Evan Noyes and Henry Gridley

Lab 3

Assignment 1

```
Breakpoint 1, PrintPerson (person=0xbefff684) at person.c:10
10      printf("%s is %d years old\n",
(gdb) print person
$1 = (struct Person *) 0xbefff684
(gdb) print *person
$2 = {name = "John\000\333\374\266\255\204\000\000\000\000\000\000\271\203\000", age = 10}
(gdb) print person->name
$3 = "John\000\333\374\266\255\204\000\000\000\000\000\000\271\203\000"
(gdb) print person->age
$4 = 10
(gdb)
```

Assignment 2

Source code for linkedlist.c (Also included in submission)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int id_count = 1;

struct Person
{
        // Unique identifier for the person
        int id;
        // Information about person
        char name[20];
        int age;
        // Pointer to next person in list
        struct Person *next;
};

struct List
{
        // First person in the list. A value equal to NULL indicates that the
        // list is empty.
        struct Person *head;
        // Current person in the list. A value equal to NULL indicates a
        // past-the-end position.
        struct Person *current;
        // Pointer to the element appearing before 'current'. It can be NULL if
```

```
	// 'current' is NULL, or if 'current' is the first element in the list.
	struct Person *previous;
};

// Give an initial value to all the fields in the list.
void ListInitialize(struct List *list)
{
	list->head = NULL;
	list->current = NULL;
	list->previous = NULL;
}

// Move the current position in the list one element forward. If last element
// is exceeded, the current position is set to a special past-the-end value.
void ListNext(struct List *list)
{
	if (list->current)
	{
		list->previous = list->current;
		list->current = list->current->next;
	}
}

// Move the current position to the first element in the list.
void ListHead(struct List *list)
{
	list->previous = NULL;
	list->current = list->head;
}

// Get the element at the current position, or NULL if the current position is
// past-the-end.
struct Person *ListGet(struct List *list)
{
	return list->current;
}

// Set the current position to the person with the given id. If no person
// exists with that id, the current position is set to past-the-end.
void ListFind(struct List *list, int id)
{
	ListHead(list);
	while (list->current && list->current->id != id)
		ListNext(list);
}

// Insert a person before the element at the current position in the list. If
// the current position is past-the-end, the person is inserted at the end of
// the list. The new person is made the new current element in the list.
```

```
void ListInsert(struct List *list, struct Person *person)
{
	// Set 'next' pointer of current element
	person->next = list->current;
	// Set 'next' pointer of previous element. Treat the special case where
	// the current element was the head of the list.
	if (list->current == list->head)
		list->head = person;
	else
		list->previous->next = person;
	// Set the current element to the new person
	list->current = person;
}

// Remove the current element in the list. The new current element will be the
// element that appeared right after the removed element.
void ListRemove(struct List *list)
{
	// Ignore if current element is past-the-end
	if (!list->current)
		return;
	// Remove element. Consider special case where the current element is
	// in the head of the list.
	if (list->current == list->head)
		list->head = list->current->next;
	else
		list->previous->next = list->current->next;
	// Free element, but save pointer to next element first.
	struct Person *next = list->current->next;
	free(list->current);
	// Set new current element
	list->current = next;
}

void PrintPerson(struct Person *person)
{
	printf("\nPerson with ID %d:\n", person->id);
	printf("\tName: %s\n", person->name);
	printf("\tAge: %d\n\n", person->age);
}

void print_menu()
{
	printf("Main menu:\n\n");
	printf("1. Add a person\n");
	printf("2. Find a person\n");
	printf("3. Remove a person\n");
	printf("4. Print the list\n");
	printf("5. Exit\n\n");
```

```
	printf("Select an option: ");
}

void strip_newline(char *s)
{
	while (*s)
	{
		if (*s == '\n')
		{
			*s = 0;
			return;
		}
		s++;
	}
}

// Adds a person to the linked list
void AddPerson(struct List *list)
{
	// Allocate memory for the person
	struct Person *person;
	person = malloc(sizeof(struct Person));

	// Recieve user input
	printf("Enter name: ");
	fgets(person->name, 20, stdin);
	printf("Enter age: ");
	scanf("%d", &(person->age));

	strip_newline(person->name);

