University of Toronto Scarborough

CSCB09 Summer 2020 Midterm Test

Duration - 1 hour 15 minutes

Aids allow: Open book

Solution and Marking

1. The `PATH` environment variable stores a lot of directory names, separated by colons, e.g.,

```
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/my game* :/usr/local/cms/bin
```

(Yes, it is legal for pathnames to contain trailing spaces and `*`.)

Editing it by hand is very annoying. This question is about developing a command to help. The end-goal is a command `pathdel` that deletes a directory from `PATH`, e.g.,

```
$ pathdel '/usr/local/my game* '
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/cms/bin:
```

(Note: The trailing colon is a non-ideal artifact of our implementation technique. It's harmless in practice.)

(a) [2 marks] From documentations, `--help` messages, and/or experiments, find out: What does this command do? As in: What kind(s) of lines from stdin will **not** be output to stdout?

```
grep -F -v -x '0*'
```

Answer: lines that are exactly 0*

Marking:

```
-1 mark for each mistake:
* overlooked -F, e.g., treat 0* as regex
* overlooked -v, e.g., output lines matching 0*
* overlooked -x, e.g., not output lines containing 0*
```

(b) [9 marks] Write a shell function `pathdel` that updates `PATH` with deleting the directory name given by the 1st parameter. If the directory name isn't in `PATH`, no change apart from possibility the extra colon at the end. Examples:

```
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/my game* :/usr/local/cms/bin
$ pathdel '/usr/local/my game* '
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/cms/bin:

$ pathdel .
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/cms/bin:
```

Use no temp files or bash extra features. It's OK to assume: The user gives 1 parameter; pathnames don't contain newlines or colons this time (even though Unix allows it)

Hint: Part (a) is useful. But before using it, you need to have one directory per line; after using it, you need to merge lines back into using colons between directories.

Solution:

```
pathdel() {
  PATH=$(echo -n "$PATH" | tr : '\n' | grep -F -v -x "$1" | tr '\n' : )
}
```

Marking:

```
{ Eligible for these marks iff no temp files, no bash extra features
  1 mark: echo -n "$PATH", missing -n is OK this time
  1 mark: tr : '\n'
  1 mark: the grep, especially the $1
  1 mark: tr '\n' :
  2 mark: correct pipelining order
}
1 mark: PATH=$() around the above
1 mark: pathdel() { }
1 mark: quoting for vars and \n, deduct this mark if any quoting
        is missing!

Note: Quoting around $() is optional when using it to set var.
```

(c) [2 marks] My friend has implemented `pathdel` as an executable shell script, not as a shell function:

```
#!/bin/sh
<Like your code but not inside a function>
```

This file has filename `pathdel`, has the executable permission flags (`chmod a+x`), and is in a directory listed in `PATH`. Nothing else is called `pathdel` on the system. Still, it has no effect:

```
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/cms/bin
$ pathdel '/bin'
$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/cms/bin:
```

Why?

Answer:

- this version launches new process
- deletion happens in new process
- nothing brings result to current shell

Marking:

```
Hitting each bullet point is 1 mark,
just need 2 for full marks.
Within reason you can reward other points not shown above.

No bonus mark for hitting more than 2. :)
```

2. [6 marks] Prof. C and Prof. Y have a lot of PDF files (filenames end with ".pdf"); for some of them, also the compressed versions (filenames end with ".pdf.gz"). Example:

```
$ ls
q2.sh  w.doc  w.doc.gz  x.pdf  x.pdf.gz  y.pdf  z.pdf  z.pdf.gz
```

They have decided that it is safe to delete certain PDF files if the compressed versions exist. They enlist your help to write a Bourne shell script (no bash extra features) for this. However! They want more control, so they will provide PDF pathnames on the command line, and the shell script should consider only those. They may make mistakes, so ignore pathnames that don't exist or don't end with ".pdf" (error messages not required today, not forbidden either). Example:

```
$ sh q2.sh y.pdf x.pdf w.doc
$ ls
q2.sh  w.doc  w.doc.gz  x.pdf.gz  y.pdf  z.pdf  z.pdf.gz
```

Explanation: Consider y.pdf and x.pdf only, ignore w.doc, and don't touch z.pdf; y.pdf stays because there is no y.pdf.gz, x.pdf is gone because there is x.pdf.gz.

