ساختمان داده ها

انواع داده ای مجرد
Abstract Data Types

مدرس: غیاثی‌شیرازی
دانشگاه فردوسی مشهد

Abstract Data Types (ADT)

- رفتار مورد انتظار از یک ساختمان داده معمولا با استفاده از انواع داده ای مجرد مشخص می شود:
  - نوع داده ای که ذخیره می شود اهمیتی ندارد.
  - گاهی داده های ذخیره شده باید برخی عملیات را حمایت کنند.
  - عملیاتی که بر روی ADT قابل انجام است مشخص می شود.
  - ممکن است مرتبه زمانی و یا میزان مصرف حافظه توسط هر یک از عملیات ذکر شود.
  - برخی گزاره های منطقی که درستی آنها همواره توسط نوع داده ای مجرد حفظ می شود، ذکر می شوند.
  - نحوه پیاده سازی این عملیات ذکر **نمی شود.**

Data Object

- به داده های ذخیره شده در یک ADT، اشیاء داده ای (Data Object) گفته می شود.

رابطه ساختمان های داده با ADT

- همانطور که گفته شد:

  یک ساختمان داده یک روش ویژه ذخیره و سازمان دهی داده‌ها در کامپیوتر است به نحوی که بتوان به صورت کارا عملیات مورد نظر را بر روی داده‌ها انجام داد.
- یک ساختمان داده برای کاربران آن به صورت یک ADT معرفی می شود.
- ممکن است چندین ساختمان داده یک ADT را تحقق بخشند.
- ADT راجع به «چه» صحبت می کند و ساختمان داده راجع به «چگونه».

مثال: نوع داده ای مجرد لیست خطی (LinearList)

- نوع داده ای که ذخیره می شود اهمیتی ندارد.
  - یک لیست خطی می تواند هر نوع داده ای را ذخیره کند.
- گاهی داده های ذخیره شده باید برخی عملیات را حمایت کنند.
  - در صورتی که بخواهیم لیست خطی را نمایش دهیم، لازم است هر عنصر قابل تبدیل شدن به یک رشته متنی باشد.

مثال: نوع داده ای مجرد لیست خطی (LinearList)

- عملیاتی که بر روی ADT قابل انجام است مشخص می شود.
  - می توان محلی را در لیست مشخص کرد و عنصری را در آن محل درج کرد.
  - می توان عنصری با اندیس مشخص را از لیست حذف کرد.
  - می توان طول لیست را به دست آورد.
  - می توان تشخیص داد که آیا لیست خالی است یا نه.
  - می توان عنصری را که در محل مشخصی است دریافت کرد.

مثال: نوع داده ای مجرد لیست خطی (LinearList)

- ممکن است مرتبه زمانی و یا میزان مصرف حافظه توسط هر یک از عملیات ذکر شود.
  - زمان محاسبه طول لیست از مرتبه $O(1)$ است.
  - زمان تشخیص دادن خالی بودن لیست از مرتبه $O(1)$ است.
  - زمان دریافت عنصری از لیست از مرتبه $O(1)$ است.

list<int> size() O(n)

مثال: نوع داده ی مجرد لیست خطی (LinearList)

- برخی گزاره های منطقی که درستی آنها همواره توسط نوع داده ای مجرد حفظ می شود، ذکر می شوند.

– not IsEmpty(Insert(L,x))

- اگر در لیست عنصری را درج کنیم و بعد بپرسیم آیا خالی است یانه، پاسخ منفی است.

تعریف انواع داده ای مجرد
در زبان های برنامه سازی شیء گرا

- همانطور که گفته شد، در انواع داده ای مجرد تنها عملیات مورد نظر بر روی ADT تعریف می شود و نحوه پیاده سازی این عملیات ذکر نمی شود.
- در زبان برنامه نویسی C++ نوع های داده ای مجرد به صورت class طراحی می شود. البته تمام متودهای این کلاس مجرد خواهند بود. =0
- در زبان برنامه نویسی جاوا نوع های داده ای مجرد معمولا به صورت interface طراحی می شود.

تعریف انواع داده ای مجرد، مستقل از نوع داده ذخیره شده

- روش اول: نوع داده ذخیره شده را معمولا به صورت یک کلاس نامشخص T در نظر میگیریم و کلاس را در C++ به صورت template (در جاوا Generic) طراحی می کنیم.
- روش دوم: یک interface برای اشیاء داده ای تعریف می کنیم که هر داده ذخیره شده باید آن را پیاده سازی کرده باشد. هر نوع داده ای که interface مشخص شده را پیاده سازی کرده باشد می تواند در ADT ذخیره شود.
  - روش دوم کاربران ADT را مجبور می کند که از کلاس خاصی ارث ببرند که در حالت کلی مطلوب نیست.

طراحی نوع داده ای مجرد LinearList به روش template به زبان C++

```
template <class T>
class LinearList
{
   …
};
```

insert (int position, const T& data)

باعث می‌شود کپی انجام نشود.

طراحی نوع داده ای مجرد LinearList به صورت Generic به زبان جاوا

```
interface LinearList<T>
{
    …
}
```

طراحی نوع داده ای مجرد LinearList با استفاده از interface خاص در زبان C++

```
class LinearListData
{
};

class LinearList
{
   void Insert(int index, LinearListData* theElement);
   …
};
```

طراحی ساختمان داده LinearList به روش استفاده از interface خاص در زبان جاوا

```
interface LinearListData
{
}

interface LinearList
{
    void Insert(int index, LinearListData theElement);
    …
}
```

طراحی ساختمان داده LinearList به روش ترکیبی در زبان جاوا

```
interface LinearListData
{
}

interface LinearList<T extends LinearListData>
{
   void Insert(int index, T theElement);
   ... T  Get (int index)
}
```

- نکته: با وجود اینکه LinearListData یک interface است، اما در تعریف انواع Generic باید از عبارت extends استفاده کرد و نه implements.

