CODE

Main.c code:

```
#include<stdio.h>

#include<stdint.h>

#include"project_1.h"

#include"memory.h"

void project_1_report();

int main()

{

project_1_report();

return 0;

}
```

Project_1 code:

```
#include<stdio.h>
#include<stdint.h>
#include"memory.h"
#include"project_1.h"
int8_t my_memmove(uint8_t *src, uint8_t *dst, uint32_t length);
int8_t my_memzero(uint8_t *src, uint32_t length);
//Function definition to    zero out given length of bytes in the memory
//Function definition to move data from one location to other location

void project_1_report()
{

   uint8_t array[32];

   uint8_t *aptr_1=NULL, *aptr_2=NULL , *aptr_3= NULL, data=31;
   uint32_t i=0;

   aptr_1=(array);
   aptr_2=(array+8);
   aptr_3=(array+16);

   for(i=0;i<=15;i++)
   {
   *(aptr_1 + i)=data;
   data++;
   }
```

```
   my_memzero(aptr_3, 16);

   my_memmove(aptr_1, aptr_3, 8);

   my_memmove(aptr_2, aptr_1, 16);

   my_reverse(aptr_1, 32);

   for(i=0;i<=31;i++)
   {
   printf("The value in byte %d is:%d \n",i,*(array+i));
   }

}
```

Memory.c code:

```
#include<stdio.h>
#include<stdint.h>
#include"memory.h"


int8_t my_memmove(uint8_t *src, uint8_t *dst, uint32_t length)                    //Function
definition to move data from one location to other location
{
   uint32_t i=0;
   uint8_t temp[50];

   if(src)
   {



      while(i<length)                                          //Copies data from source  to temp array
      {
         *(temp+i)=*(src+i);
         i++;
      }
      i=0;

      while(i<length)                                           //Copies data from temp to destination
array
      {
         *(dst+i)=*(temp+i);
         i++;
      }
```

```
    i=0;



  }

  else
  {

    printf("Pointer ERROR");                                    //Displays an error if move failed
    return 1;
  }



}

int8_t my_memzero(uint8_t *src, uint32_t length)                          //Function definition to
zero out given length of bytes in the memory
{

  uint16_t i=0,len=0;
  while(*(src+i)!='\0')
  {
    len++;                                                  //Calculates the total length of the memory
    i++;
  }

  i=0;

  if(src)
  {
    while(i<length)
    {
      *(src+i)=0;                                           //Clears the memory i.e sets it to zero for
given length of bytes
      i++;
    }

    i=0;



    return 0;
```

```
    }

    else
    {
        printf("Pointer Error");                                    //Displays an error if function fails
        return 1;
    }


}

int8_t my_reverse(uint8_t *src, uint32_t length)                    //Function definition
for reverse function
{
    uint16_t i=0,counter;

    if(src)
    {
        int8_t temp=0;
        for (counter=0;counter<(length/2);counter++)
        {
            temp=*(src+counter);                                    //Performs the reverse operation
using a temporary variable
            *(src+counter)=*(src+length-counter-1);
            *(src+length-counter-1)=temp;
        }


        return 0;
    }

    else
    {
        return 1;
    }

}
```

