Introduction to Programming and Computational Physics

Lesson n.5

Arrays
Files reading and writing

one-dimensional arrays: vectors

It happens frequently to deal with an homogeneous data set: instead of using several variables of the same kind, we can use an one-dimensional array (a vector)

Imagine that we want to record the temperature every hour and then calculate the mean value in one day:

We could define a set of 24 float variables:

```
float temp1, temp2, temp3,..., temp24;
```

But operating with such a set of variables would be quite inconvenient. We then prefer to use an array of float:

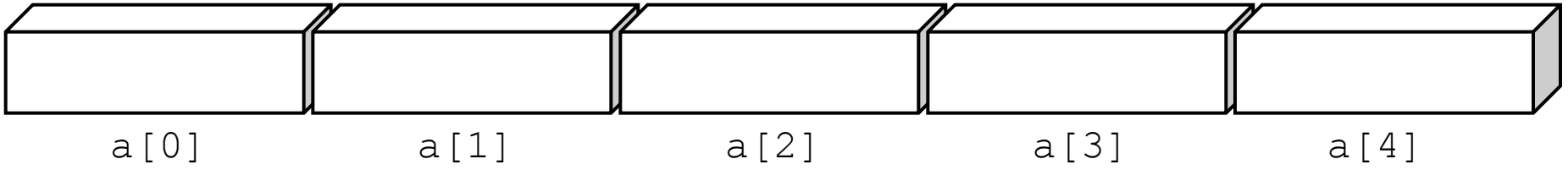
```
float temp[24];
```

one-dimensional arrays: vectors

When we define a vector, like

```
int a[5];
```

the space for 5 *contiguous* integers is allocated in memory



The vector index starts always from zero

In order to access a given element of the vector, we must use the corresponding index

```
a[0] = 5; a[1] = b;
```

```
a[2] = a[0]*2+a[1]*2;
```

```
int n = 3; a[n] = -8; a[n+1] = 20;
```

vector initialization

As for single variables, it is a good habit to initialize vectors before using them. This can be done in a very compact way with a `for` loop:

```
float temp[24];

int idx;

for (idx = 0; idx < 24; idx++)

  temp[idx] = -100;
```

The initial value should be out of the range of the values that the variables are expected to store, so that the user can immediately understand when something is wrong.

the greatest value

```c
#include <stdio.h>

int main(){
   int idx;
   float max=-100.0;
   float temp[24];
   float val;
   for (idx=0; idx<24; idx++)
      {
         printf("Enter the value n.%d: ",idx+1);
         scanf("%f",&val);
         temp[idx] = val;
         // or simply writing: scanf("%f",&temp[idx]);
         if (temp[idx]>max) max = temp[idx];
      }
   printf("The greatest value is: %.1f\n",max);
   return 0;
}
```

two-dimensional arrays: matrices

We can define a two-dimensional array by entering:

```
int a[4][3];
```

so that we have now an array composed by 12 elements (4*3) and the two indexes range now from 0 to 3 and from 0 to 2 or

```
a[0][0]   a[0][1]   a[0][2]
a[1][0]   a[1][1]   a[1][2]
a[2][0]   a[2][1]   a[2][2]
a[3][0]   a[3][1]   a[3][2]
```

At the same time we can perform the initialization:

