A Simple ADT Example

CMPT 115/117 lecture slides

Notes written by Mark Eramian, Ian McQuillan, Michael Horsch, Lingling Jin, and Dmytro Dyachuk

Objectives

After this topic, students are expected to

1. Give a simple example of an ADT for simple record types
2. Make use of a new syntax for references to records, allowing simultaneous dereference and field selection.

The Data Structure

Consider a record type defined as follows:

```
BookRecord
  refToChar title
  refToChar author
  Integer nPages
  Integer yearPublished
end BookRecord
```

The Operations

1. Store information about a book in a record
2. Deallocate memory used by book record.
3. Display the book information on the console.
4. Copy a book, as a new edition published in a new year

The Interface (1)

Algorithm createBookRecord(t,a,p,y)
Allocates and stores information about a book in a record

Pre: t :: refToChar, the title of the book
a :: refToChar, the author of the book
p :: Integer, the number of pages
y :: Integer, the year the book was first published

Post: allocates memory to store the information
Return: a reference to a BookRecord

The Interface (2)

Algorithm destroyBookRecord(book)
Deallocates memory used by book record

Pre: book :: refToBookRecord
Post: memory allocated to book is deallocated
Return: nothing

Algorithm displayBookRecord(book)
Displays the book information on the console

Pre: book :: refToBookRecord
Post: Book Record is displayed to console
Return: nothing

The Interface (3)

Algorithm newEditionOf(book, y)
Creates a new record, copying from book, with a new year

Pre: book :: refToBookRecord
y :: Integer, the year the new edition was published

Post: allocates memory to store the information
Return: a reference to a BookRecord

The Pseudocode Implementation (1)

```
Algorithm createBookRecord(t,a,p,y)
  Allocates and stores information about a book in a record

  Pre: t :: refToChar, the title of the book
       a :: refToChar, the author of the book
       p :: Integer, the number of pages
       y :: Integer, the year the book was first published

  Post: allocates memory to store the information
  Return: a reference to a BookRecord

  refToBookRecord br ← allocate new BookRecord

  (*br).title ← copyString(t)
  (*br).author ← copyString(a)
  (*br).nPages ← p
  (*br).yearPublished ← y

  return br
```

The Pseudocode Implementation (2)

```
Algorithm destroyBookRecord(book)
  Deallocates memory used by book record

  Pre: book :: refToBookRecord
  Post: memory allocated to book is deallocated
  Return: nothing

  deallocate (*book).title
  deallocate (*book).author
  deallocate book
```

The Pseudocode Implementation (3)

```
Algorithm displayBookRecord(book)
  Displays the book information on the console

  Pre: book :: refToBookRecord
  Post: Book Record is displayed to console
  Return: nothing

  print (*book).title, " by", (*book).author
  print (*book).nPages, " pages"
  print (*book).year, "."
```

The Pseudocode Implementation (4)

```
Algorithm newEditionOf(book, y)
  Creates a new record, in a new year

  Pre: book :: refToBookRecord
       y :: Integer, the year the new edition was published

  Post: allocates memory to store the information
  Return: a reference to a BookRecord

  refToBookRecord br
  br ← createBookRecord((*book).title,
                        (*book).author,
                        (*book).nPages,
                        y)
  return br
```

New Syntax for Dereference and Field Selection

- The following pattern occurs a lot:

  $$(*pointer).field$$

  where `pointer` is a reference to a struct, with an element called `field`.
- It's ugly. And error prone.
- A cleaner shorthand for this is as follows:

  $$pointer \twoheadrightarrow field$$

- This simultaneously dereferences `pointer`, and then selects field `field`.
- In C/C++, this can also be done with syntax `pointer->field`. It means exactly `(*pointer).field`

Recap: The Pseudocode Implementation (1)

```
Algorithm createBookRecord(t,a,p,y)
  Allocates and stores information about a book in a record

  Pre: t :: refToChar, the title of the book
       a :: refToChar, the author of the book
       p :: Integer, the number of pages
       y :: Integer, the year the book was first published

  Post: allocates memory to store the information
  Return: a reference to a BookRecord

  refToBookRecord br ← allocate new BookRecord

  br ⇸ title ← copyString(t)
  br ⇸ author ← copyString(a)
  br ⇸ nPages ← p
  br ⇸ yearPublished ← y

  return br
```

Recap: The Pseudocode Implementation (2)

Algorithm destroyBookRecord(book)
Deallocates memory used by book record

Pre: book :: refToBookRecord
Post: memory allocated to book is deallocated
Return: nothing

deallocate book$\not\Rightarrow$ title
deallocate book$\not\Rightarrow$ author
deallocate book

Recap: The Pseudocode Implementation (3)

```
Algorithm displayBookRecord(book)
  Displays the book information on the console

  Pre: book :: refToBookRecord
  Post: Book Record is displayed to console
  Return: nothing

  print book⇸ title, " by", book⇸ author
  print book⇸ nPages, " pages"
  print book⇸ year, "."
```

Recap: The Pseudocode Implementation (4)

```
Algorithm newEditionOf(book, y)
  Creates a new record, in a new year

  Pre: book :: refToBookRecord
       y :: Integer, the year the new edition was published

  Post: allocates memory to store the information
  Return: a reference to a BookRecord

  refToBookRecord br
  br ← createBookRecord(book⇸* title,
                        book⇸* author,
                        book⇸* nPages,
                        y)
  return br
```

Summary

- Simple ADTs hide data inside a black box.
- Good ADT design provides useful operations that can be trusted
- ADTs help prevent data corruption.
- A new syntax for references to records helps make code cleaner.