Documentation of all exercises of Day_5

Exercise 3-4 :

Question: In a two's complement number representation, our version of itoa does not handle the largest negative number, that is, the value of n equal to -(2 wordsize-1). Explain why not. Modify it to print that value correctly, regardless of the machine on which it runs.

The given itoa function to modify is as follows:

```
/* itoa:  convert n to characters in s */
void itoa(int n, char s[])
{
    int i, sign;

    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {       /* generate digits in reverse order */
        s[i++] = n % 10 + '0';  /* get next digit */
    } while ((n /= 10) > 0);    /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

The code provided for itoa does not handle the largest negative number correctly because it relies on the assumption that the input integer n is a non-negative value. In two's complement representation, the largest negative number does not have a corresponding positive value. This is because the range of representable integers in two's complement is asymmetric, with the negative range having one more value than the positive range.

In two's complement representation, the largest negative number, represented as -(2^(wordsize-1)), cannot be simply negated to obtain its positive counterpart. When this negative number is negated, it remains the same value, as there is no positive equivalent within the range of representable integers.

According to the above program the expression like n = -n is not enough to get the right answer. Because for the smallest negative number the value of n and –n remains the same as it does not have any positive equivalent. To overcome this limitation I have taken the absolute value of n%10 in the do while loop body statement.

The modified code is given below:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#define MAXLEN 1000

void reverse(char s1[], int index, int size) {
    char temp;
    temp = s1[index];
    s1[index] = s1[size - index];
    s1[size - index] = temp;

    if (index == size / 2) {
        return;
    }
    reverse(s1, index + 1, size);
}

void int_to_array(int n, char s[]) {
    int i, sign;
    sign = n;
    i = 0;
    do {
        s[i++] = abs(n % 10) + '0';
    } while (n /= 10);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    int str_length = strlen(s);
    reverse(s, 0, str_length-1);
}

int main(void) {
    int n;
    printf("Enter the integer to convert: ");
    scanf("%d", &n);
    char s[MAXLEN];
    int_to_array(n, s);
    printf("The Converted String from Integer is: %s\n", s);
return 0;
}
```

The output for the smallest negative integer according to the above program is as follows:

```
Enter the integer to convert: -2147483648
The Converted String from Integer is: -2147483648

--------------------------------
Process exited after 6.262 seconds with return value 0
Press any key to continue . . .
```

We can see that the program gives output correctly in terms of smallest negative number input.

Exercise 3-5:

Question: Write the function itob(n,s,b) that converts the integer n into a base b character representation in the string s. In particular, itob(n,s,16) formats s as a hexadecimal integer in s.

The driver code of the itob function that converts an integer n to a base b character representation in the string s is as follows:

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

#define MAXLEN 100

void itob(int n, char s[], int b) {
    int i, sign;
    sign = n;
    i = 0;
    do {
        int digit = abs(n%b);
        if(digit<10)
            s[i++] = digit + '0';
        else
            s[i++] = digit - 10 + 'A';
    } while (n /= b);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    int str_length = strlen(s);
    reverse_str(s, 0, str_length-1);
}

void reverse_str(char str1[], int index, int size)
{
  char temp;

  temp = str1[index];
  str1[index] = str1[size - index];
  str1[size - index] = temp;

