1D static arrays, random numbers and working with files

Static arrays,

More often than not we need to process a number of variables. That is our program deals not with simple single-valued data, but with a **collection**. The most common type of a collection, throughout different programming languages is an array of data. Arrays can be thought of as continuous containers (for what it is worth in thay are continuous in memory) that store elements one after the other. I like to think about it as a sequence of cups, each storing some 'stuff', arranged one-after-another.

Arrays can be either **static**, that is such that their size is known before the program is run, and remains fixed throughout the program's lifetime, or **dynamic** that is such that have their size set at *runtime*. We will now have a look at how **static** arrays are handled in **C**. Much of what we show here will be applicable to dynamic arrays.

1. The size of a static array **must be known** at compilation time!
2. How to create an array, syntax:
   - *type name[size]*
   - again: *size* **must** be known at compilation
3. While VLA are allowed in C99 standard, this construct is **forbidden** in this class. As in you will fail on a test if you use it.

1. Declaring a 1D array

So to make the long story short. To declare an array of 10 ints, you do:

In [ ]:
```
int tab[10];
```

But if you do:

In [ ]:
```
int n=10;
int tab[n];
```

You will get exactly **ZERO** (as in 0!) points on a test. You have been warned!

Let us have a complete program, that does nothing, but declares an array of 10 integers:

In [119…
```
int main()
{
    int tab[10]; //an array of 10 integers, i.e. 40B
}
```

2. Accesing elements with square brackets [ ]

```
* Indexing starts with 0 and is up to size-1
* Accessing elements outside an array will (or not but do not count on
that) cause a *run time* error:
```

In [26]:
```c
#include <stdio.h>

int main()
{
    int tab[3];

    tab[0] = 10; // indices start from 0!
    tab[1] = 20;
    tab[2] = 30;

    printf("%d %d %d\n", tab[0], tab[1], tab[2]);
}
```

```
10 20 30
```

And this will cause a *runt-time* error:

In [27]:
```c
#include <stdio.h>

int main()
{
    int tab[3];

    tab[0] = 10; // indices start from 0!
    tab[1] = 20;
    tab[2] = 30;

    tab[3] = 40; // can not access element 3 since it is out of bounds
    printf("%d %d %d %d", tab[0], tab[1], tab[2], tab[3]);
}
```

```
10 20 30 40
*** stack smashing detected ***: terminated
[C kernel] Executable exited with code -6
```

3. How is an array stored in the memory

```
* It is stored in a continous block.
* Elements are separated by the number of bytes equal the size of an array
type, i.e. 4B for ints and floats, 8B for doubles.
```

Let us try to print an address of the first element:

In [11]:
```c
#include <stdio.h>

int main()
{
    int tab[3];

    tab[0] = tab[1] = tab[2] = 0;

    printf("%d %d %d\n", tab[0], tab[1], tab[2]);
    printf("%p\n%p\n%p\n", &tab[0], &tab[1], &tab[2]);
    printf("%p\n%p\n%p\n", tab, tab+1, tab+2);
} // c is 12
```

```
0 0 0
0x7ffc0d45e0fc
0x7ffc0d45e100
0x7ffc0d45e104
0x7ffc0d45e0fc
```

```
0x7ffc0d45e100
0x7ffc0d45e104
```

We note that addresses (position in memory of tab[0] is the same as the same pointed by tab). This indicates that tab is a pointer!?

4. Are arrays pointers?

```
* Yes, almost.
* Some consequences of pointer arithmetics.
```

Use a pointer to an int to access elements of an array if int's.

In [19]:

```c
#include <stdio.h>

int main()
{
    int tab[3];
    int *p;// = &tab[0];
    p = tab;

    p[0] = 10;
    p[1] = 20;
    p[2] = 30;

    printf("%d %d %d | %d\n", tab[0], tab[1], tab[2], p[2]);
    printf("%p %p %p\n%p", &tab[0], &tab[1], &tab[2], p);
}
```

```
10 20 30 | 30
0x7ffc314e9abc 0x7ffc314e9ac0 0x7ffc314e9ac4
0x7ffc314e9abc
```

It seems that using [ ] is equivalent to using a * operation on a pointer, and in consequence the following is possible:

In [25]:

```c
#include <stdio.h>

int main()
{
    int tab[3];

    tab[0] = 10;
    tab[1] = 20;
    tab[2] = 30;

    printf("%d %d %d\n", tab[0], tab[1], tab[2]);
    printf("%d %d %d\n", *tab, *(tab+1), *(tab+2));

    printf("%d %d %d\n", *tab, *(1+tab), *(2+tab));
    printf("%d %d %d\n", 0[tab], 1[tab], 2[tab]);
}
```

```
10 20 30
10 20 30
10 20 30
10 20 30
```

The fact that arrays in **C** allow for access using *index[array_name]* should be treated as a curiosity, and a consequence of pointer arithmetics - never to be used in an actual code, as it

makes it difficult to read and understand!

