Lecture P5: Abstract Data Types

Review

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

Example: int
- Set of values: between -32,767 and 32,767 (minimum limits).
- Operations: +, -, *, /, %, printf("%d"), sqrt
- How is an int represented?

Overview

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
#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

Stack operations.
- STACKinit(): initialize empty stack
- STACKisempty(): return 1 if stack is empty; 0 otherwise
- STACKpush(int): insert new item
- STACKpop(): delete and return item most recently added

STACK.h
void STACKinit(void);
int STACKisempty(void);
void STACKpush(int item);
int STACKpop(void);

Stack Implementation with Arrays

Push and pop at the end of array.

Demo:

```
s[N] = item;
N++;
N--;
return s[N];
```

Stack Client: Balanced Parentheses

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

int main(void) {
    int c, balanced = 1;
    STACKinit();
    ... /* MAIN CODE HERE */
    if (balanced)
        printf("Balanced.\n");
    else
        printf("NOT Balanced.\n");
    return 0;
}

Good: ( ( ( ) ( ) ) )
Bad: ( ( ) ) ( )
```
Stack Client: Balanced Parentheses

Check if your C program has unbalanced parentheses.

Unix
% gcc par.c stackarray.c
% a.out < myprog.c
balanced
% a.out < someprogram.c
unbalanced

Exercise: extend to handle square and curly braces.
- Good: ( ( [ [ ] ) ( ) ] )
- Bad: ( ( [ ] ) )

Stack Client: Postfix Evaluation

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 1.
1 2 3 4 5 * + 6 * * 7 8 9 + + * +

Stack never has more than two numbers on it!
Horner’s method (see lecture A3).
```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;
}
```

**Program has some flaws.**

```
% 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
```

### Stack Client: Postfix Evaluation

**Stack Client: Infix to Postfix**

```
infix2postfix.c
#include <stdio.h>
#include <ctype.h>
#include "STACK.h"

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

**Unix**

```
% gcc infix2postfix.c ...
% a.out
(2 + ((3 + 4) * (5 * 6)))
2 3 4 + 5 6 * * +
```

**Infix to postfix algorithm:**

- Left paren: ignore.
- Right paren: pop and print.
- Operator: push.
- Digit: print.

### ADT Review

**Client can access data type ONLY through interface.**

- Example: STACK.

**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 clients 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.)

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

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

First Class ADT Client: Infix

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

int main(void) {
  Stack s1 = STACKinit();
  Stack s2 = STACKinit();
  int c, op;
  while ((c = getchar()) != EOF) {
    if (c == ')') {
      op = STACKpop(s1);
      if (op == '+')
        STACKpush(s2, STACKpop(s2) + STACKpop(s2));
      else if (op == '*')
        STACKpush(s2, STACKpop(s2) * STACKpop(s2));
    } else if (c == '+' || c == '*')
      STACKpush(s1, c);
    else if (isdigit(c))
      STACKpush(s2, c - '0');
  }
  printf("Result = %d\n", STACKpop(s2));
  return 0;
}
```

infix.c

Unix

```
% gcc infix.c ...
% a.out
(2 + ((3 + 4) * (5 * 6)))
212
```

PostScript: Abstract Stack Machine

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

Ex: convert 27531 from octal to decimal.
- 2 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.

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

PostScript: Abstract Stack Machine

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).
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.

Stacks and queues.
- Fundamental ADT's.
  - calculators
  - printers and PostScript language
  - compiler uses to implement functions (see next lecture)

"Non ADT's"

Is complex data type an ABSTRACT data type?
- NO: Representation in interface.

Are C built-in types like int ADT's?
- ALMOST: we generally ignore representation.
- NO: set of values depends on representation.
  - might use \( x \ & \ 0 \) to test if even
  - works only if they're stored as "two's complement integers"
- CONSEQUENCE: strive to write programs that function properly independent of representation.
  - \( x \ \% \ 2 == 0 \) is more portable way to test if even
  - also, use \<limits.h\> for machine-specific ranges of int, long

Queue Interface

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 and remove item at beginning of list.

```c
#include "QUEUE.h"

void QUEUEinit(void);
int QUEUEisempty(void);
void QUEUEput(int item);
int QUEUEget(void);
```
Queue Implementation

#include "QUEUE.h"
#define MAX_SIZE 1000

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

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

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

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

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

Queue Client: Josephus Problem

Flavius Josephus. (first century)

- Band of 41 Jewish rebels trapped in cave by Romans.
- Preferring suicide to capture, rebels formed a circled and killed every 3rd remaining person until no one was left.
- Where should you stand to be among last two survivors?

```
#include <stdio.h>
#include "QUEUE.h"
#define N 41
#define M 3

int main(void) {
    int i;
    QUEUEinit();
    for (i = 1; i <= N; i++)
        QUEUEput(i);
    while (!QUEUEisempty()){
        for (i = 0; i < M - 1; i++)
            QUEUEput(QUEUEget());
        printf("%d\n", QUEUEget());
    }
    return 0;
}
```