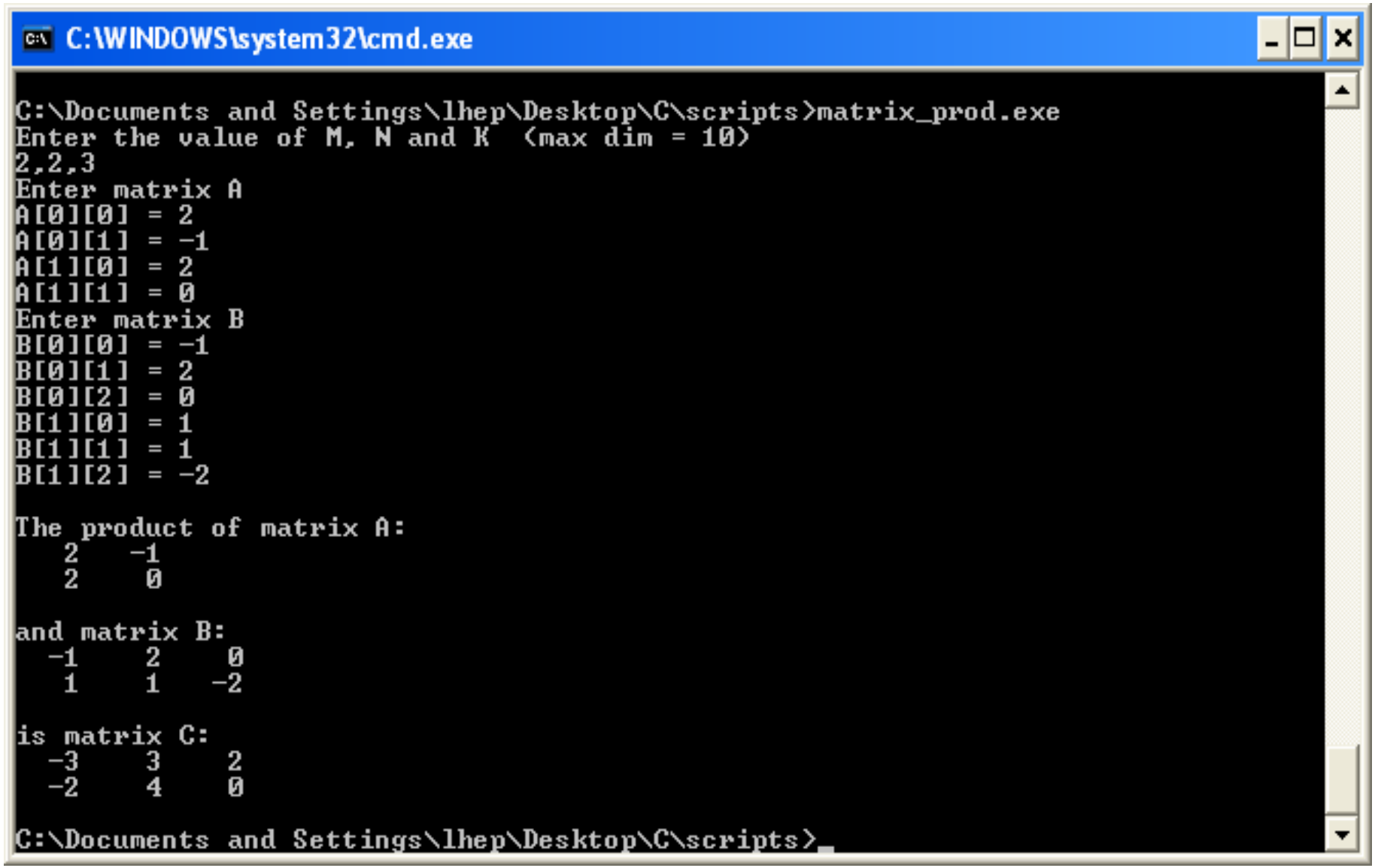
```
int id1,id2;
for(id1=0; id1<4; id1++){
   for(id2=0; id2<3; id2++){
      a[idx1][idx2] = 0;
   }
}
```

```c
/* This program calculate the product of matrices A[L,M]*B[M,N] = C[L,N]  */
#include <stdio.h>

int main()
{
   int L,M,N;
   int i,j,k;
   int A[10][10], B[10][10], C[10][10];
   printf("Enter the value of M, N and K  (max dim = 10) \n");
   scanf("%d,%d,%d",&L, &M, &N);
   printf("Enter matrix A\n");
   for(i=0;i<L;i++)
      for(j=0;j<M;j++)
      {
         printf("A[%d][%d] = ",i,j);
         scanf("%d",&A[i][j]);
      }
   printf("Enter matrix B\n");
   for(j=0;j<M;j++)
      for(k=0;k<N;k++)
      {
         printf("B[%d][%d] = ",j,k);
         scanf("%d",&B[j][k]);
      }
   for(i=0;i<L;i++)
      for(k=0;k<N;k++)
      {
         C[i][k]=0;
         for(j=0;j<M;j++)
            C[i][k] = C[i][k] + A[i][j]*B[j][k];
      }
   printf("\nThe product of matrix A: \n");
   for(i=0;i<L;i++)
   {
      for(j=0;j<M;j++)
         printf("%4d ",A[i][j]);
      printf("\n");
   }
   printf("\nand matrix B: \n");
   for(j=0;j<M;j++)
   {
      for(k=0;k<N;k++)
         printf("%4d ",B[j][k]);
      printf("\n");
   }
   printf("\nis matrix C: \n");
   for(i=0;i<L;i++)
   {
      for(k=0;k<N;k++)
         printf("%4d ",C[i][k]);
      printf("\n");
   }
   return 0;
}
```