  if (index == size / 2)
  {
    return;
  }

  reverse_str(str1, index + 1, size);
}

int main(void)
{
    int n, b;
    printf("Enter the value to convert: \n");
    scanf("%d", &n);
    printf("Enter the base: \n");
    scanf("%d", &b);
    char s[MAXLEN];
    itob(n, s, b);
    printf("The Converted Number is: %s\n", s);
}
```

In this code, the itob function takes three arguments: the integer n to convert, the string s to store the resulting character representation, and the base b to use for the conversion.

The function first stores n in the sign variable to check whether the number is signed or not in later.

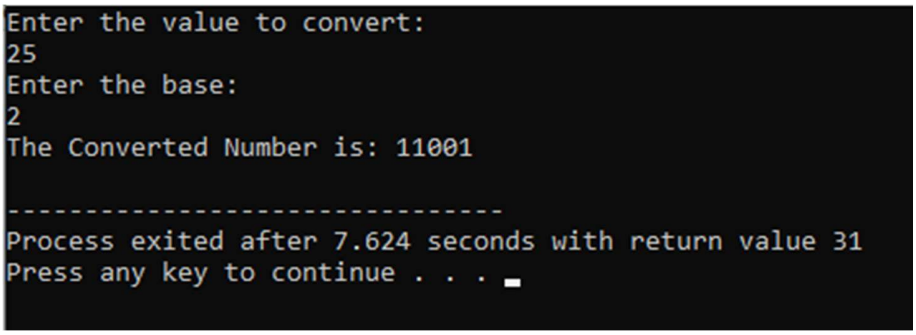
Then, a do-while loop is used to repeatedly extract the least significant digit by taking the modulus of n with b and the absolute value of the modulus of n with b is taken to handle the smallest negative integer value. If the digit is less than 10, it is converted to the corresponding character by adding '0'. If the digit is greater than or equal to 10, it is converted to the corresponding hexadecimal character by subtracting 10 and adding 'A'.

After extracting all the digits, the sign is added to the string if necessary. The string is then terminated with '\0', and the reverse function is called to reverse the string, ensuring that it is in the correct order.

From the main function the function itoa is called with necessary parameters. The next snapshot is the output for this program.

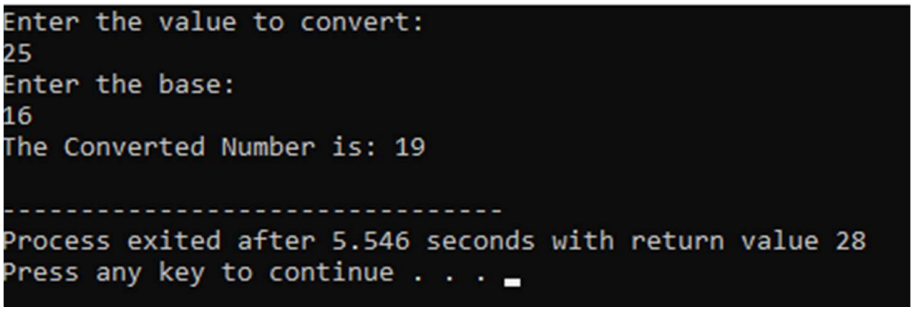
Output:

i)



In this snapshot we can see that the output is 11001 for the input of 25. The input is an integer number and the base given here is 2. For this reason converted output number is a binary representation of 25.

ii)



In this snapshot we can see that the output is 19 for the input of 25. The input is an integer number and the base given here is 16. For this reason converted output number is a hexadecimal representation of 25.

Exercise 3-6.

Question: Write a version of itoa that accepts three arguments instead of two. The third argument is a minimum field width; the converted number must be padded with blanks on the left if necessary to make it wide enough.

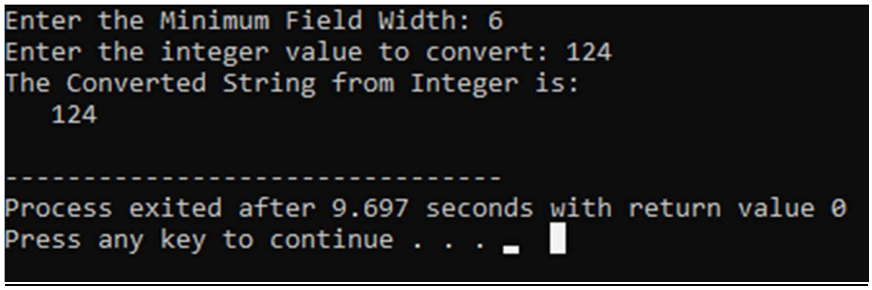
The updated version of the itoa function that accepts three arguments: the integer n to convert, the string s to store the resulting character representation, and the minimum field width minWidth. The driver code for this updated itoa function is given below:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
#define MAXLEN 1000
void reverse(char s1[], int index, int size) {
    char temp;
    temp = s1[index];
    s1[index] = s1[size - index];
    s1[size - index] = temp;

    if (index == size / 2) {
        return;
    }
    reverse(s1, index + 1, size);
}

void i_to_a(int n, char s[], int min_fwidth) {
    int i, sign;
    sign = n;
    i = 0;
    do {
        s[i++] = abs(n % 10) + '0';
    } while (n /= 10);
    if (sign < 0)
        s[i++] = '-';

    while(i<min_fwidth)
        s[i++] = ' ';
    s[i] = '\0';
    int str_length = strlen(s);
    reverse(s, 0, str_length-1);
}

int main(void) {
    int n, min_field_width;
    printf("Enter the Minimum Field Width: ");
    scanf("%d", &min_field_width);
    printf("Enter the integer value to convert: ");
    scanf("%d", &n);
    char s[MAXLEN];
    i_to_a(n, s, min_field_width);
    printf("The Converted String from Integer is:\n%s\n", s);

return 0;
}
```

The minimum field width required for the converted number is represented by the min_fWidth argument, which is now a part of the itoa function.

The function determines if the converted number's width is smaller than the min_fWidth value after generating the character representation of the number as previously. If so, it inserts spaces (' ') to increase the string's left side until the necessary minimum field width is met.

Output:



In the above output window we can see that the field with is given 6 but the inputted number is of 3 digit. The converted string output will remain same as the inputted number but it will be padded with three empty spaces to match with the field width as the minimum input field width is 6.

Chapter Question 4:

1.function fun()try to store a string to backward sequence, fill in the blanks

```
void fun (char str[])
{ char m; int i,j;
  for(i=0,j=strlen(str);i<__________;i++,j--)
  { m=str[i];
    str[i]=__________;
    str[j-1]=m;
  }
  printf("%s\n",str);
}
```

This is a string reverse function that store a string backward in backward sequence. After fill in the blanks the function will be like this:

```
void fun (char str[])
{ char m; int i,j;
  for(i=0,j=strlen(str);i<strlen(str)/2;i++,j--)
  { m=str[i];
    str[i]=str[j-1];
    str[j-1]=m;
  }
```

```
    printf("%s\n",str);
}           printf("%s\n",str);
}
```

I have also implemented the function fun() in my IDE and checked the output of this function. Here is the snapshot of this function fun():

chapter 4.c

```c
#include <stdio.h>
#define size 100
void fun (char str[])
      { char m; int i,j;
        for(i=0,j=strlen(str);i<strlen(str)/2;i++,j--)
        { m=str[i];
          str[i]=str[j-1];
          str[j-1]=m;
        }
        printf("%s\n",str);
      }

int main()
{
    char str[size];
    printf("Enter the strings: ");
    scanf(" %[^\n]", str);
    printf("The reversed string is: ");
    fun(str);
}

```

Output:

```
D:\Chapter Practise\chapter 4.exe
Enter the strings: Hello world
The reversed string is: dlrow olleH

--------------------------------
Process exited after 4.709 seconds with return value 0
Press any key to continue . . .
```