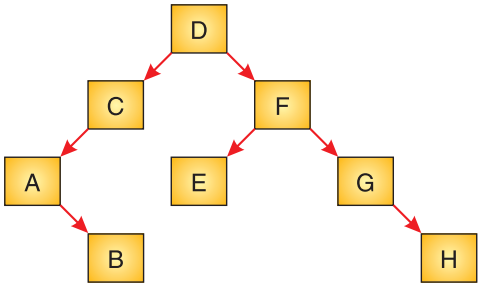
4. La struttura dati ad albero binario

L'**albero** è una struttura dati nella quale le informazioni sono organizzate in modo gerarchico, dall'alto verso il basso.
Gli elementi di un albero si chiamano **nodi**, mentre con **ramo** si indica un cammino composto da più nodi. Un nodo posto al termine di un ramo è chiamato **foglia,** mentre il nodo dal quale partono i rami è detto **radice** (*root*).



Il percorso che consente di attraversare l'albero, per visualizzare i valori contenuti nei nodi, si chiama **visita** o **attraversamento** dell'albero.
L'elaborazione che, nell'ordine, visita prima il sottoalbero di sinistra, poi la radice e infine il sottoalbero di destra si chiama attraversamento in **inordine** o ordine simmetrico.
Altre due modalità di attraversamento dell'albero sono quelle in **preordine** o ordine anticipato (prima la radice e poi i sottoalberi) ed in **postordine** o ordine posticipato (prima i sottoalberi e poi la radice). L'attraversamento dei sottoalberi nei tre casi avviene con la stessa modalità dell'attraversamento dell'albero.
Un caso particolare di albero è costituito dall'**albero binario**, nel quale da ogni nodo partono al massimo due rami distinti e chiamati sottoalbero sinistro e sottoalbero destro.

PROGETTO

Costruire il programma per gestire le operazioni su un albero binario. Ogni nodo dell'albero contiene un codice e una descrizione; il programma deve consentire di creare l'albero, aggiungendo nuovi nodi e di visualizzare i dati contenuti nella struttura in ordine di descrizione.

DESCRIZIONE DEL PROBLEMA E TRACCIA PER LA SOLUZIONE

I nodi dell'albero contengono una parte con le informazioni (campo *info*) e i puntatori al sottoalbero sinistro e destro. Le informazioni sono strutture definite dal programmatore, costituite da un campo *codice* e da un campo *descrizione*.
Pertanto la dichiarazione delle strutture in linguaggio C++ è la seguente:

```
// struttura della parte informativa
struct Dati {
   int codice;
   string descrizione;
};
```

```
// struttura per i nodi dell'albero
struct Nodo {
   Dati info;
   Nodo* sx;
   Nodo* dx;
};

Nodo* radice;                  // puntatore alla radice
```

Il programma usa anche un'enumerazione per rappresentare il risultati del confronto con gli altri elementi dell'albero quando si inserisce un nuovo nodo. Questo può servire a rendere il codice più leggibile e autodocumentato.

```
enum Confronti {Precede=1, Segue, Uguali};
```

Le funzioni per la gestione dell'albero binario si possono classificare in quattro categorie:

1. funzioni che svolgono compiti di utilità;

2. funzioni che realizzano operazioni sui dati;

3. funzioni che realizzano operazioni sulla struttura;

4. funzioni che interfacciano il programma.

Le funzioni di *tipo 1* non intervengono sui dati ma servono a compiti generici. Non devono chiamare funzioni degli altri tipi.
Le funzioni di *tipo 2* conoscono la struttura dei singoli dati e permettono di stampare, leggere e modificare il contenuto della parte di nodo che contiene le informazioni (il campo *info*). Possono chiamare solo funzioni di tipo 1.
Le funzioni di *tipo 3* conoscono i dettagli della struttura ad albero impiegata, e permettono di aggiungere i nodi. Non devono invece accedere direttamente alla parte informativa dei nodi, ma solo usando le funzioni di tipo 2.
Questa suddivisione permette di individuare con esattezza i punti da modificare qualora si debba cambiare la struttura delle informazioni o la struttura di supporto (l'albero). Se si volesse per esempio aggiungere un campo informativo si dovrebbero solo modificare le funzioni di tipo 1, mentre se si volesse usare una struttura ad albero realizzata in modo differente si dovrebbe intervenire solo sulle funzioni di tipo 2.
Infine, le funzioni di *tipo 4* vengono chiamate dal programma principale e contengono le chiamate alle funzioni di tipo 1 e 2 che realizzano le funzionalità richieste.
Esaminiamo ora in dettaglio le diverse funzioni secondo la classificazione proposta.

- **Funzioni di tipo 1:** non devono contenere riferimenti ai dati o alla struttura, né a funzioni di altro tipo.

La funzione *Operazione* scrive la descrizione dell'operazione che si vuole eseguire. È una funzione comoda per dare all'applicazione un aspetto omogeneo in tutte le circostanze senza dover riscrivere codice simile in tutte le funzioni.

```
void Operazione(string testata)
{
   cout << endl
        << testata
        << endl;
}
```

- **Funzioni di tipo 2:** possono fare riferimento alla struttura interna dei dati, ma non alla struttura usata per ospitarli (albero).

