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 Rational data type an ABSTRACT data type?

legal C, but very bad software design

RAT.h

```c
typedef struct {
    int num;
    int den;
} Rational;

Rational RATadd(Rational a, Rational b);
Rational RATmul(Rational a, Rational b);
Rational RATshow(Rational a);
Rational RATinit(int x, int y);
```

client.c

```c
#include "RAT.h"

int main(void) {
    Rational a;
    a.num = 5;
    a.den = 8;
    RATshow(a);
    return 0;
}
```

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

```c
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:

```
#include "STACK.h"
#define MAX_SIZE 1000
static int s[MAX_SIZE];
static int N;
void STACKinit(void) {
    N = 0;
}
int STACKisempty(void) {
    return N == 0;
}
void STACKpush(int item) {
    s[N++] = item;
}
int STACKpop(void) {
    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

How could valid C program have unbalanced parentheses?

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

Stack Client: Balanced Parentheses

while (balanced && (c = getchar()) != EOF) {
   if (c == '(')
      STACKpush(c);
   else if (c == ')
      if (STACKisempty())
         balanced = 0;
      else
         STACKpop();
   }
   if (!STACKisempty())
      balanced = 0;
}

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 + + + +

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 8 * * * * 7 8 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 P2).
# Stack Client: Postfix Evaluation

**postfix.c**

```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

**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: Infix to Postfix

**infix2postfix.c**

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

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

- Infix to postfix algorithm:
  - Left paren: ignore.
  - Right paren: pop and print.
  - Operator: push.
  - Digit: print.

**Unix**

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

# 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.
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.

Summary

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)

Lecture P5: Supplemental Notes
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);
```

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;
}
```

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.

```
#include "QUEUE.h"

void QUEUEInit(void);
int QUEUEIsEmpty(void);
void QUEUEPut(int);
int QUEUEGet(void);
```
Queue Implementation

```c
#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;
}
```

```c
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 circle 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
", QUEUEget());
    }
    return 0;
}
```