5. Passing arrays as arguments to functions

When passing an array to a function, the function must know the *type* and the size.

Example: Write a program that uses two function, one for initialization an array of **doubles** and another to print its values.

In [33]:
```c
#include <stdio.h>

// Those here are prototypes of functions
void fillarray(double array[], int size);
void printarray(double array[], int size);

int main()
{
    double tab[500];
    int n = 10;

    fillarray(tab, n);
    printarray(tab, n);
}

// This fills the array with values from 1 up to size
void fillarray(double array[], int size)
{
    for(int i=0; i<size; ++i)
    {
        array[i] = i+1;
    }
}

void printarray(double array[], int size)
{
    for(int i=0; i<size; ++i)
    {
        printf("i=%d, x=%lf\n", i, array[i]);
    }
}
```

```
i=0, x=1.000000
i=1, x=2.000000
i=2, x=3.000000
i=3, x=4.000000
i=4, x=5.000000
i=5, x=6.000000
i=6, x=7.000000
i=7, x=8.000000
i=8, x=9.000000
i=9, x=10.000000
```

6. Set a maximum allowable size of an array to be used

```
  * Use preprocessors **#define**
      * a good practice is to use a **uniqe** identifier for a define to
  avoid problems
```

In [38]:
```c
#include <stdio.h>
#define MAX_SIZE 50
```

```c
// Those here are prototypes of functions
void fillarray(double array[], int size);
void printarray(double array[], int size);

int main()
{
    int  array1[MAX_SIZE];
    char array2[MAX_SIZE];

    double tab[MAX_SIZE];
    int n = 10;

    fillarray(tab, n);
    printarray(tab, n);
}

// This fills the array with values from 1 up to size
void fillarray(double array[], int size)
{
    for(int i=0; i<size; ++i)
    {
        array[i] = i+1;
    }
}

void printarray(double array[], int size)
{
    for(int i=0; i<size; ++i)
    {
        printf("i=%d, x=%lf\n", i, array[i]);
    }
}
```

```
i=0, x=1.000000
i=1, x=2.000000
i=2, x=3.000000
i=3, x=4.000000
i=4, x=5.000000
i=5, x=6.000000
i=6, x=7.000000
i=7, x=8.000000
i=8, x=9.000000
i=9, x=10.000000
```

Random numbers

1. How to generate a random number?

- include **stdlib.h** and use *rand()*
- *rand()* returns a random int from zero up to RAND_MAX

Generate a random number and print it together with the value of RAND_MAX, notice that the result is the same everytime you run the code.

In [92]:

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

int main(){
    int a = rand();
    printf("%d, %d\n", a, RAND_MAX);
}
```

```
1804289383, 2147483647
```

2. Are random numbers random?

- Well now, thay are just consecutive elements of a predefined sequence of numbers that mimic randomness.
- *srand()* initializes the random sequence to start at a different position

Use *srand()*, rerun the code and see that values have changed, but are still the same on consecutive reruns.

In [97]:
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    srand(4);
    int a = rand();
    printf("%d, %d", a, RAND_MAX);
}
```

```
1968078301, 2147483647
```

3. Initialize a random sequence with current time

- Specifically, with the number of seconds (int) that passed since time Zero.
- Zero time is 00:00:00 Thursday, 1 January 1970
- Yes, there is nothing before time zero ...
- On systems where time is represented by a 32 bit intiger time ends after 2147483647 seconds i.e. 3:14:08 19 January 2038 (similar to the Year 2000 problem),see Year_2038_problem.
    - The end of the world is coming!

Run the cose a couple of times and observe that the values change now.

