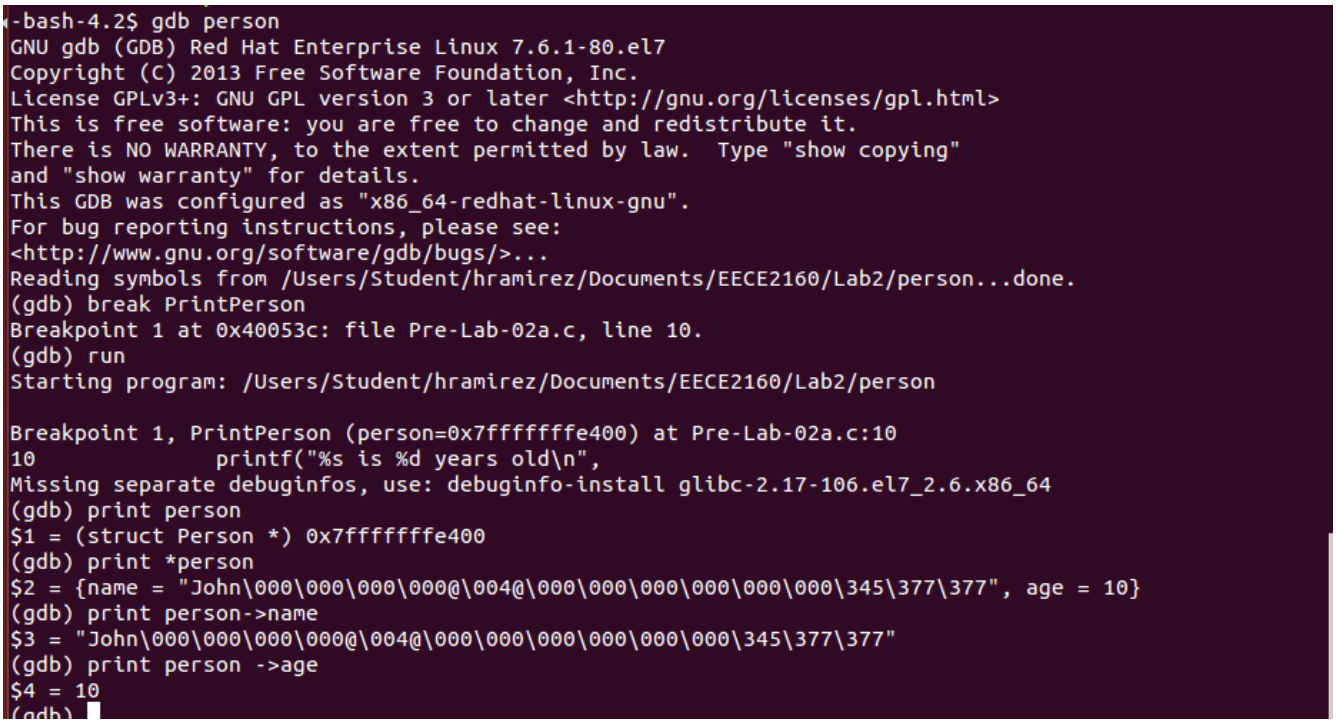
Lab 2

Hector Ramirez & Duy Vu

Assignment 1



print person: will give the data type of person, and its adress

print *person: dereferences the person and provides you with all the members in it's memory allocation

print person->name: will provide you with the name member of this object

print person->age: will provide you witht the age member of this object

Assignment 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// new
// Linked List Management Code
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;
    // Number of persons in the list
    int count;
};

//This id will be increment everytime a new person is added
int id;

// 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;
    list->count = 0;
}


// 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;
    list->count = list->count - 1;
}

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


// takes the list pointer and uses print person to print each person
void PrintList(struct List *list)
{
    //start the current form the head pointer
    ListHead(list);
    //move curent pointer through the list and print each Person link
    while (list->current) {
        PrintPerson(ListGet(list));
        ListNext(list);
    }
    //return the current pointer to the beginning
    ListHead(list);
}


//find the person in the linked list
void FindPerson(struct List *list) {
    int tempid;
    //get person id
    printf("Enter desierd id:");
    scanf("%d",&tempid);
    //point current pointer to the person using ListFind function
    ListFind(list,tempid);
    //print the person at the current pointer
    if (list->current) {
        PrintPerson(ListGet(list));
    }
    //if current is at null, print error not found to screen
    else {
        printf("Cannot find such ID");
```

```
    }
}


//remove th person
void RemovePerson(struct List *list) {
    int tempid;
    //get person id
    printf("Enter desierd id:");
    scanf("%d",&tempid);
    //point current pointer to the person need to be remove
    ListFind(list,tempid);
    //remove the person at the current pointer
    if (list->current) {
        ListRemove(list);
    }
    //print error not found if current pointing to null
    else {
        printf("Cannot find such ID");
    }
}


// Extra credit portion
//swap the position of the 2 person node
//this function is used for sorting the linked list using buble sort
void Swap(struct List *list) {
//make temp pointer for swaping
struct Person *temp;