La funzione *StampaDato* viene richiamata nelle operazioni di stampa.

```
void StampaDato(Dati d)
{
   cout << d.codice << ": " << d.descrizione << endl;
}
```

La funzione *ConfrontaDati* stabilisce il criterio di ordinamento seguito entro la struttura.

```
Confronti ConfrontaDati(Dati dato1, Dati dato2)
{
   Confronti c;
   if (dato1.descrizione == dato2.descrizione)
      c = Uguali;
   else
      if (dato1.descrizione < dato2.descrizione)
         c = Precede;
      else
         c = Segue;

   return c;
}
```

La funzione *LeggiDato* acquisisce da tastiera un codice e una descrizione e restituisce un valore per il controllo di fine inserimento.

```
int LeggiDato(Dati& d)
{
   int cod;
   cout << "Codice (0=fine): ";
   cin >> cod;
   if (cod != 0) {
      d.codice = cod;
      cout << "Descrizione: ";
      cin >> d.descrizione;
   }

   return cod;
}
```

- **Funzioni di tipo 3:** possono fare riferimento alla struttura dell'albero e non alla struttura interna dei dati.

La funzione *Inizializza* assegna il valore NULL al puntatore *radice*.

```
void Inizializza()
{
   radice = NULL;
}
```

La funzione *GeneraNodo* viene richiamata dalla funzione di creazione dell'albero e consente l'inserimento dei nodi.

```
Nodo* GeneraNodo()
{
    Nodo* p = NULL;
    Dati d;
    int fine;
    fine = LeggiDato(d);
    while (fine != 0) {
       p = TrovaNodo(d, p);
       fine = LeggiDato(d);
    }

    return p;
}
```

La funzione *TrovaNodo*, richiamata dalla funzione *GeneraNodo*, è la funzione più importante del programma. Esegue tutti i movimenti sulla struttura e restituisce un puntatore al nodo che contiene il dato ricevuto nel primo parametro.
La ricerca inizia dal nodo puntato dal puntatore *p* passato come secondo parametro ed è ricorsiva.

```
Nodo* TrovaNodo(Dati dato, Nodo* p)
{
    if (p == NULL) {
       p = new Nodo;
       p->info = dato;
       p->sx = NULL;
       p->dx = NULL;
    }
    else {                                  // p diverso da NULL
        switch (ConfrontaDati(dato, p->info)) {
       case Precede:
          p->sx = TrovaNodo(dato, p->sx);
          break;
       case Segue:
          p->dx = TrovaNodo(dato, p->dx);
          break;
       case Uguali:
          cout << 'dato duplicato, non inserito" << endl;
          break;
       }
    }

    return p;
}
```

L'operazione di stampa è ricorsiva e alla prima chiamata deve ricevere il puntatore *radice*. Essa realizza l'attraversamento dell'albero in **ordine simmetrico** (prima il sottoalbero di sinistra, poi la radice e infine il sottoalbero di destra).

```
void StampaRicorsiva(Nodo* p)
{
   if (p != NULL) {
      StampaRicorsiva(p->sx);
      StampaDato(p->info);
      StampaRicorsiva(p->dx);
   }
}
```

- **Funzioni di tipo 4:** non possono fare riferimento né alla struttura ad albero né alla struttura interna dei dati. Possono però usare le funzioni degli altri tipi.

Queste sono le funzioni che attivano le operazioni implementate per la gestione dell'albero binario: inserimento e stampa.

```
void CreaAlbero()
{
   Operazione("Inserimento dei dati");
   radice = GeneraNodo();
}

void StampaAlbero()
{
   Operazione("Stampa lista odinata");
   StampaRicorsiva(radice);
   cout << endl;
}

int PresentaMenu()
{
   int s;

   do {
      cout << "-------------------------" << endl;
      cout << "1. Creazione di un albero" << endl;
      cout << "2. Stampa ordinata" << endl;
      cout << "3. Fine lavoro" << endl;
      cout << "-------------------------" << endl;
      cout << "Inserisci la tua scelta ";
      cin >> s;
      cout << "-------------------------" << endl;
   } while (s<1 || s>3);

   return s;
}
```

Le seguenti righe di codice riassumono i prototipi delle funzioni utilizzate nel programma:

```
// prototipi delle funzioni
void Inizializza();
void Operazione(string testata);
Nodo* GeneraNodo();
Nodo* TrovaNodo(Dati dato, Nodo* p);
Confronti ConfrontaDati(Dati dato1, Dati dato2);
void CreaAlbero();
int LeggiDato(Dati& d);
void StampaDato(Dati d);
void StampaRicorsiva(Nodo* p);
void StampaAlbero();
int PresentaMenu();
```

La funzione *main* del programma contiene infine il menu di scelte per l'utente, che corrispondono alla chiamata delle funzioni viste precedentemente.

```
// funzione principale
int main()
{
   int scelta;                // scelta dell'utente
   Inizializza();
   do {
      scelta = PresentaMenu();
      switch(scelta) {
      case 1:
         CreaAlbero();        // inserimento di nuovi dati
         break;
      case 2:
         StampaAlbero();      // stampa ordinata del contenuto
         break;               // dell'albero
      }
   } while (scelta != 3);

   return 0;
}
```

Il programma presentato mantiene l'albero in ordine alfabetico di descrizione. Per costruire l'albero ordinato secondo il campo *codice* dei nodi, occorre semplicemente modificare la funzione *ConfrontaDati*, controllando l'ordine dei codici.
La traccia fornita può essere ampliata con l'implementazione di altre funzioni importanti per la gestione dell'albero:
- funzione per la stampa del contenuto dell'albero secondo un ordine diverso dall'ordine simmetrico utilizzato nella traccia;
- funzione di ricerca di un nodo, fornito il codice o la descrizione;
- funzione di cancellazione dell'albero.

Quest'ultima funzione deve essere comunque eseguita alla chiusura del programma per rilasciare la memoria puntata dai puntatori.

Utilizzando come riferimento la traccia precedente, si possono risolvere i seguenti problemi:
- Costruire l'albero binario, con le relative funzioni di gestione, che consenta di mantenere ordinati i codici degli articoli di un magazzino.
- Scrivere la funzione che consenta di ottenere l'attraversamento di un albero binario in ordine anticipato (prima la radice e poi i sottalberi).