Lecture P6: 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.

Bond: What's your escape route?
Saunders: Sorry old man. Section 26 paragraph 5, that information is on a need-to-know basis only. I’m sure you’ll understand.

"Non ADT's"

Is Rational data type a truly ABSTRACT data type?

RAT.h

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

client.c

```c
#include "RAT.h"
int main(void) {
    Rational a;
    a.num = 5;
    a.den = 8;
    RATshow(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

#include "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.

Drawback.

Stack Client 1: Balanced Parentheses

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

int main(void) {
    int c, balanced = 1;
    STACKinit();
    // MAIN CODE HERE - see next slide
    if (balanced) {
        printf("Balanced.
");
    } else {
        printf("NOT Balanced.
");
    }
    return 0;
}

Good: ( ( ( ) ( ) ) )
Bad: ( ( ) ) ( )
Stack Client 1: 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 2: 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 (and avoid parens).

Example 1.
- Infix: \((1+((2*((3+(4*5))*6)))*(7+(8+9))))\)
- Postfix: 1 2 3 4 5 * + 6 * * 7 8 9 * + * +

J. Lukasiewicz (1878-1956)

Stack Client 2: 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 2a. (convert 27531 from octal to decimal)
- \(288888*8888*8*3881+++\)

Example 2b. (convert 27531 from octal to decimal another way)
- \(28*7+8*5+8*3+8*1+\)
  - Stack never has more than two numbers on it!
  - Horner’s method (recall Arrays Lecture).
Stack Client 2: Postfix Evaluation

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

int main(void) {
    int c;        // int not char
    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;
}
```

Stack Client 2: Postfix Evaluation

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 3: Infix to Postfix

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.

Putting it Together

Unix
```
% gcc126 postfix. stackarray.c
% a.out
(2 + ((3 + 4) * (5 * 6)))
Ctrl-d
```

Windows
```
C:\> lcc126 postfix. stackarray.c
C:\> lcc126 infix2postfix.c stackarray.c
```

C:\> infix2postfix | postfix
(2 + ((3 + 4) * (5 * 6)))
Ctrl-z
```
top of stack = 212
```
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.
  - And one person’s client can be another’s implementation.
  - Layer upon layer.

PostScript: Abstract Stack Machine

Language of most printers nowadays.
  - Postfix-based 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.

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

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 stack to implement functions (see next lecture)
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.

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;
}

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\n", QUEUEget());
    }
    return 0;
}