	// Assign next unused ID
	person->id = id_count;
	id_count++;

	ListInsert(list, person);
}

// Finds a person inside the linked list after asking for an ID
void FindPerson(struct List *list)
{
	// Get ID from the user
	int find_id;
	printf("Enter ID: ");
	scanf("%d", &find_id);

	ListFind(list, find_id);

	struct Person *person = ListGet(list);
```

```
	if (person)
		PrintPerson(person);
	else
		printf("Person with ID %d not found\n", find_id);
}

// Removes a person with the given ID
void RemovePerson(struct List *list)
{
	// Get ID from user
	int find_id;
	printf("Enter ID: ");
	scanf("%d", &find_id);

	ListFind(list, find_id);

	ListRemove(list);

}

void PrintList(struct List *list)
{
	ListHead(list);

	while(list->current)
	{
		PrintPerson(ListGet(list));
		ListNext(list);
	}
}


int main()
{
	struct List list;

	ListInitialize(&list);

	while(1)
	{
		int x = 0;
		print_menu();
		scanf("%d", &x);
		// Clear the buffer of newlines
		scanf("%*[^\n]");
		scanf("%*c");
		switch(x)
		{
```

```
                    case 1:
                        AddPerson(&list);
                        break;
                    case 2:
                        FindPerson(&list);
                        break;
                    case 3:
                        RemovePerson(&list);
                        break;
                    case 4:
                        PrintList(&list);
                        break;
                    case 5:
                        printf("Goodbye!\n\n");
                        return 0;
                    default:
                        printf("Invalid Option!\n\n");
                    }
                }
}
```

Assignment 3

Testing option 1

I should be able to add 3 people to the list, the expected results should be no errors

```
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 1
Enter name: Evan
Enter age: 23
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 1
Enter name: Phaedra
Enter age: 23
```

```
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 1
Enter name: Dan
Enter age: 22
```

Testing option 4
I should be able to print out all the elements I just added to the list

```
Select an option: 4

Person with ID 3:
        Name: Dan
        Age: 22


Person with ID 2:
        Name: Phaedra
        Age: 23


Person with ID 1:
        Name: Evan
        Age: 23
```

Testing option 2
The result should return the person with the specified ID

```
Select an option: 2
Enter ID: 2

Person with ID 2:
        Name: Phaedra
        Age: 23
```

Testing option 3
The program should remove the person with the specified ID from the list and that will be reflected in the print option

```
Select an option: 3
Enter ID: 3
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 4

Person with ID 2:
        Name: Phaedra
        Age: 23


Person with ID 1:
        Name: Evan
        Age: 23
```

Testing Option 5
Should exit the program

```
Select an option: 5
Goodbye!
```

Assignment 4

```
(gdb) b 215
Breakpoint 1 at 0x100000d38: file linkedlist.c, line 215.
(gdb) run
Starting program: /Users/evannoyes/Documents/spring16/robotics/labs/3/linkedlist

Breakpoint 1, main () at linkedlist.c:215
215                     int x = 0;
(gdb) c
Continuing.
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 1
Enter name: Evan
Enter age: 23

Breakpoint 1, main () at linkedlist.c:215
215                     int x = 0;
(gdb) print list;
Invalid character ';' in expression.
(gdb) print list
$1 = {head = 0x100102930, current = 0x100102930, previous = 0x0}
(gdb) print list.head
$2 = (struct Person *) 0x100102930
(gdb) print list.head->next
$3 = (struct Person *) 0x0
```

1. I was unable to set the breakpoint at the end of the while loop so I set it at the beginning and continued through the first trigger of it to get the intended program state
2. Calling `print list` returned a list structure representing the head location (the address Evan is stored at because it is the first element of the list), the current node being pointed too (also the address Evan is stored at because it is the most recently added element), and the the previous element in the list (which is NULL because there isn't a previous element)
3. Calling `print list.head` returned the pointer to the the head element of the list (the address Evan is stored at)
4. Calling `print list.head->next` returned the pointer to the next person in the list, but since there are no other elements this is set to NULL

Assignment 5

During the development of this lab we encountered seg faults when we did not properly set the addresses that were being accessed and once the program tries to access a null pointer there will be a crash since that memory isn't accessible by the program