Lecture P5: Abstract Data Types

Overview

Data type:
- Set of values and collection of operations on those values.

Example: int
- Set of values: between -32,768 and 32,767 (typically).
- Operations: +, -, *, /, %, printf("%d"), sqrt
- How is an int represented?
  - 16 bits
  - negative integers

Separate implementation from specification.
- INTERFACE: specify the allowed operations.
- IMPLEMENTATION: provide code for operations.
- CLIENT: code that uses operations.

Abstract data type (ADT):
- Data type whose representation is HIDDEN.
- Don’t want client to directly manipulate data type.
- Operations ONLY permitted through interface.

Principle of least privilege.

"Non ADT's"

Is complex data type an ABSTRACT data type?

client.c

```c
#include <stdio.h>
#include "COMPLEX.h"

int main(void) {
    Complex a = COMPLEXinit(1.0, 2.0);
    a.re = 5.0;
    COMPLEXshow(a);
    return 0;
}
```

legal C, but very bad software design

Violates "Principle of least privilege"
ADT's for Stacks and Queues

Fundamental data type.
- Set of operations (insert, delete) on generic data.

Stack ("last in first out" or LIFO).
- push: add info to the data structure
- pop: remove the info MOST recently added
- initialize, test if empty

Queue ("first in first out" or FIFO).
- put: add info to the data structure
- get: remove the info LEAST recently added
- initialize, test if empty

Could use EITHER array or "linked list" to implement EITHER stack or queue.

Stack Interface and Client

void STACKinit(void);
int  STACKisempty(void);
void STACKpush(int);
int  STACKpop(void);

STACK.h

#include "STACK.h"

int main(void) {
    int a, b;
    ... STACKinit();
    STACKpush(a);
    ... b = STACKpop();
    return 0;
}

client.c

Stack Implementation with Arrays

Push and pop at the end of array.

Demo:

big enough?

Stack Implementation with Arrays

Push and pop at the end of array.

Demo:

Stack Implementation with Arrays

Stack Implementation with Arrays

Stack Implementation with Arrays

Balanced Parentheses

int balanced(char a[], int n) {
    int i;
    STACKinit();
    for (i = 0; i < n; i++) {
        if ('(' == a[i])
            STACKpush(a[i]);
        else{
            if (STACKisempty())
            return 0;
            STACKpop();
        }
    }
    return !STACKisempty();
}

parentheses.c

Balanced Parentheses

Balanced Parentheses

Good:  ( ( ) ( ) )
Bad:   ( ( ) ) ( ( )

Balanced Parentheses
Balanced Parentheses

#include <stdio.h>
#include "STACK.h"
#define NMAX 1000

int main(void) {
    int c, n = 0;
    char a[NMAX];
    while ((c = getchar()) != EOF)
        if (c == '(' || c == ')
            a[n++] = c;
    if (balanced(a, n))
        printf("balanced\n")
    else
        printf("unbalanced\n");
    return 0;
}  

check if balanced

check if balanced

Read from stdin, ignoring non-parentheses.

Unix

% gcc parentheses.c stackarray.c
% a.out < myprog.c balanced
% a.out < parentheses.c unbalanced

How could valid C program have unbalanced parentheses?

Exercise: extend to handle square and curly braces.

- Good: ( [[ [ ] ] ])
- Bad: ( [ ] ])

Postfix Evaluation in C

#include <stdio.h>
#include <ctype.h>
#include "STACK.h"

int main(void) {
    int c;
    STACKinit();
    while ((c = getchar()) != EOF) {
        if ('+' == c)
            STACKpush(STACKpop() + STACKpop());
        else if ('*' == c)
            STACKpush(STACKpop() * STACKpop());
        else if (isdigit(c))
            STACKpush(c - '0');
    }
    printf("top of stack = %d\n", STACKpop());
    return 0;
}  

pop 2 elements and push sum

convert char to integer and push

Practical example of use of stack abstraction.

Put operator after operands in expression.

- Use stack to evaluate.
  - operand: push it onto stack.
  - operator: pop operands, push result.
- Systematic way to save intermediate results.

Example 2a: convert 27531 from octal to decimal.

