ESERCIZIO 5
- TABELLE HASH -

F. MOGAVERO

Implementare una libreria di funzioni in **Linguaggio C++** per la gestione di una struttura dati dinamica di tipo **tabella hash** contenente **dati generici**, ovvero interi, float, stringhe, struct, ecc. Tale struttura dovrà essere implementata sfruttando le seguenti due **tecniche di risoluzione dei conflitti**: (1) **indirizzamento chiuso / hashing aperto / concatenamento**; (2) **indirizzamento aperto / hashing chiuso**.

Le funzionalità da realizzare sono di seguito elencate:

(1) **costruzione** e **distruzione** di una struttura dati;
(2) operazioni di **assegnamento** e **confronto** tra istanze diverse della struttura dati;
(3) operazioni specifiche della struttura dati (**inserimento/eliminazione** di un elemento);
(4) operazioni generiche sulla struttura dati (**controllo di esistenza** di un dato valore; operazioni di **applicazione di una funzione** a tutti gli elementi: **funzioni map**; operazioni di **accumulazione di un valore** estratto dagli elementi: **funzioni fold**; lettura della **dimensione**; **svuotamento** della struttura).

Al fine di poter testare adeguatamente le librerie sopra descritte, si richiede di definire (esternamente alle stesse, in un opportuno file di test richiamato dal "main") un insieme di procedure che implementi le seguenti funzionalità:

(1) **scelta dell'implementazione** (indirizzamento chiuso vs indirizzamento aperto) e del **tipo di dati** da gestire (interi, ecc.);
(2) **popolamento della struttura** precedentemente scelta con n valori del tipo opportuno generati casualmente, dove n è un parametro dato dall'utente in ingresso;
(3) **visualizzazione di tutti gli elementi** (effettuata per mezzo della **funzione map**);
(4) **controllo di esistenza** di un dato valore;
(5) **calcolo di una delle seguenti funzioni** (effettuato per mezzo della **funzione fold**) e relativa visualizzazione del risultato: prodotto per gli interi minori di n, somma per i float maggiori di n, e concatenazione per le stringhe con lunghezza minore o uguale a n, dove n è un parametro dato dall'utente in ingresso.

Da un opportuno menu, dovrà essere inoltre possibile operare sulla struttura scelta attraverso le funzioni di libreria di cui al punto (3). Infine, è necessario prevedere l'accesso alla funzionalità di test prevista dal docente.

Il codice sorgente prodotto dovrà seguire pedissequamente (sia nei nomi delle funzioni di libreria, sia nella strutturazione, gerarchia di classi e nei nomi delle diverse directory e file ".cpp" e ".hpp") la forma riportata nel template Exercise5.zip associato al presente esercizio.

ESERCIZIO 1
- VETTORI E LISTE DI DATI GENERICI -

F. MOGAVERO

Implementare due librerie di funzioni in **Linguaggio C++** per la gestione di strutture dati dinamiche di tipo **vettore** e **lista** per **dati generici**, ovvero interi, float, stringhe, struct, ecc.

Le funzionalità da realizzare sono di seguito elencate:

(1) **costruzione** e **distruzione** di una struttura dati;
(2) operazioni di **assegnamento** e **confronto** tra istanze diverse della struttura dati;
(3) operazioni comuni ai due tipi di strutture dati (**accesso non distruttivo** all'elemento iniziale, finale o avente uno specifico indice; **controllo di esistenza** di un dato valore; operazioni di **applicazione di una funzione** a tutti gli elementi: **funzioni map**; operazioni di **accumulazione di un valore** estratto dagli elementi: **funzioni fold**; test di **vuotezza**; lettura della **dimensione**; **svuotamento** della struttura);
(4) funzioni specifiche su vettore (**ridimensionamento** del vettore);
(5) funzioni specifiche su lista (**inserimento** di un dato valore in testa o coda; **rimozione** e **rimozione con lettura** dell'elemento in testa).

Al fine di poter testare adeguatamente le librerie sopra descritte, si richiede di definire (esternamente alle stesse, in un opportuno file di test richiamato dal "main") un insieme di procedure che implementi le seguenti funzionalità:

(1) **scelta della struttura** (vettore vs lista) e del relativo **tipo di dati** da gestire (interi, ecc.);
(2) **popolamento della struttura** precedentemente scelta con n valori del tipo opportuno generati casualmente, dove n è un parametro dato dall'utente in ingresso;
(3) **visualizzazione dell'elemento** iniziale, finale o avente uno specifico indice;
(4) **visualizzazione di tutti gli elementi** (effettuata per mezzo della **funzione map**);
(5) **controllo di esistenza** di un dato valore;
(6) **calcolo di una delle seguenti funzioni** (effettuato per mezzo delle **funzioni fold**) e relativa visualizzazione del risultato: somma per gli interi minori di n, prodotto per i float maggiori di n, e concatenazione per le stringhe con lunghezza minore o uguale a n, dove n è un parametro dato dall'utente in ingresso;
(7) **applicazione di una delle seguenti funzioni** a tutti gli elementi: $2n$ per gli interi, n^2 per i float, e "uppercase" per le stringhe.

