Lecture 8 Examples

December 3, 2018

1 1D static arrays, random numbers and working with files

1.1 Static arrays,

i.e. size must be known at compilation time

1. How to create an array
   - *type name[size]*
   - *size* **must** be known at compilation

```
In [118]: 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 cause a *run time* error

```
In [107]: #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]);

              //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
```

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.

```
In [109]: #include <stdio.h>

          int main()
          {
              int tab[3];

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

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

```
0x7ffd640ef81c 0x7ffd640ef820 0x7ffd640ef824
```

4. Are arrays pointers?

- Yes, almost.
- Some consequences of pointer arithmetics.

Use a pointer to an int to acess elements of an array if ints.

```
In [119]: #include <stdio.h>

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

              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
0x7ffe05668a0c 0x7ffe05668a10 0x7ffe05668a14
0x7ffe05668a0c
```

It seems that using [ ] is equivalent to using a * operation on a pointer, an in consequence:

```
In [139]: #include <stdio.h>

          int main()
          {
              int tab[3];

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

              printf("Using *(tab+index) \t %d %d %d\n", *tab, *(tab + 1), *(tab + 2));
              // use * on elements

              printf("Using *(index+tab) \t %d %d %d\n", *tab, *(1 + tab), *(2 + tab));

              printf("Using tab[index] \t %d %d %d\n", tab[0], tab[1], tab[2]); // OK

              printf("Using index[tab] \t %d %d %d\n", 0[tab], 1[tab], 2[tab]); // but also!
          }
          // * is equivalent []
```

```
Using *(tab+index)          10 20 30
Using *(index+tab)          10 20 30
Using tab[index]          10 20 30
Using index[tab]          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 acctual code!

5. Passing arrays as arguments to functions

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

```
In [142]: #include <stdio.h>

          //Function that initializez elements of an array
          void fill(int A[], int size){
              for(int i=0; i<size; ++i)
              {
                  A[i] = 5*i+4;
              }
          }

          //Function that prints elements of an array of type int
          void print_1Darray(int B[], int n)
          {
              for(int i=0; i<n; ++i)
              {
```

```
            printf("%d ", B[i]);
        }
        printf("\n");
    }

    int main()
    {
        int tab[5];
        fill(tab, 5);
        print_1Darray(tab, 5);
        //and some fun
        for(int i=0; i<=5; ++i)
        {
            print_1Darray(tab, i);
        }
    }
```

```
4 9 14 19 24

4
4 9
4 9 14
4 9 14 19
4 9 14 19 24
```

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 [143]:
```
#include <stdio.h>
#define MAX_SIZE 100

void fill(int A[], int size){
    for(int i=0; i<size; ++i)
    {
        A[i] = 5*i+4;
    }
}

void print_1Darray(int B[], int n)
{
    for(int i=0; i<n; ++i)
    {
        printf("%d ", B[i]);
    }
    printf("\n");
}
```

```
int main()
{
    int a[MAX_SIZE];
    int n = 10; // change this value to be larger than MAX_SIZE
    //scanf("%d", &n);
    if(n > MAX_SIZE)
    {
        printf("%d is max \n", MAX_SIZE);
        return -1;
    }

    fill(a, n);
    print_1Darray(a, n);
}
```

```
4 9 14 19 24 29 34 39 44 49
```

1.2 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 [145]: #include <stdio.h>
          #include <stdlib.h>

          int main(){
              int a = rand();
              printf("%d, %d", 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 [47]: #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 [148]: #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);
          }
```

```
1543838265 since January 1 1970
183829880, 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 ...

```
In [154]: #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):          0.022264
Scale to (0, scale):          0.222642
Shift by x_min:          10.222642
```

1.3 A basic sorting algorithm - the bubble sort

The simplest alghoritm to perform sort of a sequence of values. Works by comparing consequtive pairs of elements in a sequence and swaping 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 repeted and the second largest element becomes seconf from the end, and so on.

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

```
In [161]: #include <stdio.h>
          #include <stdlib.h>
          #include <time.h>

          //a prototype od a function implemented below
          void print_1Darray(double B[], int n);
          //A bubble sort function
          void bsort(double a[], int n)
          {
              for(int j=0; j<n-1; ++j) // repeat until all alements are moved
              {
                  for(int i=0; i<n-1-j; ++i) //traverse the collection from 0 to n-1-j
                  {
                      if(a[i] > a[i+1]) // swap
                      {
                          double tmp = a[i];
                          a[i] = a[i+1];
                          a[i+1] = tmp;
                      }
                  }
              }
          }

          int main(){
```

```
        srand(time(NULL));

        double tab[10];
        for(int i=0; i<10; ++i)
        {
            tab[i] = rand() / (double)RAND_MAX;
        }

        printf("Original array:\n");
        print_1Darray(tab, 10);
        printf("\nAfter sorting, values are increasing:\n");
        bsort(tab, 10);
        print_1Darray(tab, 10);
    }

    void print_1Darray(double B[], int n)
    {
        for(int i=0; i<n; ++i)
        {
            printf("%lf ", B[i]);
        }
        printf("\n");
    }
```

```
Original array:
0.224041 0.652877 0.226167 0.114034 0.137115 0.017589 0.361988 0.600466 0.873132 0.891514

After sorting, values are increasing:
0.017589 0.114034 0.137115 0.224041 0.226167 0.361988 0.600466 0.652877 0.873132 0.891514
```

1.4 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 [93]: #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);
          }
```

2. 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 [172]: #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!\
              //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!
```

3. 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 [167]: #include <stdio.h>

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

              int a;
              for(int i=0; i<7; ++i)
```

```
        {
            fscanf(file, "%d", &a);
            printf("%d ", a);
        }

        fclose(file);
    }
```

```
1 2 3 4 5 6 6
```

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