATIC E FUNZIONE STATIC
  - variabile static

  In una classe una variabile static è una variabile comune a tutti gli oggetti della classe → variabili di classe. Si condivide l'informazione con tutti gli oggetti istanziati. Una variabile statica può essere modificata da tutti gli oggetti della classe e le modifiche saranno visibili a tutti gli altri oggetti. Per un attributo statico(di classe) si conserva un memoria una sola copia per tutti gli oggetti invece che una copia per ogni oggetto. Ciò è ovviamente un gran risparmio di memoria.
  - funzione static

  E' una funzione il cui comportamento è indipendente dall'oggetto di destinazione. → metodo di classe. Può fare riferimento solo a membri statici, non a variabili d'istanza. Non può fare riferimento a this.

- MODIFICATORE CONST

Una funzione membro const non può modificare gli attributi dell'oggetto destinatario. Vengono infatti dichiarate const le funzione get e non quelle set.
Un oggetto const può attivare solo funzioni membro const dunque non può essere modificato.

- FUNZIONI FRIEND
  Una funzione friend non è membro di una classe ma ha accesso ai membri private della classe. Può essere una funzione globale o membro di un'altra classe.
  Solo in alcune situazione usare le friend è accettabile nella OOP:
  - overloading di operatori binari
  - considerazioni di efficienza
  - relazioni speciali tra classi.

  Esistono anche classi friend di altre classi che possono accedere ai membri pubblici e privati o protetti di quella classe.
  L'uso della parola *friend* viola le leggi di incapsulamento.

- OVERRIDING VS OVERLOADING
  Sono due cose diverse. Si ha overriding solamente se la funzione della classe derivata ha la stessa identica intestazione della funzione della classe primaria, cambierà il corpo. E' infatti una sovrascrizione della funzione originale.
  Si ha overloading quando scriviamo una funzione con lo stesso nome simile a quella originaria ma cambia l'intestazione e il corpo. Non è una sovrascrizione ma una funzione separata e diversa dalla originaria. può essere usata quando abbiamo esigenza di eseguire la stessa operazione su dati di tipo differenti o numeri di argomenti differenti.

- OVERLOADING OPERATORI

  Un operatore overloaded può essere implementato come:
  - funzione non membro:; Il numero degli argomenti deve corrispondere con il numero degli operandi. Se viene dichiarato friend di una classe, ha accesso a tutti i membri della classe;
  - funzione membro: l'operando sinistro è implicitamente legato al puntatore this, quindi un operatore binario prevede un solo parametro formale, uno unario nessun parametro formale.

  Operatori *speciali:* incremento\decremento in forma post fissa o prefissa.
  L'overloading dell'operatore di *assegnamento* si rende necessario quando l'inizializzazione membro a membro da un oggetto non è sufficiente. Ad esempio quando la classe contiene puntatori ad aree allocate dinamicamente.
  L'operatore di *indicizzazione* [] va implementato come funzione membro, deve prevedere un parametro formale e se il tipo di ritorno è un reference allora può essere usato come lvalue.

- ELLIPSIS
  Offrono la possibilità di dichiarare funzioni che prendono un numero variabile di argomenti.

- EREDITARIETÀ'
  Definizione di gerarchie di classi.
  E' il meccanismo mediante il quale una classe acquisisce tutte le caratteristiche di un'altra classe definita in precedenza. E' importante nel *riutilizzo del software*. E' quindi un modo per creare una nuova classe partendo da una già esistente e aggiungendo nuovi attributi e funzioni. La classe derivata può aggiungere nuovi metodi o modificare quelli ereditati→ è possibile sovrascrivere il metodo (Overriding)
  REGOLA ISA: ogni classe derivata " is a" derivata da classe originaria.
  Per chiamare un metodo viene usato il risolutore di scope.
  Non è possibile ereditare: costruttori e distruttori; operatore di assegnamento; funzioni friend.

- COSTRUTTORI E DISTRUTTORI
  Un costruttore è una funzione che ha lo stesso nome della classe. Non restituisce nulla. Serve a inizializzare gli attributi della classe. Se il programmatore non esplica il costruttore viene chiamato quello di default.

  Il distruttore ha lo stesso nome della classe preceduto dalla tilde. Esso viene invocato quando l'oggetto deve essere distrutto:
  - quando viene chiamato delete
  - quando l'esecuzione del programma oltrepassa lo scope in cui l'oggetto è istanziato.
  Esso non distrugge realmente l'oggetto ma si occupa di operazioni di base sulla memoria prima di restituirla al sistema. Non è consentito l'overloading del distruttore. Quello del costruttore sì.

