15CSE201 : Data Structures and Algorithms

Lecture 3 : Abstract Data Types(ADT's)

Ritwik M

Based on the reference materials by Prof. Goodrich, OCW METU and Dr. Vidhya Balasubramanian

Abstract Data Types

- Two important aspects of a data structure
  - Interface: describes what a data structure does
    - Also known as Abstract Data Type
    - Provides list of supported operations
    - Includes specification of types of arguments accepted and the return values
  - Implementation: describes how the data structure does it
    - Internal representation of data structure
    - Algorithms to implement the functionalities

Definition

- Abstract Data Type
  - Is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics [2]
  - e.g a stack ADT is defined by its operations, push, pop and top.
- ADT defines
  - the set of operations supported by a data structure and
  - the semantics, or meaning, of those operations

Implementation Strategy

- Usually implemented as modules
  - the module's interface declares procedures that correspond to the ADT operations
- Can be multiple implementations of a single ADT
- Stack ADT Interface
  - push(x)
  - pop(x)
  - top(x)

ADTs in Detail

- An abstract data type is defined as a mathematical model of the data objects that make up a data type as well as the functions that operate on these objects
- The definition can vary based on the type of programming language
  - Imperative Definition
  - Object Oriented Definition

Imperative Definition

- An abstract data structure is considered an entity that may be in different states at different times
  - Operations may change the state of the ADT
  - Order of operations important
- Implementation
- Use concept of abstract variables (simplest non trivial ADT), that admits two operations
  - store(V,x), x is any value
  - fetch(V)
    - Returns value that is most recently stored.

Imperative ADT

- Instance Creation
  - One may need to create a new instance of a stack
  - create() operation is used for this purpose
- Example: Abstract Stack defines state of stack S
  - push(S,x)
  - pop(S)
  - create() - creates a new stack different from other stacks

Object Oriented Definition

- Defines the ADT as a class containing
  - Variables
  - Functions
- E.g Stack ADT in Object Oriented Paradigm

Advantages of Abstract Data Typing

- Encapsulation
  - Guarantees that the ADT has certain properties and abilities
  - User does not need to know the implementation details to use the ADT
- Localization of Change
  - Change to ADT implementation does not impact the code that uses the ADT
  - The implementation must still comply with the ADT definition, hence the interface will not change

Advantages Contd

- Flexibility
  - Can use different implementations of an ADT in a code
    - All have same properties and abilities
  - The efficiency of different implementations maybe different
- Can use the most suitable implementation

Templates – A C++ Concept

- Allows data structure implementations that can be applied to different data types
- Types
  - Function Templates

```
template <typename Type>
Type max(Type a, Type b) {
return a > b ? a : b;
}
```

  - Class Templates
    - provides a specification for generating classes based on parameters

Example Class Template

- Syntax

  template <class identifier> function_declaration;
  template <typename identifier> function_declaration;

- Example

```
template <class T>  //more template parameters can be specified within the angle brackets
T GetMax (T a, T b) {
  T result; //type T is used within the template function to declare new objects of that type
  result = (a>b)? a : b;
  return (result);
}
```

```
k=GetMax<int>(i,j);
n=GetMax<long>(l,m);
```

Templates Contd.

- Class templates can also be written so that a class can have members that use template parameters as types
  - Functions defined outside class template
- Example

```
template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};
```

```
template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}
```

Standard Template Libraries

- STLs are libraries that implement the common data structures
  - Available in different languages
- Available STLs in C++
  - C++ STL
  - Open Data Structures

References

1. Pat Morin, “Open Data Structures in C++”, Edition 0.1
2. “Data Structures and Algorithms in C++”