Da un opportuno menu, dovrà essere inoltre possibile operare sulla struttura scelta attraverso le funzioni di libreria di cui ai punti (4) e (5). Infine, è necessario prevedere l'accesso alla funzionalità di test prevista dal docente.

Il codice sorgente prodotto dovrà seguire pedissequamente (sia nei nomi delle funzioni di libreria, sia nella strutturazione, gerarchia di classi e nei nomi delle diverse directory e file ".cpp" e ".hpp") la forma riportata nel template Exercise1.zip associato al presente esercizio.

ESERCIZIO 2
- PILE E CODE DI DATI GENERICI CON RAPPRESENTAZIONI MULTIPLE -

F. MOGAVERO

Implementare due librerie di funzioni in **Linguaggio C++** per la gestione di strutture dati dinamiche di tipo **pila** e **coda** di **dati generici**, ovvero interi, float, stringhe, struct, ecc. Tali strutture dovranno essere implementate concretamente sfruttando le strutture dati precedentemente sviluppate nell'Esercizio 1: (1) **vettori** con efficiente **ridimensionamento automatico** dello spazio allocato; (2) **liste**.

Le funzionalità da realizzare sono di seguito elencate:

(1) **costruzione** e **distruzione** di una struttura dati;
(2) operazioni di **assegnamento** e **confronto** tra istanze diverse della struttura dati;
(3) operazioni comuni ai due tipi di strutture dati (**inserimento** di un dato elemento; **rimozione**, **rimozione con lettura** e **lettura non distruttiva** dell'elemento immediatamente accessibile; test di **vuotezza**; lettura della **dimensione**; **svuotamento** della struttura).

Al fine di poter testare adeguatamente le librerie sopra descritte, si richiede di definire (esternamente alle stesse, in un opportuno file di test richiamato dal "main") un insieme di procedure che implementi le seguenti funzionalità:

(1) **scelta della struttura** (pila vs coda), della relativa **implementazione** (vettore vs lista) e del **tipo di dati** da gestire (interi, ecc.);
(2) **popolamento della struttura** precedentemente scelta con n valori del tipo opportuno generati casualmente, dove n è un parametro dato dall'utente in ingresso.

Da un opportuno menu, dovrà essere inoltre possibile operare sulla struttura scelta attraverso le funzioni di libreria di cui al punto (3). Infine, è necessario prevedere l'accesso alla funzionalità di test prevista dal docente.

Il codice sorgente prodotto dovrà seguire pedissequamente (sia nei nomi delle funzioni di libreria, sia nella strutturazione, gerarchia di classi e nei nomi delle diverse directory e file ".cpp" e ".hpp") la forma riportata nel template Exercise2.zip associato al presente esercizio.

ESERCIZIO 3
- ALBERI BINARI CON RAPPRESENTAZIONI MULTIPLE -

F. MOGAVERO

Implementare una libreria di funzioni in **Linguaggio C++** per la gestione di una struttura dati dinamica di tipo **albero binario** contenente **dati generici**, ovvero interi, float, stringhe, struct, ecc. Tale struttura dovrà essere implementata sfruttando le seguenti due **rappresentazioni concrete**: (1) **vettore dei nodi**; (2) **puntatori ai nodi**.

Le funzionalità da realizzare sono di seguito elencate:

(1) **costruzione** e **distruzione** di una struttura dati;
(2) operazioni di **assegnamento** e **confronto** tra istanze diverse della struttura dati (il confronto deve poter essere effettuato indipendentemente dalla rappresentazione concreta);
(3) operazioni specifiche della struttura dati (**interrogazione** delle proprietà di un nodo: **accesso in lettura/scrittura** al dato contenuto nei nodi, **controllo di esistenza e accesso** al figlio sinistro/destro di un nodo; **navigazione** per mezzo di iteratori);
(4) operazioni generiche sulla struttura dati (**controllo di esistenza** di un dato valore; operazioni di **applicazione di una funzione (in ampiezza/pre-ordine/ordine/post-ordine)** a tutti gli elementi: **funzioni map**; operazioni di **accumulazione di un valore (in ampiezza/pre-ordine/ordine/post-ordine)** estratto dagli elementi: **funzioni fold**; test di **vuotezza**; lettura della **dimensione**; **svuotamento** della struttura).