In [109…
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
    printf("%ld since January 1 1970\n", time(NULL));
    srand(time(NULL));

    int a = rand();
    printf("%d, %d", a, RAND_MAX);
}
```

```
1606128313 since January 1 1970
1127191404, 2147483647
```

4. How to generate random doubles in a given range?

- Start with values from zero to 1 and than scale and shift as needed
- or learn formulas by hard ...

Generate random numbers from 0 up to 1:

In [122…
```c
#include <stdio.h>
```

```
#include <stdlib.h> // this here defines RAND_MAX
#include <time.h>

int main()
{
    srand(time(NULL));
    for(int i=0; i<10; ++i)
    {
        double x = (double)rand()/RAND_MAX;
        printf("x=%lf\n", x);
    }
}
```

```
x=0.554433
x=0.898342
x=0.798349
x=0.854367
x=0.567420
x=0.794366
x=0.656387
x=0.620818
x=0.520150
x=0.371525
```

Now modify the program so it returns a value from an arbitrary range:

In [123…

```
#include <stdio.h>
#include <stdlib.h> // this here defines RAND_MAX
#include <time.h>

int main()
{
    srand(time(NULL));

    double x_min = 10, x_max = 15;
    for(int i=0; i<10; ++i)
    {
        double x = (double)rand()/RAND_MAX;
        x = x_min + x*(x_max-x_min);
        printf("x=%lf\n", x);
    }
}
```

```
x=11.573040
x=13.410400
x=12.489999
x=10.993379
x=13.131483
x=14.834072
x=12.573587
x=12.681914
x=11.628608
x=12.526821
```

In [124…

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
    srand(time(NULL));

    double x_max = 20, x_min = 10; // the range of numbers

    double a = rand() / (double)RAND_MAX; // this gives values from zoro to 1
```

```
    printf("Scale to (0, 1): \t %lf\n", a);
    //now scale:
    a *= (x_max - x_min);
    printf("Scale to (0, scale): \t %lf\n", a);
    //and shift
    a += x_min;
    printf("Shift by x_min: \t %lf\n", a);
}
```

```
Scale to (0, 1):         0.182814
Scale to (0, scale):     1.828140
Shift by x_min:          11.828140
```

A basic sorting algorithm - the bubble sort

The simplest algorithm to perform sort of a sequence of values. Works by comparing consecutive pairs of elements in a sequence and swapping them such that the larger is moved towards end of the collection. After the first pass the last element is the largest, in the second pass the process is repeated and the second largest element becomes second from the end, and so on. This is illustrated with the below animation:

Or with the use of Hungarian folk dance: Here (They do other sorting algorithms as well!)

Exercise: Implement Bubble sort and test it for a sequence of random numbers from 0 to 1

In [140…

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MS 100

void tabfill(double a[], int n);
void print(double a[], int n);
void bubble_sort(double a[], int n);

int main()
{
    srand(time(NULL));
    int n = 5;
    double tab[MS];

    tabfill(tab, n);
    print(tab, n);
    bubble_sort(tab, n);
    print(tab, n);
}

//fill an array with values from 0 up to 1
void tabfill(double a[], int n)
{
    for(int i=0; i<n; ++i)
    {
        a[i] = (double)rand() / RAND_MAX;
    }
}

//print elements of an array
void print(double a[], int n)
{
    for(int i=0; i<n; ++i)
```

```
    {
        printf("%lf ", a[i]);
    }
    printf("\n");
}

//sort an array of doubles
void bubble_sort(double a[], int n)
{
    double tmp;
    int issorted = 1;
    for(int j=0; j<n-1; ++j)
    {
        issorted = 1;
        for(int i=0; i<n-1-j; ++i)
        {
            if(a[i] > a[i+1]) // the swap
            {
                tmp = a[i];
                a[i]  = a[i+1];
                a[i+1]= tmp;
                issorted = 0;
            }
        }
        if(issorted==1) return;
    }
}
```

```
0.077629 0.156381 0.093824 0.243227 0.139298
0.077629 0.093824 0.139298 0.156381 0.243227
```

Working with files

1. FILE structure handles access and operations on files
    - Create a pointer to FILE and initialize with fopen()
        - fopen() works in different modes (see Lecture 8)

fopen() usage: *fopen("path_to_file", "access_mode")*.

In the example the file is opened for writing - in this mode file does not need to exist. Should we try to use *fopen* in *reading* mode without the file there would be an error.

In [144…

