Lecture 4: Abstract Data Types
Exercises

- Adapt the preceding specification of stacks (LIFO, Last-In First-Out) to describe queues instead (FIFO).

- Adapt the preceding specification of stacks to account for bounded stacks, of maximum size capacity.
  - Hint: *put* becomes a partial function.
value = item (remove (put (remove (put (put (remove (put (put (new, x8), x7), x6)), item (remove (put (new, x5), x4))), x2)), x1)))
Expressed differently

\[
\begin{align*}
    \text{value} &= \text{item} (\text{remove} (\text{put} (\text{remove} (\text{put} (\text{put} (\text{put} (\text{new}, x_8), x_7), x_6)), \text{item} (\text{remove} (\text{put} (\text{put} (\text{new}, x_5), x_4))), x_2), x_1)) \\
    &\quad \text{s}_1 = \text{new} \\
    &\quad \text{s}_2 = \text{put} (\text{put} (\text{s}_1, x_8), x_7), x_6) \\
    &\quad \text{s}_3 = \text{remove} (\text{s}_2) \\
    &\quad \text{s}_4 = \text{new} \\
    &\quad \text{s}_5 = \text{put} (\text{put} (\text{s}_4, x_5), x_4) \\
    &\quad \text{s}_6 = \text{remove} (\text{s}_5) \\
    &\quad y_1 = \text{item} (\text{s}_6) \\
    &\quad \text{s}_7 = \text{put} (\text{s}_3, y_1) \\
    &\quad \text{s}_8 = \text{put} (\text{s}_7, x_2) \\
    &\quad \text{s}_9 = \text{remove} (\text{s}_8) \\
    &\quad \text{s}_{10} = \text{put} (\text{s}_9, x_1) \\
    &\quad \text{s}_{11} = \text{remove} (\text{s}_{10}) \\
    &\quad \text{value} = \text{item} (\text{s}_{11})
\end{align*}
\]
Expression reduction

\[
\text{value} = \text{item} ( \text{remove} ( \text{put} ( \text{remove} ( \text{put} ( \text{put} ( \text{put} ( \text{new}, x_8), x_7), x_6) \\
, \text{item} ( \text{remove} ( \text{put} ( \text{new}, x_5), x_4) \\
, x_2) \\
, x_1) \\
)) \\
)
\]
Expression reduction

value = item (remove (put (remove (put (put (new, x8), x7), x6), item (remove (put (new, x5), x4), x2), x1), x1), x1), Stack 1

Chair of Software Engineering
value = item (remove (put (remove (put (put (new, x8), x7), x6)
          , item (remove (put (put (new, x5), x4)
                  , x2)
          , x1)
      )
  )
)

Stack 1

Chair of Software Engineering

Programming in the large - Lecture 4
value = item (remove (put (remove (put (remove (put (put (new, x8), x7), x6)

, item (remove (put (put (new, x5), x4)

), x2)

, x1)

)
Expression reduction

value = item (remove (put (remove (put (put (put (put (new, x8), x7), x6)
                        , item (remove (put (put (new, x5), x4)
                                      , x2)
                               , x1)
                            )
                      )
                 )
            )
         )
Expression reduction

value = item (remove (put (remove (put (put (remove (put (put (new, x8), x7), x6)
                    remove (item (remove (put (put (new, x5), x4)
                                           remove (item (x2)
                                                  remove (item (x1)
                                                      remove (item)
                                                          remove (item)))))
                                                              put (x1), item)))))
                                                          put (x2), item)))))
                                                          put (x3), item)))))
                                                          put (x4), item)))))
                                                          put (x5), item)))))
                                                          put (x6), item)))))
                                                          put (x7), item)))))
                                                          put (x8), item)))))

