day-7-data-structures

September 6, 2023

1 Day 7 Data structures

1.0.1 Topics

- Data structures
  - singly-linked list
  - more advanced linked list
  - doubly-linked list
  - (optional) `calloc` and `realloc` implementation
  - (optional) C++ classes

1.0.2 1. Singly-linked list

Implement a program that stores numbers in a linked list and then prints them out. The nodes of your linked list should be an appropriate struct type and should be dynamically allocated using `malloc()`. The program should let the user input numbers at runtime and should contain a function that adds nodes to the end of the list. (Note that pressing `Ctrl+D` at the terminal makes `scanf()` return EOF. In the notebook, enter ``^D'' and press return to end user input.)

You might find it helpful to split your code into multiple files, including a header file for use in your `main` file, but it is up to you to choose how to organise your code.

1.0.3 2. More advanced linked list

Add functions to:

- delete the last number in the list
- add a number to the start of the list
- search for a number in the list and return either a pointer to it or `NULL` if the number is not in the list

1.0.4 3. Doubly-linked list

Change your code to use a doubly-linked list. Add a function that takes a pointer to a node in the list and deletes the corresponding node from the list. Remember to `free()` the memory used by the node!

1.0.5 4. Implementing `calloc` and `realloc` (optional)

Write functions `calloc2()` and `realloc2()`, that use `malloc()` and `free()` to implement the functionality of `calloc()` and realloc(), respectively. These functions will have the following declarations:

```
void *calloc2(size_t nmemb, size_t size);
void *realloc2(void *ptr, size_t old_size, size_t new_size);
```

Remember that `calloc()` sets the allocated memory to zero (for this exercise, you may ignore testing for integer overflows when multiplying the arguments of `calloc()` together). When implementing the copying part of `realloc()`, recall that char is 1 byte; the C standard states that you may use `char *` pointers to access individual bytes of memory. For `realloc2()`, since you don't know how large an area of memory `ptr` points to, you will need to provide the function with both the old size as well as the new size you are requesting. The real `realloc()` normally only needs the latter, as it has access to internal data structures used by `malloc()`, etc. to keep track of the former.

For reference, here are the definitions of the behaviour of `calloc()` and `realloc()` that you should implement:

- calloc
- realloc

1.0.6 5. C++ classes (optional)

Write a C++ class to represent a fraction. You will need to store two integer values: `numerator` and `denominator`. You will need to create a `main.cpp` to test it.

Add the following functionality: - Constructor taking two parameters `Fraction(numerator, denominator)` - A copy constructor - An output operator (<<) and input operator (>>) - Operators for mathematical functions: +, -, , /, +=, =, -=, /=,. . . - Comparison Operators: ==, !=, <, >, <=, >= - Add a function that simplifies each fraction (i.e. the value `2/4` is simplified to `1/2`) and use it to keep the fraction in its simplest form. You will need to call this in all of your mathematical functions.