```
#include <stdio.h>

int main(){
    FILE *f;
    f = fopen("my_file_to_open", "w");

    fclose(f);
}
```

Inspect your working directory, an empty file *my_file_to_open* should have been created.

You can provide a full or relative path to a file with, but **be careful since you can easily override something important**!

In [163…

```
#include <stdio.h>

int main(){
    FILE *f;
```

```
    f = fopen("../my_file_to_open", "w");
    // or f = fopen("/absolute_path/my_file_to_open", "w");

    fclose(f);
}
```

1. Writing to file is performed with fprintf()
    - similar to printf()
    - *fprintf(FILE* , "the message")*

Print some text to a file and on the screen, compare the results:

In [147…

```
#include <stdio.h>

int main(){
    FILE *file = fopen("my_file_to_open", "w");

    //printing to a file with fprintf
    fprintf(file, "I enjoy \t CS1 \t lectures on Monday,\nthis is myfavorite class!\n" );

    //printing on a screen with printf()
    printf("I enjoy \t CS1 \t lectures on Monday,\nthis is my favorite class!\n" );
    fclose(file);
}
```

```
I enjoy          CS1     lectures on Monday,
this is my favorite class!
```

1. Reading from a file with fscanf()
    - similar to scanf()
    - *fscanf(FILE* , "format", type *)*

Create a file data.dat and fill it with some integers. Read and print:

In [150…

```
#include <stdio.h>

int main(){
    FILE *file = fopen("samples/data.dat", "r");

    double a;
    for(int i=0; i<3; ++i)
    {
        fscanf(file, "%lf", &a);
        printf("%lf ", a);
    }

    fclose(file);
}
```

```
12.893726 12.893726 12.893726
```

1. Remembar to clean after yourself, i.e. use fclose()
    - for every *fopen* there needs to be an *fclose*
    - *fclose(FILE*)*

Examples to be done

Example 1:

In [4]:

```
//%cflags:-lm
#include <stdio.h>
#include <math.h>
#define PI 4.0*atan(1.0)

int main(){
    int n;
    //scanf("%d", &n);
    n = 100;
    FILE *f = fopen("samples/sin.dat", "w");
    double h = (2.0*PI - 0) / (n-1);
    double x=0.0;
    double y;
    for(int i=0; i<n; ++i)
    {
        y = sin(x);
        fprintf(f, "%lf, %lf\n", x, y);
        x += h; // x = i*h
    }
    fclose(f);
}
```

Example 2:

In [15]:

```
//%cflags:-lm
#include <stdio.h>
#include <math.h>
#define PI 4.0*atan(1.0)

int main(){
    int n, ninside=0;
    FILE *f = fopen("samples/points.dat", "r");
    fscanf(f, "%d", &n);
    printf("n=%d\n", n);

    double x, y;

    for(int i=0; i<n; ++i)
    {
        //fscanf(f, "%lf %lf", &x, &y);
        fscanf(f, "%lf", &x);
        fscanf(f, "%lf", &y);
        printf("x=%lf y=%lf\n", x, y);
        if( x*x+y*y < 1 )
            ++ninside;
    }
    printf("ninside=%d\n", ninside);

