C programming
5613

Lecture 7

Marina Krstic Marinkovic
mmarina@maths.tcd.ie

School of Mathematics
Trinity College Dublin

Timing the execution of your code

```c
//Piece of code for measuring the time spent in a calling process

#include <time.h>

clock_t start, end;
double cpu_time_used;

start = (double)clock();
... /* Do the work. */
end = (double)clock();
cpu_time_used = (end - start) / (double)CLOCKS_PER_SEC;
```

- CPU time consumed by the **calling process** between the two calls
- **clock_t clock(void) -** values of type **clock_t** are numbers of clock ticks
- **int CLOCKS_PER_SEC** - number of clock ticks per second measured by the **clock()** function
- Main CPU-Time page: https://www.gnu.org/savannah-checkouts/gnu/libc/manual/html_node/CPU-Time.html

Timing the execution of insertion sort

```c
/*Timing the insertion sort algorithm*/
#include<stdio.h>
#include<time.h>
int main()
{
    int data[100],n,temp,i,j;
    double t1,t2,dt;

    printf("Enter number of terms(should be less than 100): ");
    scanf("%d",&n);
    printf("Enter elements: ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&data[i]);
    }
    t1=(double)clock();
    for(i=1;i<n;i++)
    {
        temp = data[i];
        j=i-1;
        while(temp<data[j] && j>=0)
        /*To sort elements in descending order, change temp<data[j]
to temp>data[j] in above line.*/
        {
            data[j+1] = data[j];
            --j;
        }
        data[j+1]=temp;
    }
    t2=(double)clock();
    dt=(t2-t1)/(double)(CLOCKS_PER_SEC);

    printf("In ascending order: ");
    for(i=0; i<n; i++)
        printf("%d\t",data[i]);
    printf("\nTime needed to sort the array: %.2e sec\n",dt);
    return 0;
```

Data I/O from/to files in C

```c
//File type
FILE *fptr1, *fptr2, *fptr;

//Opening a file for creation/edit:
fptr = fopen("fileopen","mode")

//Creating a new file
fptr1=fopen("oldinput.txt","r");

//Opening an existing file
fptr1=fopen("newinput.txt","wa");

//Closing a file
fclose(fptr1);
fclose(fptr2);
fclose(fptr);
```

```c
//Reading information from a text file
fscanf(fptr,"%d",&num);
fscanf(fptr,"%d\n",&num);

//Writing information to a text file
int num;
double entry;
fprintf(fptr,"%d",num);
fprintf(fptr,"%.4f",entry);

//Reading information from a binary file
threeNum num;
fread(&num, sizeof(struct threeNum), 1, *fptr);

//Writing information to a binary file
threeNum num;
fwrite(&num, sizeof(struct threeNum), 1, *fptr);
```

- Two most commonly used types of files: text(.txt) and binary(.bin) files
- .txt - intuitive, easy to read and edit/delete
- .bin - can hold higher amount of data, not easily readable and provide a better security than text files

Data I/O from/to files in C

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exists, it will be created.

Sort an array read from a file

```c
/* Sorting Elements of an array in ascending order using insertion sort algorithm*/
/* The elements of an array are read from a file, where the first. */
#include<stdio.h>
#include<time.h>
#include<stdio.h>
#include<time.h>
void insertsort(int *arr,int n)
{
    int i,j,temp;
    for(i=1;i<n;i++)
    {
        temp = arr[i];
        j=i-1;
        while(temp<arr[j] && j>=0)
        {
            arr[j+1] = arr[j];
            j-;
        }
        arr[j+1]=temp;
    }
}
int main()
{
    int data[10000],n,temp,i,j;
    double t1,t2,dt;
    FILE *fptr1,*fptr2;

    fptr1 = fopen("array.txt","r");
    fptr2 = fopen("sorted.txt","w");
    fscanf(fptr1,"%d",&n); //Reading number of terms in an array
    if (n>10000) //Maximal size of an array is 10000
        printf("The number of elements cannot be larger than 10000. Please modify the input file");

    for(i=0;i<n;i++) //proceeding to read the elements of an array
    {
        fscanf(fptr1,"%d",&data[i]);
    }
    t1=(double)clock();
    insertsort(data,n);
    t2=(double)clock();
    dt=(t2-t1)/(double)(CLOCKS_PER_SEC);

    for(i=0; i<n; i++)
        fprintf(fptr2,"%d\n",data[i]);
    fprintf(fptr2,"\nTime needed to sort the array: %.2e sec\n",dt);

    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```

Sorting the array: quick sort

- “divide-and-conquer” strategy
- **Choose a pivot value:** e.g. the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array
- **Partition:** Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice: the array may be divided in non-equal parts
- **Sort both parts:** Apply quick sort algorithm recursively to the left and the right parts

Quick sort vs. insertion sort

```c
void quicksort(int *arr, int low, int high)
{
    int pivot, i, j, temp;
    if(low < high) {
        pivot = low; // select a pivot element
        i = low;
        j = high;
        while(i < j) // increment i until you get a number > than the pivot element
        {
            while(arr[i] <= arr[pivot] && i <= high) //decrement j until getting a number < pivot element
                i++;
            while(arr[j] > arr[pivot] && j >= low)
                j--;
            // if i < j swap the elements in locations i and j
            if(i < j)
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        // when i >= j it means the j-th position is the correct position
        // of the pivot element, hence swap the pivot element with the
        // element in the j-th position
        temp = arr[j];
        arr[j] = arr[pivot];
        arr[pivot] = temp;

        quicksort(arr, low, j-1);   // Repeat quicksort for the two sub-arrays, one to the left of j
        quicksort(arr, j+1, high);  // and one to the right of j
    }
}
```

- Compare the execution times between quick and insertion sort
- For: n=10,100,1000,10000

Example from Lecture 6:

```c
#include <stdio.h>
void bubblesort(char string[],int n)
{
  int step,i;
  char temp;

    for(step=0;step<n-1;step++)
        for(i=0;i<n-step-1;i++)
        {
            if(string[i]>string[i+1])
            {
                temp=string[i];
                string[i]=string[i+1];
                string[i+1]=temp;
            }
    }
}
int main()
{
  char s[] = "the quick brown fox jumps over the lazy dog"; //string to sort
  char c;
  int i,length;

  length=0;
  i=0;
  while ((s[i])!='\0') //counting the length of the string
  {
  length++;
  i++;
  }
  bubblesort(s,length); //sorting the string (array of characters)
  printf("%s\n",s);
}
```

- alternatively: use header file <string.h> and predefined function:
- size_t **strlen**(const char *str)

Example to read text from a file:

```c
#include <stdio.h>
#include <stdlib.h> // For exit() function
int main()
{
    char c[1000];
    FILE *fptr;

    if ((fptr = fopen("input.txt", "r")) == NULL)
    {
        printf("Error! opening file"); // Program exits if file pointer
returns NULL.
        exit(1);
    }

    // reads text until newline
    fscanf(fptr,"%[^\n]", c);

    printf("Data from the file:\n%s\n”, c);
    fclose(fptr);

    return 0;
}
```

- checking whether the input file (file to be read in) exists and can be opened
- reading text from file until newline is reached