Data.c code:

```
#include<stdint.h>
#include<stdio.h>
#include"data.h"

int32_t my_atoi(int8_t *str) ;                  //Function definition for ascii string to integer function
void dump_memory(uint8_t *start, uint32_t length) ;
void reverse(int8_t *str, int32_t length);      //Function declaration for reverse
uint32_t big_to_little(uint32_t data);
```

```
uint32_t little_to_big(uint32_t data);
int8_t * itoa(int32_t num, int8_t *str, int32_t base);


int32_t my_atoi(int8_t *str)                    //Function definition for ascii string to integer function
{
  printf("string:%s\n",str);                    //Prints the input string
  int32_t i=0;
  int32_t length=0;                             //Initialising length of the string to zero
  while(*(str+i)!='\0')                         //Calculate the length of the string by incrementing length
till the string reaches the null character
  {
     length++;
     i++;
  }

  printf("length is: %d\n",length);                //Displaying the length of the string
  for(i=0;i<length;i++)                         //Converts the ascii character of each character in the string
to its corresponding integer value
  {
     printf("%d",*(str+i));                     //Displays the integer value of the input string
  }
  return 0;
}

void dump_memory(uint8_t *start, uint32_t length)       //Function definition for printing hex
output of the data bytes in the memory given pointer to a memory location & length of bytes to
print
{
  int32_t i;
  for(i=0;i<length;i++)
  {
    printf("\n Hex output: %x\t",*(start+i));        //Prints the hex value of number of bytes given in
the length


  }

}

uint32_t big_to_little(uint32_t data)               //Function definition for converting data from big
endian to little endian
{
  int32_t z = 1;                                //Initialising the value of z to one to check endianess
  int8_t *y = (int8_t*)&z;                      //To check the byte in lower memory address
  printf("The value in  lower memory is:%c\n",*y+48);   //Displays the value in lower memory
address
```

```
  if((*y+48)=='0')                          //If the byte in lower memory address is zero, then it is big
endian hence convert it  to little endian
  {
     data = ( data >> 24 ) | (( data << 8) & 0x00ff0000 )| ((data >> 8) & 0x0000ff00) | ( data << 24)  ;
//Bitwise shifting for the conversion
     printf("Little endian data is  = %x\n", data);    //Displays the converted little endian data
  }

  else
     printf("The data is already stored as little endian\n");
  return 0;
}

uint32_t little_to_big(uint32_t data)               //Function definition for converting data from little
endian to big endian
{
  int32_t z = 1;                            //Initialising the value of z to one to check endianess
  int8_t *y = (int8_t*)&z;                        //To check the byte in lower memory address
  printf("The value in  lower memory is:%c\n",*y+48);   //Displays the value in lower memory
address

  if((*y+48)=='1')                          //If the byte in lower memory address is one, then it is little
endian hence convert it to big endian
  {

     data = ( data >> 24 ) | (( data << 8) & 0x00ff0000 )| ((data >> 8) & 0x0000ff00) | ( data << 24)  ;
//Bitwise shifting for the conversion
     printf("Big endian data is = %x\n", data);        //Displays the converted big endian data
  }

  else
     printf("The data is already stored as big endian\n");
  return 0;
}

int8_t * itoa(int32_t num, int8_t *str, int32_t base)     //Function definition for converting data from
integer to ascii string
{
   int32_t i = 0;
   int32_t neg=0;

                                        // Handle 0 explicitely, otherwise empty string is printed for 0
   printf("the number is:%d\n",num);
   if (num ==0)
   {
      *str='0';
      i++;
                    *(str+i)='\0';
```

```
    printf("The string is:%s",str);
    return str;
  }

  if(num<0 && base==16)
  {
    sprintf(str,"%X",num);                              // convert decimal to hexadecimal
    printf("converting %d to hexadecimal notation %s\n",num,str);    //shows the hex output for
signed integer
    return str;
  }

  if(num<0 && base==8)
  {
    sprintf(str,"%o",num);                             //convert decimal to octal
    printf("converting %d to octal notation %s\n",num,str);       //shows the octal output of signed
integer
    return str;
  }

  if (num < 0 && (base == 10 || base==2))
  {
    neg = 1;                                          //Set neg varibale to 1 if the number is negative
    num = -num;                                        //Consider only unsigned number initially for
conversion
  }

                                                    // Process individual digits
  while (num != 0)
  {
    int32_t rem = num % base;
    *(str+i)= (rem > 9)?(rem-10) + 'a' : rem + '0';            //Converting integer to ascii string
    i++;
    num = num/base;
  }

  if (neg==1)                                         //If number is negative, append '-'
  {
    *(str+i)= '-';
    i++;
  }

  *(str+i)= '\0';                                     // Append string terminator
  reverse(str, i);                                     // Reverse the string
  printf("The string is:%s",str);                             //Print the ascii string

  return str;
}
```

```
void reverse(int8_t *str, int32_t length)                    //Function to perform reverse of the string
{
   int32_t start = 0;
   int32_t end = length -1;
   while (start < end)                                       //Swap the string
   {
      int8_t temp= *(str+start);
      *(str+start)= *(str+end);
      *(str+end)=temp;
      start++;
      end--;
   }
}
```