McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 2S03
Exercise Sheet 10
Solution Hints

SFWR ENG 2S03 — Principles of Programming

15 November 2006

Exercise 10.1 — Association Lists — Web Spider (55% of Final, 2005)

The following C type definitions will be used to define "**URL sets**" implemented as **ordered linked lists**:

```
typedef struct UrlListNodeStruct { const char *                url;
                                   struct UrlListNodeStruct * next; } UrlListNode;

typedef UrlListNode * UrlSet;
```

Building on this, the following C type definitions will be used to record the link structure of set of web pages, again implemented as an **ordered linked list**, ordered define "**URL sets**" represented as ordered lists:

```
typedef struct PageListNodeStruct { const char *              pageURL;
                                    UrlSet                    links;
                                    struct
                                          PageListNodeStruct * next;   } PageListNode;
typedef PageListNode * PageSet;
```

The considered price lists will satisfy the following **invariants**:

(1) None of the char * fields is *NULL*.

(2) Each char * field points to a zero-terminated string.

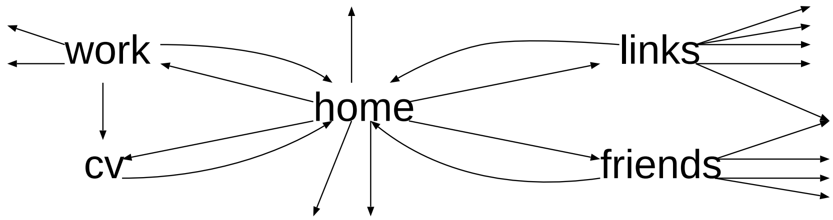
(3) Each char * field has been obtained by *malloc*().

(4) All lists are **ordered** with respect to their char * fields and the ordering implemented by the following library function:

> int *strcmp*(const char **s1*, const char **s2*);
>
> The *strcmp*() function compares the two strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

(5) In every list, no two of the char * fields contain matching (i.e., equal) strings.

A typical personal web site might have the following link structure, where arrows without named targets are **external** links:



(The solutions to the items (a,b) and (c) are ***independent of each other!***)

(a) $\boxed{\approx 11\%}$ Assuming — for simplicity — that each URL has 30 characters, we are interested in the memory demands for a 5-page *PageSet* representing the personal web site drawn above, on a 32-bit machine. (I.e., how many bytes will the whole *PageSet*, including all list nodes and strings, occupy?)

Different representations with different memory requirements are possible; identify and explain two different possibilities, and calculate the memory demands for each.

Clearly state your assumptions and document your reasoning.

(You do not have to calculate the results of multiplications if these results are greater than 400.)

Solution Hints

14 external links to 13 external targets.
9 internal links to 5 internal targets.

An *UrlListNode* needs 4 (url) + 4 (next) = 8 bytes.
An URL string needs (padded) 32 bytes.
A *PageListNode* needs 4 + 4 + 4 = 12 bytes.

In any case, we need 5 *PageListNode*s and 23 *UrlListNode*s: 5 * 12 + 23 * 8 = 60 + 184 = 244.

Two possibilities:

- No shared strings: 5 + 23 = 28 strings; altogether 244 + 28 * 32 = 244 + 896 = 1140 bytes.

- No duplicated strings — all occurrences of the same URL point to the same string: 5 + 13 = 18 strings; altogether 244 + 18 * 32 = 236 + 576 = 820 bytes.

(b) $\boxed{\approx 6\%}$ What are the consequences of the two alternatives you identified in (a) above for programming, for example an *insertPage* or a *deletePage* function?

(Do not program these functions here; just explain what differences between the two possibilities would be.)

Solution Hints

No shared strings: Easy: *malloc* for insertion and *free* every URL while deleting.

Shared strings: Have to find out for each inserted URL whether it is already there — if a *PageSet* has external links, this is expensive. For deletion, have to find out whether there still is a reference to the URL in question — even worse.

(c) $\boxed{\approx 38\%}$ **<u>Design and implement</u>** a C function with the following prototype:

UrlSet findExternal(*PageSet*);

Passed a *PageSet* as argument, this shoud create a *UrlSet* containing exaclty the external links referenced as *url* inside the argument *PageSet*, but not as *pageURL*. (Note (5) above!)

Decompose into sub-functions as appropriate, and describe the interface and specification of each sub-function.

Solution Hints

A URL is external of it is not a *pageURL* in the *PageSet* — we let the searching function return the list node containing as its *pageURL* a copy of *url*, or *NULL* if no such node exists.

$\boxed{2\% + 6\%}$

```
PageListNode * findPage( PageSet pages, const char * url ) {
  if ( pages ) {
    switch ( strcmp(pages->pageURL, url) ) {
      case  0:  return pages;
      case -1:  return findPage ( pages->next, url );
    }
  }
  // if we get here, "url" is not in the list
  return NULL;
}
```

Since we will not encounter external nodes in sequence, we have to perform ordered insertion, avoiding to create duplicates, updating the argument list *urls* passed by reference, and reporting *NULL* for out-of-memory, and (feature not used here) the address of the list node containing the inserted URL in the case of success.

$\boxed{3\% + 9\%}$

```
// assuming new does not need to be copied.
UrlListNode * insertURL( UrlSet * urls, const char * new) {
  if ( *urls ) {
    switch ( strcmp((*urls)->url, new) ) {
      case  0:  return *urls;
      case -1:  return insertURL ( &((*urls)->next), new );
    }
  }
  // if we get here, we have to insert as first element
```

```
  UrlListNode * result = malloc(sizeof(UrlListNode));
  if ( result ) {
    result→url = new;
    result→next = *urls;
    *urls = result;
  }
  return result;
}
```

With these utilities, the implementation of the *findExternal* function is straight-forward: Iterate through all links (in a nested while-loop); for each, check whether it is a key in the *PageSet*, and if not, insert it into the result *UrlSet*.

$\boxed{3\% + 15\%}$

```
UrlSet findExternal( PageSet pages) {
  PageListNode * currentPage = pages;
  UrlSet result = NULL;
  while ( currentPage ) {
    UrlListNode * currentLink = currentPage→links;
    while ( currentLink ) {
      if ( findPage ( pages, currentLink→url ) )
       {}     // internal
      else { // external
        if ( insertURL( &result, currentLink→url ) ) {} // OK
        else { // NOK
          fprintf(stderr, "findExternal:  Could not allocate memory for result node!\n");
        }
      }
      currentLink = currentLink→next;
    }
    currentPage = currentPage→next;
  }
  return result;
}
```