داده های ذخیره شده از عملیات خاصی پشتیبانی می کنند

- گاهی لازم است داده هایی که در یک ADT ذخیره می شوند، عملیات خاصی را حمایت کنند:
  - مثلا قابل مقایسه باشند.
  - مثلا بتوان مقدار آنها را به صورت یک رشته متنی نمایش داد.
  - مثلا یک interface خاص را پیاده سازی کرده باشند.

مقایسه پذیری اشیاء داده در C++

1. یکی از عملگرهای مقایسه (مانند =>) برای داده های ذخیره شده تعریف شود.
2. تابعی مشخص شود که می تواند دو نمونه از داده های ذخیره شده را مقایسه نماید.
3. کلاسی مشخص شود که دارای متودی برای مقایسه است. یک پیاده سازی پیش فرض از این کلاس می تواند این باشد که اشیاء را با عملگر مقایسه با یکدیگر مقایسه کند. در صورتی که عملگر مقایسه برای شیء داده قابل تعریف نباشد، می توان یک پیاده سازی خاص از این کلاس داشت.

روش اول: پیاده سازی عملگر <

```
class LinearListData
{
public:
	virtual bool operator<(const LinearListData& data) = 0;
};
```

مقایسه بین *this و data

- با این روش تمام اشیاء داده (Object Data) باید عملگر < را پیاده سازی نمایند.

روش دوم: مشخص کردن تابعی برای مقایسه

- در این مثال به سازنده شیء LinearList یک تابع cmp که اشیاء از نوع LinearListData را مقایسه می کند فرستاده می شود.

```
class LinearList
{
public:
LinearList(bool(*cmp)(LinearListData&, LinearListData&)){
        mCmpFunc = cmp;
}
private:
bool(*mCmpFunc)(LinearListData& d1, LinearListData& d2);
};
```

یک اشاره گر

یک صفت از نوع اشاره گر به تابع

روش سوم: استفاده از کلاس سومی برای عمل مقایسه

- مثال: کلاس less در کتابخانه STL در زبان C++ چنین تعریف شده است:

```
template <class T> struct less {
  bool operator() (const T& x, const T& y)const
      {
          return x < y;
      }
};
```

این کلاس به طور پیش فرض دو شیء T را با استفاده از عملگر > مقایسه می کند. اما در صورت لزوم می توان کلاس جدیدی نوشت که اشیاء داده را به نحوی دیگر مقایسه کند.

تضمین مقایسه پذیری اشیاء داده در جاوا

- در جاوا برای تضمین اینکه اشیاء ساخته شده از یک کلاس با نوع T قابل مقایسه هستند، آن کلاس باید رابط Comparable<T> را پیاده سازی کند.
- رابط Comparable<T> دارای یک متود با نام compareTo است که شیئی از نوع T را دریافت می کند و نتیجه مقایسه this با آن شیء را به صورت int برمیگرداند.
  - عدد منفی == کوچک تر
  - صفر == مساوی
  - عدد مثبت == بزرگ تر

تضمین مقایسه پذیری اشیاء داده در جاوا

- روش دیگر برای مقایسه اشیاء در جاوا پیاده سازی رابط Comparator<T> است که شیء مستقلی است که عمل مقایسه داده های از نوع T را انجام می دهد.
- رابط Comparator<T > دارای یک متود با نام compare است که دو شیئ از نوع T را دریافت می کند و نتیجه مقایسه آنها را به صورت int برمیگرداند.
  - عدد منفی == کوچک تر
  - صفر == مساوی
  - عدد مثبت == بزرگ تر

تعیین عملیات قابل انجام بر روی ADT

- از آنجا که عملیات قابل انجام بر روی ADT باید به صورت مجرد تعریف شوند، از interface برای تعیین این عملیات استفاده می کنیم.
- در C++ از کلاس هایی که تمام متودهای آنها توابع مجرد هستند استفاده می کنیم.

طراحی نوع داده مجرد LinearList در C++

```
template <class T>
class LinearList
{
   virtual bool isEmpty() const = 0;
   virtual int length() const = 0;
   virtual T* retrieve(int index) const = 0;
   virtual int indexOf(const T* theElement) const = 0;
   virtual T* delete(int index) = 0;
   virtual void insert(int index, T* theElement) = 0;
};
```

طراحی نوع داده مجرد LinearList در جاوا

```
interface LinearList<T>
{
    boolean isEmpty();
    int length();
    T retrieve(int index);
    int indexOf(T theElement);
    T delete(int index);
    void insert(int index, T theElement);
}
```

خلاصه

- انواع داده مجرد (Abstract Data Types) معرفی شدند.
  - برای تجرید داده های ذخیره شده از template و یا رابطی که داده های ذخیره شده از آن ارث می برند استفاده می کنیم.
  - عملیاتی که بر روی ADT قابل انجام است را بطور دقیق و بدون پیاده سازی تعریف می کنیم.
- اشیاء ذخیره شده در ADT را Data Object می نامیم.
- نحوه مقایسه اشیاء داده در زبان‌های C++ و جاوا تشریح شد.
- نوع داده ای مجرد LinearList به عنوان مثال تعریف شد.