DD1324 Applied Programming and Computer Science, Part 2

Laboratory Assignment 2 – Single-linked list

The goal of this lab assignment is write a small program that implements a single-linked list data structure. Each node of the list will store the name and age of a person. Each node will contain a pointer which points to the base address of the next node. The program should have functions which support the insertion of new nodes, removal of nodes, printing of the list, and counting the nodes in the list.

NAME

lab02 (single-linked list)

SYNOPSIS

lab02 [NO ARGUMENTS]

DESCRIPTION

When the program starts, it should query the user for the name and age of a person, and use this information to create the first node of the list.

```
Begin by typing a name followed by age: Marvin 25
```

After that, the program will prompt the user with the following choices repeatedly and executes the appropriate action until the user quits.

```
  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
```

SPECIFICATIONS

The program should be written to the following specifications.

1. Your project can follow a more simple organization than `Laboratory01`. All files can be placed into a single folder :
   a. One or more source files (`.c`)
   b. Zero or more header files (.h)
   c. `Makefile`
2. Your program does not need to parse any command line arguments
3. The main file of your program should do the following:
   a. Prompt the user to enter the `name` and `age` as data for the first node in the linked list when the program begins. It should create a single linked list of 1 node.

b. Repeatedly ask the user to perform one of the following actions. Each action should be handled by a different function.
    i. Add a node to the head of the list. This calls `list_append()`. When adding a node the user should be prompted to enter a `name` and age.
    ii. Insert a node in the middle of the list. This calls `list_insert()`. When adding a node the user should be prompted to enter a `name`, `age`, and the `index` where the node should be added.
    iii. Display the length of the list. This calls `count_list()`
    iv. Remove a node from the list. This calls `list_remove()`. When removing a node the user should be prompted to enter which node `index` should be removed.
    v. Print the list. This calls `print_list()`
    vi. Quit the program. This calls `list_destroy()`

4. The list should be implemented as a single linked list.
5. Use the debug header `dbg.h` to debug your code and to print errors for incorrect usage.
6. You should use good programming style. This includes informative comments.

EXAMPLE

The example below shows an example interaction with the program and the expected output.

```
$./lab02
  Begin the list by typing a name followed by age: Marvin 21
  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
0. Marvin 21

  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
a
   Type name followed by age: Alvin 14
 0. Alvin 14
 1. Marvin 21

  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
i
```

User enters initial data to create the first node in the linked list when the program starts

Calls `list_append()` to add a new node to the head of the linked list

Calls `list_insert()` to add a new node at `index=1` in the linked list

```
  Type name followed by the age: Jenny 24
  Type the index where you want to insert: 1
0. Alvin (14)
1. Jenny (24)
2. Marvin (21)

  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
r
```

Calls `list_remove()` to remove node at `index=2` from the linked list

```
  Type the index you want to remove: 2
0. Alvin (14)
1. Jenny (24)

  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
l
```

Calls `count_list()` to determine the number of nodes in the list

```
The list contains 2 names
  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
p
```

Calls `print_list()` to print the data from the members of the linked list

```
0. Alvin (14)
1. Jenny (24)

  What would you like to do?
  (a) add a node to the head of list
  (i) insert a node in the middle of the list
  (l) display the length of the list
  (r) remove a node from the list
  (p) print the list
  (q) quit
q
```

Calls `list_destroy()` and frees memory from remaining members of the linked list then exits the program

```
0. Alvin (14) DESTROYED
1. Jenny (24) DESTROYED
```

RESOURCES

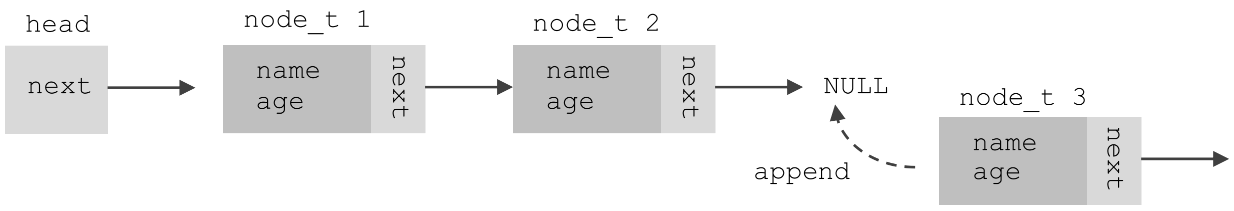
On the course web page you will find the following resources

- A zip file containing a program skeleton, `skeleton.c`. This file contains most of the code necessary to complete the laboratory. Several function definitions and a few other lines are missing. **Comments in the `skeleton.c` indicate where code is missing.** Note that you can choose to implement the program without `skeleton.c` if you choose to.