Al fine di poter testare adeguatamente le librerie sopra descritte, si richiede di definire (esternamente alle stesse, in un opportuno file di test richiamato dal "main") un insieme di procedure che implementi le seguenti funzionalità:

(1) **scelta dell'implementazione** (vettore vs puntatori) e del **tipo di dati** da gestire (interi, ecc.);
(2) **popolamento della struttura** precedentemente scelta con n valori del tipo opportuno generati casualmente, dove n è un parametro dato dall'utente in ingresso;
(3) **visualizzazione in ampiezza/pre-ordine/ordine/post-ordine di tutti gli elementi** (effettuata per mezzo dell'opportuna **funzione map**);
(4) **controllo di esistenza** di un dato valore;
(5) **calcolo di una delle seguenti funzioni** (effettuato per mezzo delle **funzioni fold**) e relativa visualizzazione del risultato: prodotto per gli interi minori di n, somma per i float maggiori di n, e concatenazione per le stringhe con lunghezza minore o uguale a n, dove n è un parametro dato dall'utente in ingresso;
(6) **applicazione di una delle seguenti funzioni** a tutti gli elementi: $3n$ per gli interi, n^3 per i float, e concatenazione in testa di una specifica stringa *str* data dall'utente in ingresso nel caso delle stringhe.

Da un opportuno menu, dovrà essere inoltre possibile operare sulla struttura scelta attraverso le funzioni di libreria di cui al punto (3). Infine, è necessario prevedere l'accesso alla funzionalità di test prevista dal docente.

Il codice sorgente prodotto dovrà seguire pedissequamente (sia nei nomi delle funzioni di libreria, sia nella strutturazione, gerarchia di classi e nei nomi delle diverse directory e file ".cpp" e ".hpp") la forma riportata nel template Exercise3.zip associato al presente esercizio.

ESERCIZIO 4
- ALBERI BINARI DI RICERCA -

F. MOGAVERO

Implementare una libreria di funzioni in **Linguaggio C++** per la gestione di una struttura dati dinamica di tipo **albero binario di ricerca** contenente **dati generici**, ovvero interi, float, stringhe, struct, ecc. Tale struttura dovrà essere implementata sfruttando la rappresentazione esplicita di un albero binario per mezzo di **puntatori ai nodi**.

Le funzionalità da realizzare sono di seguito elencate:

(1) **costruzione** e **distruzione** di una struttura dati;
(2) operazioni di **assegnamento** e **confronto** tra istanze diverse della struttura dati;
(3) operazioni specifiche della struttura dati (**inserimento/eliminazione** di un dato elemento; **rimozione, rimozione con lettura** e **lettura non distruttiva** dell'elemento minimo/massimo o del predecessore/successore di un dato elemento);
(4) operazioni generiche sulla struttura dati (**controllo di esistenza** di un dato valore; operazioni di **applicazione di una funzione (in ampiezza/pre-ordine/ordine/post-ordine)** a tutti gli elementi: **funzioni map**; operazioni di **accumulazione di un valore (in ampiezza/pre-ordine/ordine/post-ordine)** estratto dagli elementi: **funzioni fold**; test di **vuotezza**; lettura della **dimensione**; **svuotamento** della struttura).

Al fine di poter testare adeguatamente le librerie sopra descritte, si richiede di definire (esternamente alle stesse, in un opportuno file di test richiamato dal "main") un insieme di procedure che implementi le seguenti funzionalità:

(1) **scelta del tipo di dati** da gestire (interi, ecc.);
(2) **popolamento della struttura** precedentemente scelta con n valori del tipo opportuno generati casualmente, dove n è un parametro dato dall'utente in ingresso;
(3) **visualizzazione in ampiezza/pre-ordine/ordine/post-ordine di tutti gli elementi** (effettuata per mezzo dell'opportuna **funzione map**);
(4) **controllo di esistenza** di un dato valore;
(5) **calcolo di una delle seguenti funzioni** (effettuato per mezzo delle **funzioni fold**) e relativa visualizzazione del risultato: prodotto per gli interi minori di n, somma per i float maggiori di n, e concatenazione per le stringhe con lunghezza minore o uguale a n, dove n è un parametro dato dall'utente in ingresso.

Da un opportuno menu, dovrà essere inoltre possibile operare sulla struttura scelta attraverso le funzioni di libreria di cui al punto (3). Infine, è necessario prevedere l'accesso alla funzionalità di test prevista dal docente.

Il codice sorgente prodotto dovrà seguire pedissequamente (sia nei nomi delle funzioni di libreria, sia nella strutturazione, gerarchia di classi e nei nomi delle diverse directory e file ".cpp" e ".hpp") la forma riportata nel template Exercise4.zip associato al presente esercizio.