Product of matrices

Product of matrices: execution



Files in C language

In order to deal with files (reading, writing, …) we need to introduce a new data type: `FILE` (capital letters)

It is defined in the header `stdio.h` It has to be included if we want to use a file

```
#include <stdio.h>

int main()

{

   FILE *f; //file pointer

   f = fopen("data.txt","r");

   ...

   ...

   fclose(f);

   return 0;

}
```

File opening modalities

```
f = fopen("data.txt","r");
```

"r" reading only.

"w" writing only. If the file doesn't exist, it is automatically created. Otherwise, the previous content is deleted.

"a" appending. The file is opened in writing only mode. If it doesn't exist, it is automatically created. Otherwise, the previous content is not deleted and any new writing operation is added at the end.

Formatted reading and writing

We will deal only with data file and we will introduce only the functions for *formatted* reading and writing: `fprintf fscanf`

The former can be used if the file is opened in writing (or appending) mode, the latter in reading mode.

The two functions work in the same way as `printf` and `scanf`, but the output(input) will be printed(taken) to(from) a file

```
printf("hallo world"); //output to the screen

fprintf(f,"hallo world"); //output to the file
indicated by f
```

The function fscanf

Input file:
data1.txt

```
34.1 40.5
-12.7 4.8
75.7 -10.6
54.9 10.1
-77.8 45.2
1.2 56.8
33.4 -10.3
10.2 22.2
-29.3 10.3
34.0 -55.1
```

```c
#include <stdio.h>
#include <math.h>

int main(){
  FILE *f = fopen("data1.txt", "r");
  int i;
  float X1=0,Y1=0;
  float dist = 0;
  for (i=1; i<=10 ; i++)
    {
      fscanf(f,"%f %f", &X1,&Y1);
      dist = sqrt((X1*X1)+(Y1*Y1));
      printf("The distance from the origin of point no.%d is %.3f\n",i,dist);
    }
  fclose(f);
  return 0;
}
```

```
scanner@lheppc46:/terabig/scan/ciro/C/lec> ./lec6_fscanf.exe
The distance from the origin of point no.1 is 52.944
The distance from the origin of point no.2 is 13.577
The distance from the origin of point no.3 is 76.439
The distance from the origin of point no.4 is 55.821
The distance from the origin of point no.5 is 89.977
The distance from the origin of point no.6 is 56.813
The distance from the origin of point no.7 is 34.952
The distance from the origin of point no.8 is 24.431
The distance from the origin of point no.9 is 31.058
The distance from the origin of point no.10 is 64.746
```

Output to your screen

The function fprintf

```c
#include <stdio.h>
#include <math.h>

int main(){
  FILE *f = fopen("data2.txt", "w");
  float i;
  for (i=1; i<=10 ; i++)
    {
      fprintf(f,"%.3f %.3f %.3f \n", i, i*i,sqrt(i));
    }
  fclose(f);
  return 0;
}
```

```
1.000 1.000 1.000
2.000 4.000 1.414
3.000 9.000 1.732
4.000 16.000 2.000
5.000 25.000 2.236
6.000 36.000 2.449
7.000 49.000 2.646
8.000 64.000 2.828
9.000 81.000 3.000
10.000 100.000 3.162
```

Output file:
data2.txt

The function feof

```
#include <stdio.h>

int ret;

FILE *f;

f = fopen("data.txt","r");

...

...

ret = feof(f);

...
```

The return value of `feof()` is 1 if the file pointer `f` is positioned at the end of the file, otherwise 0. It is very useful when the file length is unknown.

The function feof

```c
#include <stdio.h>
#include <math.h>

int main(){
  FILE *f = fopen("data1.txt", "r");
  int i=1;
  float X1=0,Y1=0;
  float dist=0;
  while(!feof(f))
    {
      fscanf(f,"%f %f \n", &X1,&Y1);
      dist = sqrt((X1*X1)+(Y1*Y1));
      printf("The distance from the origin of point no.%d is %.3f\n",i,dist);
      i++;
    }
  fclose(f);
  return 0;
}
```