Stack 1  Stack 2
Expression reduction

\[
\text{value} = \text{item} \left( \text{remove} \left( \text{put} \left( \text{remove} \left( \text{put} \left( \text{put} \left( \text{remove} \left( \text{put} \left( \text{new}, x8 \right), x7 \right), x6 \right) \right) \right), x5 \right), x4 \right) \right), x2 \right), x1 \right) \right)
\]
Expression reduction

value = item (remove (put (remove (put (put (put (new, x8), x7), x6)
remove (put (put (new, x5), x4)
, x2)
, x1)
, x5))
, x8)
, x4)
Stack 1
Stack 2
Expression reduction

```
value = item (  
    remove (  
      put (  
        remove (  
          put (  
            put (  
              remove (  
                put (  
                  put (  
                    new , x8 , x7  
                  )  
                )  
              )  
            )  
          )  
        )  
      )  
    )  
)  
```

Stack 1

Stack 2
Expression reduction

value = item (remove (put (remove (put (put (put (new, x8), x7), x6))
  , item (remove (put (put (new, x5), x4))
    , x2)
  , x1)
  )
)
value = item (remove (put (remove (put (put (new, x8), x7), x6))
remove (put (put (new, x5), x4))
, item (remove (put (put (new, x5), x4))
, x2)
, x1)
)
)}
Expression reduction

value = item (remove (put (remove (put (remove (put (remove (put (remove (put (new, x8), x7), x6) item (remove (put (new, x5), x4) , x2) , x1) , x2) , x8) , x7) , x5) , x2) , x1))
Expression reduction

value = item (remove (put (remove (put (remove (put (new, x8), x7), x6) , item (remove (put (new, x5), x4) ) , x2) ) , x1) ) , x5) , x7) , x8) Stack 1
Expression reduction

value = item (
  remove (put (remove (put (put (new, x8), x7), x6)
    , item (remove (put (put (new, x5), x4)
        , x2)
    , x1))
  , x1))
Expression reduction

```
value = item (remove (put (remove (put (put (new, x8), x7), x6)
    , item (remove (put (new, x5), x4)
    )
    
    , x2)
    , x1)
)
```

Stack 1
value = \texttt{item (}
  \texttt{remove (}
    \texttt{put (}
      \texttt{remove (}
        \texttt{put (}
          \texttt{remove (}
            \texttt{put (}
              \texttt{remove (}
                \texttt{put (}
                  \texttt{put (}
                    \texttt{put (}
                      \texttt{new, x8}, x7, x6)
                , item (}
                  \texttt{remove (}
                    \texttt{put (}
                      \texttt{new, x5}, x4)
                , x2)
              , x1)
            , x2)
          , x1)
        , x2)
      , x1)
    , x2)
  , x1)
\texttt{put (}
\texttt{item (}
\texttt{Stack 1})
Expressed differently

\[ \begin{align*}
\text{value} & = \text{item} \left( \text{remove} \left( \text{put} \left( \text{remove} \left( \text{put} \left( \text{remove} \left( \text{put} \left( \text{new} \left( x8 \right) \right), x7 \right), x6 \right), \text{item} \left( \text{remove} \left( \text{put} \left( \text{new} \left( x5 \right), x4 \right) \right) \right), x2 \right), x1 \right) \right) \right) \\
\text{s1} & = \text{new} \\
\text{s2} & = \text{put} \left( \text{put} \left( s1, x8 \right), x7 \right), x6 \right) \\
\text{s3} & = \text{remove} \left( s2 \right) \\
\text{s4} & = \text{new} \\
\text{s5} & = \text{put} \left( s4, x5 \right), x4 \right) \\
\text{s6} & = \text{remove} \left( s5 \right) \\
\text{y1} & = \text{item} \left( s6 \right) \\
\text{s7} & = \text{put} \left( s3, y1 \right) \\
\text{s8} & = \text{put} \left( s7, x2 \right) \\
\text{s9} & = \text{remove} \left( s8 \right) \\
\text{s10} & = \text{put} \left( s9, x1 \right) \\
\text{s11} & = \text{remove} \left( s10 \right) \\
\text{value} & = \text{item} \left( s11 \right)
\end{align*} \]
An operational view of the expression

```
value = item (remove (put (remove (put (put (put (put (new, x8), x7), x6), item (remove (put (put (new, x5), x4)))), x2), x1))
```
Sufficient completeness