    fclose(f);
}
```

```
n=100
x=0.899915 y=0.480382
x=0.256794 y=0.379916
x=0.825410 y=0.279753
x=0.534659 y=0.416145
x=0.679364 y=0.850037
x=0.154672 y=0.708026
x=0.365754 y=0.066581
x=0.488391 y=0.728678
x=0.669858 y=0.026407
```

x=0.326477 y=0.838494
x=0.556811 y=0.715800
x=0.197294 y=0.376059
x=0.529422 y=0.488890
x=0.907731 y=0.395970
x=0.074972 y=0.499215
x=0.443492 y=0.974886
x=0.979597 y=0.700286
x=0.354802 y=0.805007
x=0.980039 y=0.889461
x=0.221152 y=0.659404
x=0.739498 y=0.375824
x=0.367429 y=0.105252
x=0.442405 y=0.855820
x=0.833930 y=0.112263
x=0.882227 y=0.160406
x=0.950757 y=0.439038
x=0.876206 y=0.148050
x=0.815097 y=0.405628
x=0.636940 y=0.722828
x=0.801598 y=0.711912
x=0.222043 y=0.245090
x=0.686799 y=0.201640
x=0.945376 y=0.041601
x=0.006647 y=0.925416
x=0.931061 y=0.227799
x=0.584819 y=0.670559
x=0.603623 y=0.952249
x=0.775811 y=0.046028
x=0.808069 y=0.609741
x=0.158291 y=0.690296
x=0.770147 y=0.109047
x=0.129334 y=0.646353
x=0.257098 y=0.944431
x=0.051981 y=0.894038
x=0.667259 y=0.853579
x=0.605950 y=0.889302
x=0.098669 y=0.292749
x=0.090942 y=0.044046
x=0.334349 y=0.097589
x=0.969461 y=0.265411
x=0.325388 y=0.554280
x=0.935969 y=0.929011
x=0.506529 y=0.711780
x=0.975039 y=0.314598
x=0.321521 y=0.133329
x=0.004894 y=0.091668
x=0.242377 y=0.134227
x=0.738021 y=0.499474
x=0.078659 y=0.790003
x=0.393512 y=0.745918
x=0.643582 y=0.999463
x=0.635220 y=0.742252
x=0.292212 y=0.726163
x=0.786297 y=0.626561
x=0.823751 y=0.755759
x=0.891972 y=0.149139
x=0.310039 y=0.827941
x=0.078150 y=0.816568
x=0.539722 y=0.053189
x=0.131166 y=0.861243
x=0.186518 y=0.136059
x=0.952911 y=0.428895
x=0.270287 y=0.690932
x=0.928369 y=0.348945
x=0.480935 y=0.321881

```
x=0.094863 y=0.124517
x=0.321344 y=0.730084
x=0.866769 y=0.613556
x=0.456247 y=0.653066
x=0.240117 y=0.279998
x=0.408824 y=0.132088
x=0.429137 y=0.718863
x=0.960030 y=0.507288
x=0.535431 y=0.499751
x=0.560477 y=0.666597
x=0.360994 y=0.746995
x=0.802656 y=0.313904
x=0.175890 y=0.072943
x=0.004836 y=0.104259
x=0.421889 y=0.485771
x=0.426140 y=0.516752
x=0.610288 y=0.747485
x=0.246836 y=0.477057
x=0.361040 y=0.703083
x=0.130123 y=0.601157
x=0.983081 y=0.538947
x=0.733246 y=0.412218
x=0.257810 y=0.693275
x=0.919506 y=0.793241
x=0.193027 y=0.479983
ninside=78
```

Example 3

In [19]:

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

int main(){
    srand(time(NULL));

    int n;
    //scanf("%d", &n);
    n = 1000;

    FILE *fffaaa = fopen("samples/random_values.dat", "w");
    fprintf(fffaaa, "%d\n", n);
    for(int i=0; i<n; ++i)
    {
        fprintf(fffaaa, "%lf\n", (double)rand() / RAND_MAX);
    }
    fclose(fffaaa);
}
```

Example 4

In [40]:

```c
#include <stdio.h>
#define N 5000

double avg(double *arr, int n, int *p1, int *p2);

int main(){
    int n;
    FILE *f = fopen("samples/random_values.dat", "r");
    fscanf(f, "%d", &n);
    printf("n=%d\n", n);
    if(n > N)
    {
```

```
        printf("To large!\n");
        fclose(f);
        return 1;
    }
    double tab[N];

    for(int i=0; i<n; ++i)
    {
        fscanf(f, "%lf", &tab[i]);
        //fscanf(f, "%lf", tab+i);
        //printf("%lf\n", tab[i]);
    }

    int a, b;
    double av = avg(tab, n, &a, &b);
    printf("%lf %d %d\n", av, a, b);

    fclose(f);
}

double avg(double *arr, int n, int *p1, int *p2)
{
    double a = 0;
    int aavg = 0;
    int bavg = 0;
    for(int i=0; i<n; ++i)
    {
        a += arr[i];
    }
    a /= n;
    for(int i=0; i<n; ++i)
    {
        if(arr[i] > a)
        {
            aavg = aavg + 1;
            //aavg += 1;
            //++aavg;
        }
        if(arr[i] < a)
            ++bavg;
    }
    *p1 = aavg;
    *p2 = bavg;
    return a;
}
```

```
n=1000
0.492233 489 511
```

In [ ]: