Abstraction

- The concept of abstraction is fundamental in programming

- Nearly all programming languages support process abstraction with subprograms

- Nearly all programming languages designed since 1980 have supported data abstraction with some kind of module

Encapsulation

- *Original motivation*:
  - Large programs have two special needs:
    1. Some means of organization, other than simply division into subprograms
    2. Some means of partial compilation (compilation units that are smaller than the whole program)

- *Obvious solution*: a grouping of subprograms that are logically related into a unit that can be separately compiled
  - These are called *encapsulations*

Examples of Encapsulation Mechanisms

1. Nested subprograms in some ALGOL-like languages (e.g., Pascal)

2. FORTRAN 77 and C - Files containing one or more subprograms can be independently compiled

3. FORTRAN 90 and Ada - separately compilable modules

Def: An *abstract data type* is a user-defined data type that satisfies the following two conditions:

1. **The representation of and operations on objects of the type are defined in a single syntactic unit; also, other units can create objects of the type.**

2. **The representation of objects of the type is hidden from the program units that use these objects, so the only operations possible are those provided in the type's definition.**

Advantage of Restriction 1:

- Same as those for encapsulation: program organization, modifiability (everything associated with a data structure is together), and separate compilation

Advantage of Restriction 2:

- *Reliability*--by hiding the data representations, user code cannot directly access objects of the type. User code cannot depend on the representation, allowing the representation to be changed without affecting user code.

Built-in types are abstract data types

e.g. `int` type in C

- The representation is hidden
- Operations are all built-in
- User programs can define objects of `int` type

- User-defined abstract data types must have the same characteristics as built-in abstract data types

Language Requirements for Data Abstraction:

1. A syntactic unit in which to encapsulate the type definition.

2. A method of making type names and subprogram headers visible to clients, while hiding actual definitions.

3. Some primitive operations must be built into the language processor (usually just assignment and comparisons for equality and inequality)
   - Some operations are commonly needed, but must be defined by the type designer
     - e.g., iterators, constructors, destructors

Language Design Issues:

1. Encapsulate a single type, or something more?
2. What types can be abstract?
3. Can abstract types be parameterized?
4. What access controls are provided?

Language Examples:

1. Simula 67

- Provided encapsulation, but no information hiding

2. Ada

- The encapsulation construct is the package

- Packages usually have two parts:

1. **Specification package (the interface)**
2. **Body package (implementation of the entities named in the specification**

- Any type can be exported

- Information Hiding

- Hidden types are named in the spec package in, as in:

```
type NODE_TYPE is private;
```

- Representation of an exported hidden type is specified in a special invisible (to clients) part of the spec package (the `private` clause), as in:

```
package … is
    type NODE_TYPE is private;
    …
    type NODE_TYPE is
      record
      …
      end record;
    …
```

- A spec package can also define unhidden types simply by providing the representation outside a private clause

- *The reasons for the two-part type definition are:*

1. The compiler must be able to see the representation after seeing only the spec package (the compiler can see the private clause)
2. Clients must see the type name, but not the representation (clients cannot see the private clause)

- *Private types* have built-in operations for assignment and comparison with = and /=

- *Limited private types* have no built-in operations

---> SHOW specification and body packages (pp. 422-423) and the procedure that uses them (p. 423)

Evaluation of Ada Abstract Data Types

1. Lack of restriction to pointers is better
- Cost is recompilation of clients when the representation is changed

2. Cannot import specific entities from other packages

4. C++

- **Based on C `struct` type and Simula 67 classes**
- **The class is the encapsulation device**
- **All of the class instances of a class share a single copy of the member functions**
- **Each instance of a class has its own copy of the class data members**
- **Instances can be static, stack dynamic, or heap dynamic**

- ***Information Hiding:***
  - ***Private clause* for hidden entities**
  - ***Public clause* for interface entities**
  - ***Protected clause* - for inheritance (see Ch. 11)**

- ***Constructors:***
  - **Functions to initialize the data members of instances (they *DO NOT* create the objects)**
  - **May also allocate storage if part of the object is heap-dynamic**
  - **Can include parameters to provide parameterization of the objects**
  - **Implicitly called when an instance is created**
  - **Can be explicitly called**
  - **Name is the same as the class name**

- Destructors

- Functions to cleanup after an instance is destroyed; usually just to reclaim heap storage

- Implicitly called when the object's lifetime ends

- Can be explicitly called

- Name is the class name, preceded by a tilda (~)

---> SHOW class definition for `stack` (p. 425-426 and the example program that uses it (p. 426)

***- Friend functions or classes* - to provide access to private members to some unrelated units or functions (*NECESSARY* in C++)**

Evaluation of C++ Support for Abstract Data Types

- Classes are similar to Ada packages for providing abstract data type

- Difference: packages are encapsulations, whereas classes *are* types

A Related Language: Java

- Similar to C++, except:
  - All user-defined types are classes
  - All objects are allocated from the heap and accessed through reference variables
  - Individual entities in classes have access control modifiers (private or public), rather than clauses
  - Java has a second scoping mechanism, package scope, which can be used in place of friends
    - All entities in all classes in a package that do not have access control modifiers are visible throughout the package

--> SHOW Java class definition for stacks (p. 428) and the class that uses it (p. 429

Parameterized Abstract Data Types

1. Ada Generic Packages

- Make the stack type more flexible by making the element type and the size of the stack generic

---> SHOW `GENERIC_STACK` package and two instantiations (p. 430)

2. C++ Templated Classes

- Classes can be somewhat generic by writing parameterized constructor functions

e.g.

```
stack (int size) {
  stk_ptr = new int [size];
  max_len = size - 1;
  top = -1;
 }
stack (100) stk;
```

- **The stack element type can be parameterized by making the class a templated class**

---> SHOW the templated class `stack` (p. 431)

- **Java does not support generic abstract data types**