- Three forms of functions in the specification of an ADT $T$:
  - Creators:
    \[ \text{OTHER} \rightarrow T \]  
    e.g. \textit{new}
  - Queries:
    \[ T \times \ldots \rightarrow \text{OTHER} \]  
    e.g. \textit{item, empty}
  - Commands:
    \[ T \times \ldots \rightarrow T \]  
    e.g. \textit{put, remove}

- Sufficiently complete specification: a “Query Expression” of the form:
  \[ f (\ldots) \]

where $f$ is a query, may be reduced through application of the axioms to a form not involving $T$
Stack: An Abstract Data Type

- **Types:**
  \( STACK \ [G] \)
  -- \( G \): Formal generic parameter

- **Functions (Operations):**
  \( put: \ STACK \ [G] \times G \rightarrow STACK \ [G] \)
  \( remove: \ STACK \ [G] \rightarrow STACK \ [G] \)
  \( item: \ STACK \ [G] \rightarrow G \)
  \( empty: \ STACK \ [G] \rightarrow BOOLEAN \)
  \( new: \ STACK \ [G] \)
Abstract data types provide an ideal basis for modularizing software.

- Build each module as an implementation of an ADT:
  - Implements a set of objects with same interface
  - Interface is defined by a set of operations (the ADT’s functions) constrained by abstract properties (its axioms and preconditions).

- The module consists of:
  - A representation for the ADT
  - An implementation for each of its operations
  - Possibly, auxiliary operations
Implementing an ADT

- **Three components:**

  (E1) The ADT’s specification: functions, axioms, preconditions.
  (Example: stacks.)

  (E2) Some representation choice.
  (Example: <representation, count>.)

  (E3) A set of subprograms (routines) and attributes, each implementing one of the functions of the ADT specification (E1) in terms of the chosen representation (E2).
  (Example: routines *put*, *remove*, *item*, *empty*, *new*.)
A choice of stack representation

"Push" operation:

\[
\begin{align*}
\text{count} & := \text{count} + 1 \\
\text{representation}[\text{count}] & := x
\end{align*}
\]
Application to information hiding

Public part:
ADT specification ($E_1$)

Secret part:
- Choice of representation ($E_2$)
- Implementation of functions by features ($E_3$)
Object technology: A first definition

- Object-oriented software construction is the approach to system structuring that bases the architecture of software systems on the types of objects they manipulate — not on “the” function they achieve.
Object-oriented software construction is the construction of software systems as structured collections of (possibly partial) abstract data type implementations.
Classes: The fundamental structure

- Merging of the notions of **module** and **type**:
  - Module = Unit of decomposition: set of services
  - Type = Description of a set of run-time objects ("instances" of the type)

- The connection:
  - The services offered by the class, viewed as a module, are the operations available on the instances of the class, viewed as a type.
Class relations

- Two relations:
  - Client
  - Heir
Overall system structure

CHUNK
- space_before
- space_after
- add_space_before
- add_space_after

PARAGRAPH
- word_count
- justified
- add_word
- remove_word
- justify
- unjustify

FIGURE

WORD
- length
- font
- hyphenate_on
- hyphenate_off

FEATURES

QUERIES

COMMANDS
- set_font
- Client

Inheritance
deferred class
  COUNTER

feature
  item: INTEGER is
  -- Counter value
defered
end

up is
  -- Increase item by 1.
defered
ensure
  item = old item + 1
end

down is
  -- Decrease item by 1.
defered
ensure
  item = old item - 1
end

invariant
  item >= 0
end
End of lecture 4