Advanced Answer to Homework 8 (Reusable)

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/*      FILENAME:       Advanced Version of HM 8                       */
/*      REVISION:       1.0                                            */
/*      REVISION DATE:  11/22/92                                       */
/*                                                                     */
/*      FUNCTIONAL DESCRIPTION: This is an advanced template for       */
/*                              homework 8                             */
/*                                                                     */
/*---------------------------------------------------------------------*/

#include <stdio.h>
#include

/* define employee structure                                          */
typedef struct employee
    {
    char        name[20];
    long        id_number;
    float       wage;
    float       hours;
    float       overtime;
    float       gross;
    }  employee;

/* define linked list of employees                                    */
typedef struct linked_list_node
    {
    struct employee emp;
    struct linked_list_node *next;
    }  linked_list_node;

/*---------------------------------------------------------------------*/
/*                                                                     */
/* FUNCTION:  create_node                                              */
/*                                                                     */
/* DESCRIPTION:  This function will create a node for the linked list. */
/*                                                                     */
/* PARAMETERS:   None                                                  */
/*                                                                     */
/* OUTPUTS:      A new node for the linked list                        */
/*                                                                     */
/*---------------------------------------------------------------------*/
linked_list_node * create_node()
    {
    return (linked_list_node *) malloc(sizeof(struct linked_list_node));
    }

/*---------------------------------------------------------------------*/
/*                                                                     */
/* FUNCTION:  is_empty                                                 */
/*                                                                     */
/* DESCRIPTION:  This function will determine if a linked list is empty.*/
/*               If the front of the list is NULL, then the list does  */
/*               not contain any nodes.                                */
/*                                                                     */
/* PARAMETERS:   Pointer to the front of the linked list               */
/*                                                                     */
```

```
/* OUTPUTS:      Returns 0 or 1, where a value of 0 indicates the list  */
/*               is not empty and 1 indicates the linked list is empty. */
/*                                                                      */
/* CALLS:        None                                                   */
/*                                                                      */
/*----------------------------------------------------------------------*/
int is_empty(front_of_list)
linked_list_node *front_of_list;
    {
    if(front_of_list == NULL)
        {
        return 1;
        }
    return 0;
    }

/*----------------------------------------------------------------------*/
/*                                                                      */
/* FUNCTION:  find_back_of_list                                         */
/*                                                                      */
/* DESCRIPTION:  This function will traverse a linked list until it     */
/*               encounters the end of the linked list.                 */
/*                                                                      */
/* PARAMETERS:   A pointer to the front of the list                     */
/*                                                                      */
/* OUTPUTS:      A pointer the end of the linked list.                  */
/*                                                                      */
/* CALLS:        is_empty                                               */
/*                                                                      */
/*----------------------------------------------------------------------*/
linked_list_node * find_back_of_list(front_of_list)
linked_list_node *front_of_list;
    {
    linked_list_node *list_ptr;

    /* If the list is empty, you are already at the end.  Simply return */
    /* a null value                                                     */

    if(is_empty(front_of_list))
        {
        return NULL;
        }

    /* Otherwise, traverse the list until you reach the end             */

    for(list_ptr = front_of_list; list_ptr->next != NULL;
      list_ptr = list_ptr->next)
        /* DO NOTHING */ ;

    /* Return pointer to the last item in the list                      */

    return list_ptr;
    }

/*----------------------------------------------------------------------*/
/*                                                                      */
/* FUNCTION:  add_node_to_end_of_list                                   */
/*                                                                      */
/* DESCRIPTION:  This function will add a node to the end of a linked   */
/*               list.                                                  */
/*                                                                      */
```

```
/* PARAMETERS:   Pointer to Pointer to front of list                    */
/*               a Pointer to a node to add to the list                 */
/*                                                                      */
/* OUTPUTS:      None.                                                  */
/*                                                                      */
/* CALLS:        is_empty                                               */
/*               find_back_of_list                                      */
/*                                                                      */
/*----------------------------------------------------------------------*/
void  add_node_to_end_of_list(front_of_list, list_node)
linked_list_node **front_of_list;
linked_list_node *list_node;
    {
    linked_list_node *temp_ptr;

    /* Check to see if the list is empty                                */

    if(is_empty(*front_of_list))
        {
        /* Change the front of the list to point to the new             */
        /* node (list_node)                                             */

        /* Since the front_of_list pointer to the address of a          */
        /* pointer to a linked_list_node, we use *front_of_list         */
        /* to change the value of what front of list points to          */

        *front_of_list = list_node;
        }
    else
        {
        /* Link the new node (list_node) at the back of the list        */

        temp_ptr = find_back_of_list(*front_of_list);
        temp_ptr->next = list_node;
        }