//if the current is head, then we need to mark the next as head
//before swap the current head
if (list->current == list->head) {
list->head = list->current->next;
}

//current node is 1st, next of it is 2nd, and next of its next is 3rd
//0th is the node previous to current
//temp point to the next node (2nd node)
temp = list->current->next;
//point the current node 1st to the (3rd node)
list->current->next = list->current->next->next;
//connect the 0th node to the 2nd node
if (list->previous)
list->previous->next = temp;

//point the (2nd node) to the current node 1st
```

```
temp->next=list->current;
//point previous pointer to the prviously 2nd node
list->previous = temp;
}
```

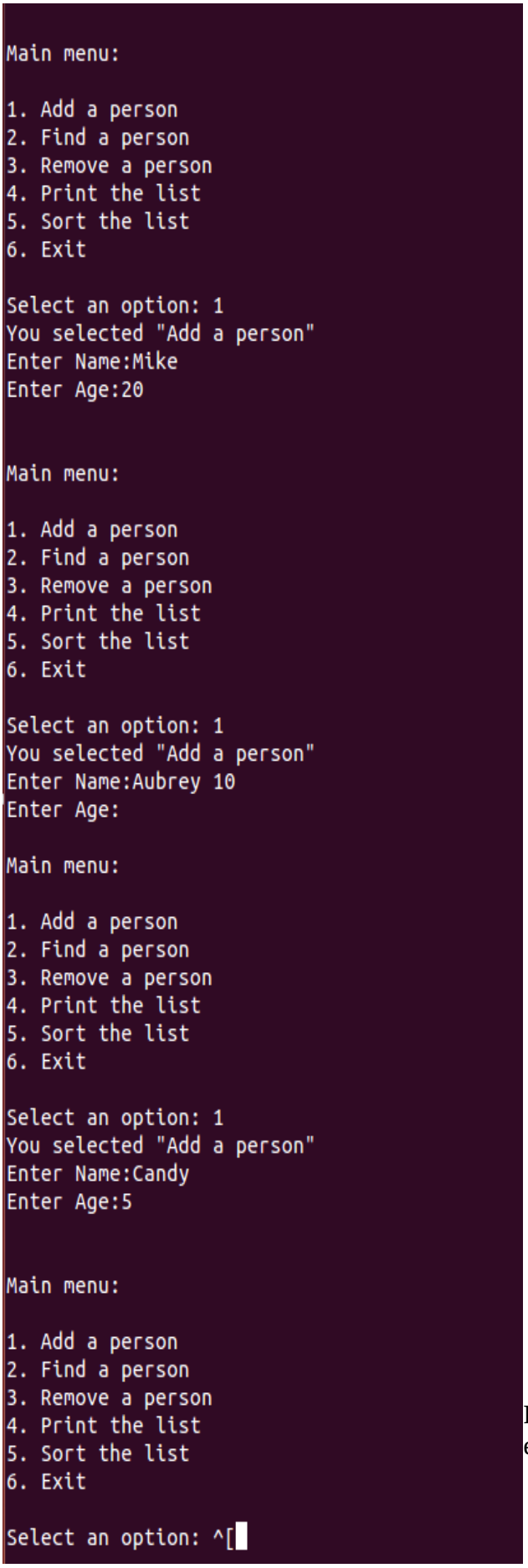
```
//bubble sort the linked list
void Sort(struct List *list) {
   int i,j,k;
        //promt for sort by name or age
   printf("\nPlease choose one of the following sorts\n");
   printf("1. name:\n");
   printf("2. age:\nOptions: ");
   scanf("%d",&i);

        //place current pointer at head
   ListHead(list);

        //bubble sort
   for (j=0;j<list->count-2;j++)
   {
      for (k=0;k<list->count-j-2;k++)
      {
                        //sort by name
         if (i == 1)
         {
            if (strcmp(list->current->name, list->current->next->name)>0)
            {
               Swap(list);
            }
         }
                        //sort by age
         else if (i == 2)
         {
            if (list->current->age > list->current->next->age)
            {
               Swap(list);
            }
         }
                        //no such type of sorting
         else
         {
            printf("please adhere to the options in the the menu");
            return ;
         }
```

```
        }
    }
}


//Add a person the the linked list
void AddPerson(struct List *list) {
    char tempname[20];
    int tempage;
    //promt for name and age
            printf("Enter Name:");
    scanf("%s",tempname);
    printf("Enter Age:");
    scanf("%d",&tempage);

            //create a new person
    struct Person *tempperson = (struct Person *)malloc(sizeof(struct Person));

            //adding info of the new person base on the provided inputs
    tempperson->next = NULL;
    tempperson->id=id;
    strcpy(tempperson->name,tempname);
    tempperson->age=tempage;

            //insert the person
    ListInsert(list, tempperson);

            //increment count
    id = id + 1;
            //adjusting the count of the list
    list->count = list->count + 1;
}


/** main function: Will create and process a linked list
 */
int main() {
    struct List list;                       // Create the main list
    ListInitialize(&list);                  // Initialize the list
//*************** PUT THE REST OF YOUR CODE HERE  *****************