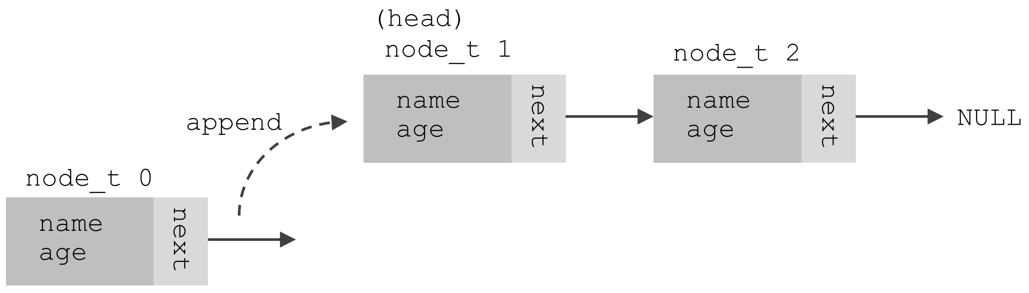
HINTS

The following list of hints may prove helpful.

- There are several ways to implement a single-linked list. Almost all implementations use some type of `node_t` structure data type to represent each element of the list. Some implementations will use another special `start` or `head` structure data type which points to the first element in the list. Implementations that use this approach usually add new elements to the end of the list by default.

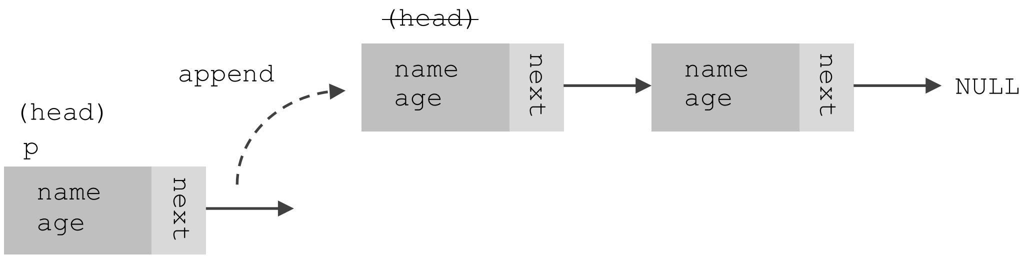


  The drawback of this approach is that it requires another special data type to represent the head. The implementation in `skeleton.c` simplifies this by eliminating the special head data type, and simply calling the first node in the chain the head. Note that in this implementation, it's easier to append to the other side of the linked list because the main function is always keeping track of the head node.