    /* make the back of the list (list_node) point to NULL (terminate   */
    /* the list                                                         */

    list_node->next = NULL;
    return;
    }

/*----------------------------------------------------------------------*/
/*                                                                      */
/* FUNCTION:  print_list                                                */
/*                                                                      */
/* DESCRIPTION:  This function will print the contents of a linked      */
/*               list.  It will traverse the list from beginning to the */
/*               end, printing the contents at each node.               */
/*                                                                      */
/* PARAMETERS:   linked_list_node *emp1 - pointer to a linked list      */
/*                                                                      */
/* OUTPUTS:      None                                                   */
/*                                                                      */
/* CALLS:        None                                                   */
/*                                                                      */
/*----------------------------------------------------------------------*/
void print_list(emp1)
linked_list_node *emp1;
    {
```

```
    linked_list_node *tmp; /* tmp pointer value to current node */

    /* print your header or call a routine to print the header */

    /* Start a beginning of list and print out each value              */
    /* loop until tmp points to null (remember null is 0 or false)    */
    for(tmp = emp1; tmp ; tmp = tmp->next)
        {
        /* print the members at each node */
        }
    }

/*--------------------------------------------------------------------*/
/*                                                                    */
/* FUNCTION:  read_hours                                              */
/*                                                                    */
/* DESCRIPTION:  This function will read in the hours, determine the  */
/*               overtime hours and gross pay.                        */
/*                                                                    */
/* PARAMETERS:   linked_list_node *emp1 - pointer to linked list      */
/*                                                                    */
/* OUTPUTS:      None                                                 */
/*                                                                    */
/* CALLS:        None                                                 */
/*                                                                    */
/*--------------------------------------------------------------------*/
void read_hours(emp1)
linked_list_node *emp1;
    {
    linked_list_node *tmp;

    /* Start a beginning of list and read in hour values              */
    /* for each employee                                              */

    /* Loop:  read hours, calc ot hours, calc gross pay               */
    /*        you may want to call functions to do this               */
    }

/*--------------------------------------------------------------------*/
/*                                                                    */
/* FUNCTION:  add_employees                                           */
/*                                                                    */
/* DESCRIPTION:  This function will add employee information to a linked*/
/*               list until the user indicates he/she is done.        */
/*                                                                    */
/* PARAMETERS:   linked_list_node *emp1 - pointer to a linked list    */
/*                                                                    */
/* OUTPUT:       new_emp_cnt - number of new employees added to list  */
/*                                                                    */
/* CALLS:        add_node_to_end_of_list                              */
/*               create_node                                          */
/*                                                                    */
/*--------------------------------------------------------------------*/
int add_employees(emp1_list)
linked_list_node **emp1_list;
    {
    int new_emp_cnt = 0;              /* count of new employees       */
    char read_value[BUFSIZ];          /* temporary input buffer       */
    linked_list_node *new_employee;   /* pointer to linked list       */

    do
```

```
        {
        /* create a new node for the linked list                          */
        if( !(new_employee = create_node()) )
            {
            fprintf(stderr,"Ran-out of dynamic memory\n");
            exit(1);
            }

        /* Read in Employee Name, ID, and Wage                            */

        printf("Enter Employee Name: ");
        gets(new_employee->emp.name);

        printf("Enter Employee ID: ");
        gets(read_value);
        sscanf(read_value,"%ld", &new_employee->emp.id_number);

        printf("Enter Employee Wage: ");
        gets(read_value);
        sscanf(read_value,"%f", &new_employee->emp.wage);

        /* Add to linked list                                             */

        printf("add employee\n");

        /* since emp1_list is a pointer to a pointer to a linked list   */
        /* (emp1_list points to the address of the linked list) the     */
        /* just pass emp1_list list since add_node_to_end_of_list       */
        /* will modify the contents of what emp1_list points            */
        /* to not the actual pointer (emp1_list)                        */

        add_node_to_end_of_list(emp1_list,new_employee);

        /* Update new employee count                                      */
        ++new_emp_cnt;

        /* Does user wish to add another employee to the list           */

        printf("\n\nAdd another employee (y/n): ");
        gets(read_value);

        } while (read_value[0] == 'y'  || read_value[0] == 'Y');

    /* return of the count of new employees that were added to the list */
    return new_emp_cnt;
    }

/* Declare and intialize the linked list for employees                  */

linked_list_node * employee_list = NULL;

main ()
    {
    /* Initialize variables                                           */
    int employee_count = 0;    /* total number of employees           */
    int new_employees = 0;     /* total new employees added           */

    /* pass the address of employee_list since the function will change */
    /* the value of employee_list (ie: it will now point to list of new */
    /* employees)                                                     */
```

```
new_employees = add_employees(&employee_list);

/* just pass employee list since read_hours will modify the         */
/* contents of what employee_list points to, not the actual pointer */

read_hours(employee_list);

print_list(employee_list);

printf("\n\n***End of Program\n");
}
```