If a pathname includes a directory, look for the compressed version there. Example:

```
$ ls
q2.sh  z.pdf  z.pdf.gz
$ ls ../music
tso.pdf  tso.pdf.gz  tsv.pdf
$ sh q2.sh z.pdf ../music/tso.pdf ../music/tsv.pdf
$ ls
q2.sh  z.pdf.gz
$ ls ../music
tso.pdf.gz  tsv.pdf
```

Since there is no ../music/tsv.pdf.gz, don't delete ../music/tsv.pdf.

Solution:

```
for f in "$@"; do
    case "$f" in
        *.pdf)
            if [ -e "$f.gz" ]; then
                rm "$f"
            fi
            ;;
    esac
done
```

Marking:

```
1 mark: loop over command line parameters (while-shift-loop OK)
```

```
1 mark: process *.pdf only, skip non-*.pdf
2 marks: check -e/-f foo.pdf.gz before rm
         lose all of these 2 marks if they miscalculate,
         e.g., for foo.pdf, they check foo.gz or foo.pdf.pdf.gz
         Reason: MAJOR DATA LOSS if the check is wrong.
1 mark: the rm
1 mark: quoting. deduct this mark if any necessary quoting is missing

I don't have marks for checking -e/-f foo.pdf,
not required for simplicity,
any of these is OK: don't care (my solution), rm -f, explicit check.
```

3. [9 marks] This question is about allocating and deallocating heap memory (malloc/calloc, free).

A struct type `player` is defined to have a player name (NUL-terminated string as usual) and amount of hitpoints:

```
typedef struct player {
  char *name;
  unsigned hitpoints;
} player;
```

Function `mk_player` should allocate heap memory for a new player (initial hitpoints 100, initial name copied from parameter). Note that the name should be a copy made in newly heap-allocated space, for maximum safety and independence. Function `free_player` should deallocate said heap memory. Here is their code, but they are missing the allocations and deallocations (note the blank lines). Your job is to put them back properly. For simplicity, no need to check for `NULL` today.

```
player *mk_player(const char *name)
{
  player *p;

  p->hitpoints = 100;

  strcpy(p->name, name);
  return p;
}

void free_player(player *p)
{


}
```

A starter file q3.c is provided to save typing. It also includes a main function that shows a simple use case.

(Solution and marking: next page.)

Solution:

```
player *mk_player(const char *name)
{
  player *p;
  p = calloc(1, sizeof(player));
  p->hitpoints = 100;
  p->name = calloc(strlen(name) + 1, sizeof(char));
  strcpy(p->name, name);
  return p;
}

void free_player(player *p)
{
  free(p->name);
  free(p);
}
```

Marking:

```
Note: I use calloc, but malloc also OK.
We are also kind, accept calloc(s, n) where calloc(n, s) is expected.

mk_player:
1 mark: p = calloc(1, sizeof(player))
PROVIDED THE ABOVE (close enough is OK):
4 marks: p->name = calloc(strlen(name) + 1, sizeof(char));
         Using 1 for sizeof(char) is OK, see the C standard :)
         If strlen(name): 1 mark
         If strlen(name)+2 or bigger: 3 marks
Very serious mistake I hope no one does it: If the other order,
1 mark or less for mk_player!

free_player:
4 marks: free(p->name);
         free(p);
         in that order
If reverse order: 1 mark
If missing one: 1 mark
```

4. This question is about designing and using a tagged union type.

The following struct type is defined:

```
typedef struct node {
  enum { LEAF, INTERNAL } tag;
  int x, y, width;
  struct node *child[4];  // used when tag==INTERNAL only
  unsigned char rgb[3];   // used when tag==LEAF only
} node;
```

This is a node in a *quadtree*, which is useful for storing a square image (story for another day). Each node is one of these two cases:

- `tag`=`LEAF`: no children, has RGB colour intensities, `rgb[0]` is red intensity.
- `tag`=`INTERNAL`: has 4 child pointers (all non-NULL), no RGB intensities.

