Generic data structures

Computing Lab

Indian Statistical Institute

Generics: useful functions

```
#include <string.h>

int memcmp(const void *s1, const void *s2, size_t n);
void *memcpy(void *dest, const void *src, size_t n);
void *memmove(void *dest, const void *src, size_t n);
```

- `memcmp()`: compares the first `n` bytes (each interpreted as unsigned char) of the memory areas `s1` and `s2`
- `memcpy()`: copies `n` bytes from `src` to `dest` (memory areas must not overlap)
- `memmove()`: copies `n` bytes from `src` to `dest` (memory areas may overlap)

Generic stacks

```
#ifndef _GSTACK_
#define _GSTACK_

typedef struct {
    void *elements;
    size_t element_size, num_elements, max_elements;
} STACK;

int init_stack (STACK *s, int element_size, int capacity);
void free_stack(stack *s);
bool is_empty(const STACK *s);
int push(STACK *s, const void *eptr);
int pop(STACK *s, void *eptr);

#endif // _GSTACK_
```

Implementation notes

- Choose a default stack size initially (`max_elements`); `realloc()` to double the current size as needed
- Use `memcpy()` for `push()` and `pop()`

  Example:

```
stackElementAddress = (char *) s->elements + s->num_elements * s->
    element_size;
memcpy(stackElementAddress, argument, s->element_size); // for push
    ()
memcpy(argument, stackElementAddress, s->element_size); // for pop()
```

Exercises - I

1. Implement a generic data structure SEQUENCE to hold a list of elements of any one of the following types: `int`, `float`, or null-terminated strings (`char *`) (all elements in a particular list will be of the same type). Your data structure should support the operations given below.
   - `void get_element(SEQUENCE s, size_t i)`: prints the numerical value of the i-th element of the sequence `s`, if it exists. The function should print an error message if the i-th element does not exist. The numerical value of an integer n is n itself; the numerical value of a floating point number f is the integer nearest to f; the numerical value of a string s of alphanumeric characters is the sum of the ASCII values of all the characters contained in s. Note that the numerical value of an empty string is 0.
   - `size_t length(SEQUENCE s)`: returns the number of elements in the sequence `s`.

Exercises - II

- `void summation(SEQUENCE s)`: prints the sum of the numerical values of the elements in the sequence `s`. Please see above for the definition of numerical value for sequences of various types. For a sequence containing no elements, `summation(s)` should print 0.

2. Write a program that takes N sequences as input (from stdin), and prints the sequence `s` for which `summation(s)` is the maximum.

   Input format: A positive integer N, followed by N sequences, one per line. Each of these lines will begin with `i`, `f` or `s` to specify whether the sequence on that particular line consists of `int`, `float` or string elements. This single letter will be followed by a non-negative integer that specifies the number of elements in the sequence, which in turn will be followed by the elements of the sequence.

Exercises - III

3. Write a program that can accept a generic but homogeneous list of elements from the user and report the next greater element for every element in the list in linear time. Consider a lexicographic comparison of the elements (i.e., use `memcmp()`).