- AGGREGATI E COLLEZIONI, RELAZIONE PART-OF
  Nella relazione di **composizione**, l'oggetto contenuto non ha un vita propria, viene creato e distrutto insieme all'oggetto contenitore, che è responsabile della costruzione/distruzione, della inizializzazione ecc.(motore→ automobile). Si indica con il rombo pieno.
  Nella relazione di **aggregazione**, l'oggetto aggregato ha vita proprio anche senza l'oggetto contenitore (persona → automobile). Si indica con il rombo vuoto.
  Ciclo di vita degli oggetti indipendenti. Contenitore non responsabile della costruzione/distruzione dell'oggetto contenuto. Si deve creare il contenitore passandogli un puntatore all'oggetto contenuto.

- PUBLIC, PRIVATE e PROTECTED
  Sono modificatori di accesso per i membri della classe. Membri public sono visibili in ogni parte del programma; i metodi public costituiscono l'interfaccia della classe. Membri private sono visibili solo alle funzioni della classe; metodi private vengono chiamati metodi di servizio. Membri protected sono visibili alle funzioni della classe e delle classi derivate, ma non all'esterno.
  Se il modificatore non viene specificato, il compilatore considererà la variabile/funzione private.

- REFERENCE E PUNTATORI
  I puntatori sono molto comodi perché consentono il passaggio di dati di grandi dimensioni con la massima efficienza, ma hanno dei difetti: la sintassi può essere tediosa, il puntatore potrebbe puntare a nullptr quindi deve essere controllato, inoltre può essere riassegnato.
  Le **reference** furono introdotte per mantenere i vantaggi dei puntatori ma eliminare le complicazioni. La notazione <type> & indica una reference ad un oggetto di tipo type. La reference va inizializzata contestualmente alla sua dichiarazione con oggetto del tipo specificato. Rispetto ai puntatori che possono assumere valore nullptr, le reference sono sempre valide. Inoltre la reference non può essere riassegnata ad altro oggetto. Gli operatori di incremento e decremento operano sugli oggetti di cui sono alias le reference.

- TIPI DI RIFERIMENTI
  - riferimenti di tipo **lvalue** modificabile o non const: riferimenti ad oggetti che possono cambiare stato.
  - riferimenti **const**(con lvalue): il riferimento non potrà essere utilizzato per modificare l'oggetto di cui è alias.
  - riferimenti **const**(con **rvalue**): un rvalue(ad esempio 1) viene utilizzato per inizializzare il valore del riferimento.
    const int &var = {1};
    Il ciclo di vita dell'oggetto temporaneo con valore 1, che inizializza var, è identico a quello della reference, quindi verranno distrutti insieme.
  - riferimenti come parametri formali di funzioni
  - funzioni che restituiscono reference: utile nell'overloading di [];

- POLIMORFISMO
  Una classe polimorfa definisce una interfaccia comune a differenti implementazioni presenti in classi derivate. La catena di derivazioni deve essere pubblica e le funzioni dell'interfaccia devono essere dichiarate virtual. Il polimorfismo a tempo di esecuzione avviene mediante puntatori o reference.
  - In C++ classi astratte permettono polimorfismo. Grazie al polimorfismo due oggetti della stessa classe possono avere comportamenti differenti. Avranno interfaccia comune con metodi overrided.
  - Se le funzione dell'interfaccia sono dichiarate virtual si ottine il polimorfismo a run-time.
  - Polimorfismo + virtual= Polimorfismo dinamico(a tempo di esecuzione)
  - Overloading= Polimorfismo statico. La risoluzione avviene a tempo di compilazione.

- LISTE DI INIZIALIZZAZIONE
  Rappresentano del codice che viene scritto nella definizione del costruttore e che richiamano altri costruttori. Specificano gli argomenti (parametri attuali) per il costruttore selezionato. Variabili costanti a reference vanno necessariamente inizializzati.

- ADT
    - Abstract Data type. Definizione di dati astratti permette di rendere operazioni sui dati indipendenti dalla loro implementazione. es.Stack.

- EREDITARIETA MULTIPLA
  Una classe può derivare contemporaneamente da più di una classe base. La sintassi prevede di inserire uno specificatore di accesso per ogni superclasse.

- TEMPLATE
  E' possibile definire classe e funzioni template che operano su tipi generici. E' permesso trattare i tipi dei dati come parametri. Una funzione template è una sorta di modello dal quale il compilatore genera specializzazioni→ templace istantiation.
  E' possibile fare overloading di funzioni template(diverso numero di parametri).