But both have x-y coordinates and width. (And `tag` obviously.)

(a) [3 marks] Since `child` and `rgb` are mutually exclusive, modify the definition of `node` to use a union type over them.

Solution:

```
typedef struct node {
  enum { LEAF, INTERNAL } tag;
  int x, y, width;
  union {
    struct node *child[4];  // used when tag==INTERNAL only
    unsigned char rgb[3];   // used when tag==LEAF only
  } data;
} node;
```

Marking:

```
Note: the name "data" is up to the student.
1 mark: union over child and rgb
1 mark: "union { ... } <give name here>". Here is why:
        in "union typename { ... } varname;"
        for this question, typename is optional, varname is necessary,
        but someone will forget this >:)
1 mark: union not over other components,
        and other components are preserved
```

(b) [9 marks] Implement this recursive function to compute and return the maximum red intensity in the tree rooted at the given node `v`:

```
unsigned char max_red(const node *v)
```

It should work for a correctly modified definition of `node` that now uses a union type. You may assume that `v` and all child pointers are non-NULL. You may assume that every `tag` field has either `LEAF` or `INTERNAL`.

Solution:

```
unsigned char max_red(const node *v)
{
  unsigned char m = 0;
  int i;
  switch (v->tag) {
  case LEAF:
    return v->data.rgb[0];
  case INTERNAL:
    for (i = 0; i < 4; i++) {
      unsigned char c = max_red(v->data.child[i]);
      if (c > m) m = c;
    }
    return m;
  }
}
```

Marking:

```
Note: the name "data" is up to the student, from part (a).
But if they did "union foo { ... }" then pretend it's "foo".

1 mark: use v->tag to discern base case vs recursive case
1 mark: base case. if "v->rgb[0]", OK here, penalty below
1.5 mark: recursive case: recursive call for all 4 children.
          if "v->child[blah]", OK here, penalty below
1.5 mark: recursive case: max of the 4 answers, return it
2 marks: v->data.rgb, not v->rgb
2 marks: v->data.child, not v->child
(YES THAT HARSH >:) Reason: as much a test on using unions as
 1st-year recursion.)
```

5. Miscellaneous C knowledge. (Put all answers in the same q5.txt file, they're all short.)

   (a) [2 marks] On an old 16-bit computer system and its C compiler, `sizeof(double)`=8 and `sizeof(int)`=2. Given the two type definitions below, what were `sizeof(s)` and `sizeof(lingling)` on that system? Assume that in `s`, there is no gap between the two fields.

```
typedef struct s {
  double r[5];
  int a[5];
} s;
```

```
typedef union lingling {
  double r[5];
  int a[5];
} lingling;
```

   Answer and marking:

```
1 mark: sizeof(s)=40+10=50
1 mark: sizeof(lingling)=40
```

   (b) [2 marks] Given the following declarations, two questions: What is the type of `q`? What is the type of `y`?

```
double* p, q;
typedef double *t;
t x, y;
```

   Answer and marking:

```
1 mark: q is double
1 mark: y is pointer to double
```

   (c) [1 mark] To read a character from stdin, what's wrong with "`c = getchar();`", if `c` has been declared with "`char c;`"?

   Answer and marking:

```
1 mark: EOF does not fit in char.
```

   (d) [3 marks] A beginner in C has coded up an attempt to determine whether stdin is empty:

```
#include <stdio.h>
int main(void)
{
  if (feof(stdin)) {
    printf("empty\n");
  } else {
    printf("not empty\n");
  }
  return 0;
}
```

This is run with stdin redirected from an empty file. It mistakenly reports "not empty". Help the beginner by briefly answering: Why is this approach ineffective? And what is a correct approach?

Answer and marking:

```
1 mark: feof doesn't know until trying to read
2 marks: the fix, e.g., getchar() == EOF, getchar() then feof(stdin).
```

(End of questions.)