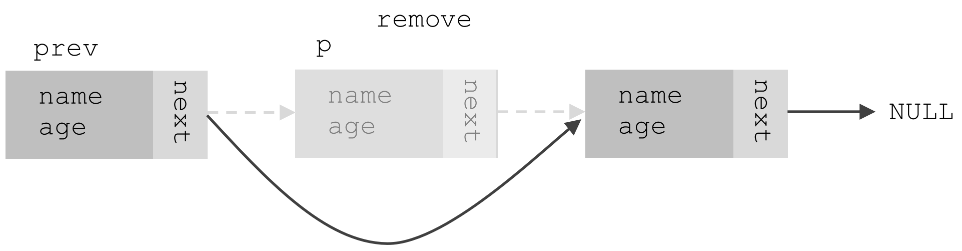


- In C the operator `->` is a special operator to access a member of a structure pointed to by a pointer variable. If `node_t * mynode` is a pointer to a structure, then `mynode->name` gives access to the `name` field. It is equivalent to writing `(*mynode).name`.
- Below is a list of functions in skeleton.c along with a description. Some functions are already written for you.
  - `node_create` A function to create a new node structure and allocate memory from the heap. It returns a `node_t* p` pointer to the new node. Creating the node requires 2 memory allocations, one for the structure (`node_t`) and one for the character array `node_t.name`
  - `node_destroy` A function to destroy an existing node and return memory to the operating system
  - `print_list` A function print all the names and ages in the linked list. It takes a pointer to the head node as an argument and loops through all nodes in the linked list, prints the name and age of each as it goes. It stops once it reaches the end of the list (`NULL` pointer)

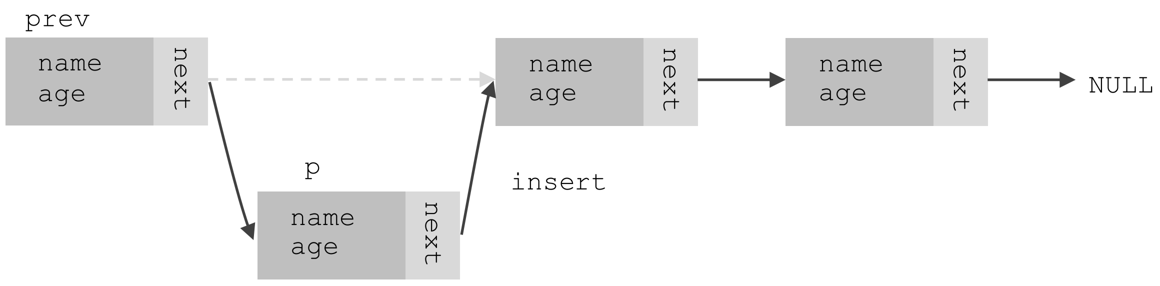
- `count_list` A function count how many nodes appear in the linked list. It takes a pointer to the head node as an argument and loops through all the nodes in the linked list, keeping count of how many it has traversed. Once it reaches the end of the linked list (`NULL` pointer) it prints the total number of nodes in the list
- `list_append` A function to add a new node to the head of the linked list. It takes a pointer to the head node as an argument, and then queries the user for the name and age. It calls `node_create()` to create the node, then fills the new node with the collected data. Finally it points the new node to the old head of the list and returns the new node as the new head of the linked list.



- `list_remove` A function that removes a node from the list. It takes the head of the list and returns a new head of the list. It prompts the user to type the index he/she wants to remove and stores the index. If the index is 0, the head node is destroyed and `head->next` becomes the new head. If another index is specified a loop is used to set the pointer *prev to the node previous to index we want to destroy. We assign `p` to `prev->next`. We then attach the link from `prev->next` to `p->next`. Finally we destroy `p`.



- `list_insert` A function that inserts a node into the linked list. It takes the head of the list and returns a new head of the list. It prompts the user to type the index he/she wants to insert to and stores the index. It calls `node_create()` to create the node, then fills the new node with the collected data. If the index is 0, the new node becomes the head node. If another index is specified a loop is used to set the pointer *prev to the node previous to index we want to insert at. We then assign `p->next` to `prev->next`, and `p` to `prev->next`.



- `list_destroy` A function to destroy the entire linked list. It is given a pointer to the head of the list, then iterates through the list, destroying each node it goes (freeing memory)

EVALUATION

Arrange to demonstrate your code to one of the instructors when you are ready.

1. Before meeting, print out the Laboratory Assessment Protocol form (from the course web site) and bring it to the meeting.
2. You will be asked to demonstrate your code. This will consist of several steps:
   a. You should show that your program meets the specifications and produces the correct output.
   b. You will be asked to compile your code using your Makefile.
   c. The instructor will test your code by running the program and creating a list
   d. You should walk the instructor through each step of your program, explaining the overall architecture, and the purpose of each function. You should be prepared to answer questions about your code.
3. You will be asked to demonstrate that your code does not have any memory leaks using `valgrind`
4. You should be prepared to show how to debug your code using `gdb`.
5. Email your source code in a zip file to the course instructor ksmith@kth.se