- 2 8 8 8 * * * * 7 8 8 * * * 5 8 8 * * 3 8 * 1 + + + +

Example 2b: convert 27531 from octal to decimal.

- 2 8 * 7 + 8 * 5 + 8 * 3 + 8 * 1 +
- Stack never has more than two numbers on it!
- Horner’s method (see lecture A3).
Program has some flaws.
- What happens with input 2 + 5
- What happens with input 16 12 +

Unix

```bash
% gcc postfix.c stackarray.c
% a.out
  2 4 +
  top of stack = 6

% a.out
  1 2 3 4 5 + 6 * 7 8 9 + + *
  top of stack = 6624

% a.out
  5 9 8 + 4 6 * 7 + *
  top of stack = 2075

% a.out
  2 8 * 7 + 8 * 5 + 8 * 3 + 8 * 1 +
  top of stack = 12121
```

ADT Review

Client can access data type ONLY through implementation.
- Example: STACK implementation.

Representation is HIDDEN in the implementation.
- Provides security.

Convenient way to organize large problems.
- Decompose into smaller problems.
- Substitute alternate solutions (time / space tradeoffs).
- Separation compilation.
- Build libraries.
- Different client can share the same ADT.

Powerful mechanism for building layers of abstraction.
- Client works at a higher level of abstraction.

First Class ADT

So far, only 1 stack per program.

First Class ADT:
- ADT that is just like a built-in C type.
- Can declare multiple instances of them.
- Pass specific instances of them to interface as inputs.
- Details omitted in COS 126 - see Sedgewick 4.8 or COS 226 if interested.

```c
STACKinit();
...
STACKpush(a);
...
b = STACKpop();
```

Stack s1, s2;
s1 = STACKinit();
s2 = STACKinit();
...
STACKpush(s1, a);
STACKpush(s2, b);
...
c = STACKpop(s2);

PostScript

Language of most printers nowadays.
- Postfix language.
- Abstract stack machine.

Ex: convert 97531 from octal to decimal
- 9 8 mul 7 add 8 mul 5 add 8 mul 3 add 8 mul 1 add

Stack uses:
- Operands for operators.
- Arguments for functions.
- Return value(s) for functions.
PostScript

Some commands:
- Coordinate system: rotate, translate, scale, ...
- Turtle commands: moveto, lineto, rmoveto, rlineto, ...
- Graphics commands: stroke, fill, ...
- Arithmetic: add, sub, mul, div, ...
- Stack commands: copy, exch, dup, currentpoint, ...
- Control constructs: if, ifelse, while, for, ...
- Define functions: /XX { ... } def

Everyone’s first PostScript program (draw a box).

```
%!  
50 50 translate  
0 0 moveto 0 512 rlineto 512 0 rlineto  
0 -512 rlineto -512 0 rlineto  
stroke  
showpage
```

Overview

Data type.
- Set of values and collection of operations on those values.

ABSTRACT data type (ADT).
- Data type whose representation is completely HIDDEN from client.
  - client can’t directly manipulate data type
  - operations only permitted through interface
- Powerful software engineering model.
  - different clients can use the same ADT
  - can change ADT without changing clients
  - client works at a higher level of abstraction

Stacks and queues.
- Fundamental abstract data type.
  - calculators
  - printers - PostScript language
  - functions (see next lecture)

Lecture P5: Supplemental Notes

Queue Interface and Implementation

Queue operations.
- QUEUEinit(): initialize empty queue.
- QUEUEisempty(): return 1 if queue is empty; 0 otherwise
- QUEUEput(int): insert new item at end of list.
- QUEUEget(): return first item at beginning of list.

```
&%  
void QUEUEinit (void);  
int  QUEUEisempty (void);  
void QUEUEput (int);  
int  QUEUEget (void);  
```
Queue Interface and Implementation

```c
#include "QUEUE.h"
#define N 1000

static int q[N];
static front, back;

void QUEUEinit(void) {
    front = N;
    back = 0;
}

int QUEUEisempty(void) {
    return front % N == back;
}

void QUEUEput(int item) {
    q[back++] = item;
    back = back % N;
}

int QUEUEget(void) {
    front = front % N;
    return q[front++];
}
```

### Variables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>17</td>
<td>34</td>
<td>2</td>
<td>5</td>
<td>8</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

N = 7