    //using a string here to make sure user enter
    int option;
    id=1;
    //switch cases
    while (option != 6) {
        printf("\n\nMain 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. Sort the list\n");
        printf("6. Exit\n\n");
        printf("Select an option: ");

        //getting user's option input
        scanf("%d", &option);

        //switch cases
        switch (option) {
        case 1 :
           printf("You selected \"Add a person\"\n");
           AddPerson(&list);
           break;
        case 2 :
           printf("You selected \"Find a person\"\n");
           FindPerson(&list);
           break;
        case 3 :
           printf("You selected \"Remove a person\"\n");
           RemovePerson(&list);
           break;
        case 4 :
           printf("You selected \"Print the list\"\n");
           PrintList(&list);
           break;
        case 5:
           printf("You selected \"Sort the list\"\n");
           Sort(&list);
           break;
        case 6 :
           printf("Exiting...\n\n\n");
           //Finalize();
           break;
        default:
           printf("Invalid Option");
           while (getchar() != '\n');
           break;
        }

    }

    return 0;
} //end main
```

Assignment 3

```
You selected "Find a person"
Enter desierd id:3
Person with ID 3:
        Name: Candy
        Age: 5



Main menu:

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

Select an option: 3
You selected "Remove a person"
Enter desierd id:1


Main menu:

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

Select an option: 4
You selected "Print the list"
Person with ID 3:
        Name: Candy
        Age: 5

Person with ID 2:
        Name: Aubrey
        Age: 10
```



For Assignment 3, we believe we obtained the expected output for each of the different functions.

Assignment 4

```
Reading symbols from person...done.
(gdb) break 288
Breakpoint 1 at 0x400ddb: file Lab2.c, line 288.
(gdb) run
Starting program: /home/turtle/EECE2160/Lab2/person

Breakpoint 1, main () at Lab2.c:288
288             printf("\n\nMain menu:\n\n");
(gdb) c
Continuing.


Main menu:

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

Select an option: 1
You selected "Add a person"
Enter Name:he
Enter Age:10

Breakpoint 1, main () at Lab2.c:288
288             printf("\n\nMain menu:\n\n");
(gdb) print list
$1 = {head = 0x603010, current = 0x603010, previous = 0x0, count = 1}
(gdb) print list.head
$2 = (struct Person *) 0x603010
(gdb) print list.head->next
$3 = (struct Person *) 0x0
(gdb)
```

Only one element was added to this list. The print list statement would provide all the members of the object list and their contents. The print list.head would provide the data type of the specific member head and it's contents. We can also call something from the element to which head is pointing to by using the → operatior. In this case print list.head->next points to the element after the head element; which happens to be null due to the fact that there exists only one element.

Assignment 5

```
Main menu:

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

Select an option: 1
You selected "Add a person"
Enter Name:e
Enter Age:10


Main menu:

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

Select an option: 4
You selected "Print the list"
Person with ID -1992206527:
        Name: ◆AVI◆◆AUI◆◆ATL◆%
        Age: 764233813


Program received signal SIGSEGV, Segmentation fault.
0x00000000004009b4 in PrintPerson ()
(gdb) backtrace
#0  0x00000000004009b4 in PrintPerson ()
#1  0x0000000000400a29 in PrintList ()
#2  0x0000000000400ec3 in main ()
(gdb) print list
$1 = list
(gdb) print list.head
Attempt to extract a component of a value that is not a structure.
(gdb) print person
No symbol "person" in current context.
(gdb) print list
$2 = list
(gdb) print list
$3 = list
```

```
int main() {
   struct List list;                    // Create the main list
   ListInitialize(&list + 1);           // Initialize the list
   +1 add one to the address of list so that the list being past is a wrong memory address
…
}
```

Therefore when adding a person the code will properly place the person. But due to the fact that we purposely misplaced the location of the list the output of the print option provided garbage from unwanted memory locations.
The back trace explained that the error comes from a seg fault in print person function.
print list commands returns no data type for list variable prove that the print function is not receiving the correct data types.

Extra Credit

```
Main menu:

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

Select an option: 4
You selected "Print the list"
Person with ID 4:
        Name: A
        Age: 4
Main menu:

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

Select an option: 5
You selected "Sort the list"

Please choose one of the following sorts
1. name:
2. age:
Options: 2


Main menu:

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

Select an option: 4
You selected "Print the list"
Person with ID 1:
        Name: D
        Age: 1

Person with ID 2:
        Name: C
        Age: 2

Person with ID 3:
        Name: B
        Age: 3

Person with ID 4:
        Name: A
        Age: 4
```

```
Main menu:

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

Select an option: 5
You selected "Sort the list"

Please choose one of the following sorts
1. name:
2. age:
Options: 1


Main menu:

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

Select an option: 4
You selected "Print the list"
Person with ID 4:
        Name: A
        Age: 4

Person with ID 3:
        Name: B
        Age: 3

Person with ID 2:
        Name: C
        Age: 2

Person with ID 1:
        Name: D
        Age: 1
```