

Announcements

Reminder: Candidates on campus in next few weeks
Day-by-day schedule may change, so check often

Thursday Extra:

Monday, November 25 @ 4:00 pm

Tenure-track candidate #1

Kevin Angstadt, U. of Michigan

All Computers Great and Small: Supporting and Security the
Systems of Tomorrow

Notes on Supplemental Problem 4

Note schedule change

Program Management Notes

Switch Lab Partners today

Consider histogram.c — a solution to Supplemental Problem 4

Features

- No use of strings; only array `int count [26]`
- Input: No 1D array, as input character not needed after processing
- Counting: no need to search an alphabetic array or a sequence of letters; just use `count [tolower (ch) - 'a'] ++;`
- Output
 - No 2D array for output, as can print based on counts
 - Outer loop goes from max to 1, reflecting row/line count

Program statistics

- Total lines: 48
- Blank lines: 8
- Lines with only comments: 8
- Lines with only `}`: 6
- Lines with `#include`: 2
- Lines with `main/return 0`: 2
- Lines with non-trivial code: 23

Program Management Notes

- compiling program hello-world.c with 1 gcc command
gcc -o hello-world hello-world.c
 - translate hello-world.c to machine language
 - link in pre-compiled libraries
- compiling in 2 steps
gcc -c hello-world.c (obtain hello-world.o)
gcc -o hello-world hello-world.o
- role of #include
- challenge of duplicate declarations

Work from Monday's lab to be submitted

- final versions of files scheme-lists.h, scheme-lists.c, and scheme-lists-test.c, Makefile
- Statement of what actually happened for steps 7 and 8

Program Management Notes, Continued

Some lines in the readings contain the -Wall flag
(e.g., gcc -Wall -c scheme-lists.c)

What does this mean/do?

- A. Be kind, writing this drove me up the wall!
- B. Use the Wall library, as needed
- C. Report all warnings
- D. Extend the C Standard using the Wall Report
- E. Something else

Announcements

Stacks and Queues

Two levels of interest

Conceptual: FIFO or FILO/LIFO

Implementation: Either arrays or linked lists

Both widely used in practice!

Stacks: piles of materials (e.g., on a desk)
constrained run-time stack

Queues: lines in stores/airports
jobs sent to printer (if strict FIFO)

Questions; Clicker Questions

Assume:

add	means either	push	or	enqueue
remove	means either	pop	or	dequeue
get	means either	top	or	front

Notes:

- most stack ADTs include `top`
- queue ADTs may or may not include `front`

For which ADTs can a get operation be implemented simply from add and remove? (A simple extra variable might be used, but not an extra array, linked list, or other structure.)

- A. Stack
- B. Queue
- C. Both
- D. Neither

Assume:

- add means either push or enqueue
- remove means either pop or dequeue
- get means either top or front

Consider the program:

```
Container a;  
add(a, "Orange");  
add(a, "Red");  
add(a, "Green");  
remove(a);  
add(a, "Blue");  
printf("%s", get(a));
```

The program prints "Red".
Container must be a ...

- A.Stack
- B.Queue
- C.Neither
- D.Impossible to tell

Assume:

- add means either push or enqueue
- remove means either pop or dequeue
- get means either top or front

Consider the program:

```
Container b;  
add(b, "Happy");  
remove(b);  
add(b, "Sneezy");  
add(b, "Dopey");  
add(b, "Grumpy");  
printf("%s", get(b));
```

The program prints
"Dopey".

Container must be a ...

- A.Stack
- B.Queue
- C.Neither
- D.Impossible to tell

Assume:

- add means either push or enqueue
- remove means either pop or dequeue
- get means either top or front

Consider the program:

```
Container c;  
add(c, "Helium");  
add(c, "Neon");  
remove(c);  
printf("%s", get(c));
```

The program prints
"Helium".

Container must be a ...

- A.Stack
- B.Queue
- C.Neither
- D.Impossible to tell

Header file stringStack.h is given, where details of the stringStack typedef will be included.

Implementation details will be given in file stringStack.c

Assume a user's program will use the stringStack by writing
`#include "stringStack.h"`

What changes should be needed in a user' program if the implementation changes?

- A. None
- B. Modest changes likely
- C. Extensive review needed

```
/* type declaration for a stack of strings
 */
typedef
. . .
stringStack;

/* initialize stack referenced by the parameter
 */
void initializeStack (stringStack * stack) ;

/* return whether the stack is empty
 */
int isEmpty (stringStack stack) ;

/* return whether the stack is full
 */
int isFull (stringStack stack) ;

/* insert given item onto top of the stack
 */
int push (stringStack *stack, char * item) ;

/* removes top item from stack
   referenced by parameter
 */
char * pop (stringStack *stack) ;

/* returns a pointer to top item on the stack
 */
char * top (stringStack stack) ;
```

Array Implementation for Stacks

```
#define MaxStack 50;    /* MaxStack stands for the size  
                        of all stack arrays */  
  
typedef struct {  
    int topPosition;  
    char * stringData [MaxStack];  
} stringStack;          /* type for a stack of strings */
```

Two interpretations for topPosition

1. where current top data item resides in stack
2. where next data item will be placed in stack

For these interpretations, how should topPosition be initialized?

- | | |
|--------------------------|---------------------------|
| A. Impl.1: topPosition=0 | C. Impl.1: topPosition=-1 |
| Impl.2: topPosition=-1 | Impl.2: topPosition=-1 |
| B. Impl.1: topPosition=0 | D. Impl.1: topPosition=-1 |
| Impl.2: topPosition=0 | Impl.2: topPosition=0 |
| | E. Something else |

Switch lab partners today

Fill out peer educator survey!!!!

Link in email sent last night