University of Toronto Scarborough

CSCB09 Summer 2020 Midterm Test

Duration - 1 hour 15 minutes

Aids allow: Open book

1:	/13
2:	/ 6
3:	/ 9
4:	/12
5:	/ 8
Total:	/48

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*'
```

(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.

(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?

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.

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.

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.

(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`.

A starter file q4.c (for both parts) is provided to save typing. It starts with the original `node` without union.

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;
```

   (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;
```

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

   (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?

(End of questions.)