Lecture 7
1D Arrays

Test is coming

5'th December

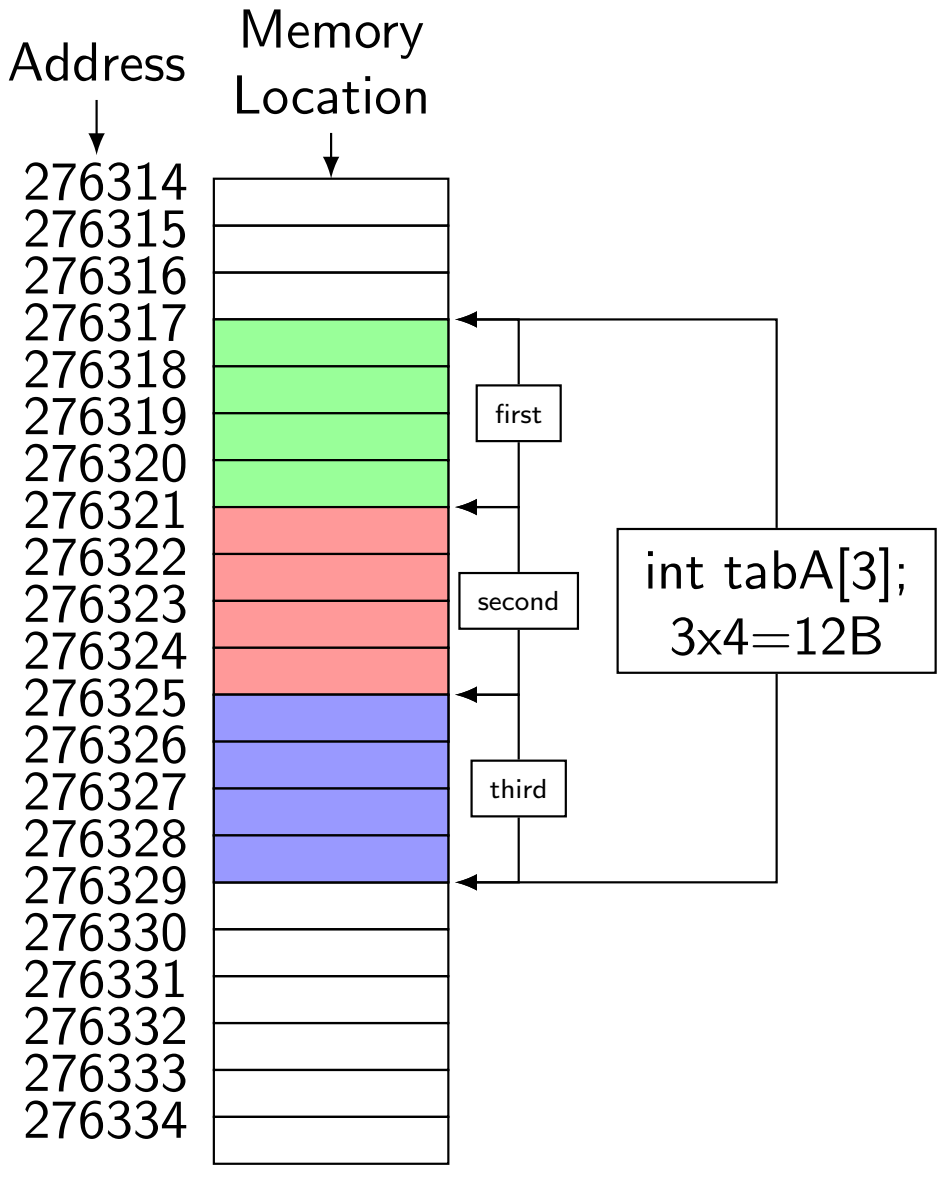
- Data types
- Functions
- I/O operations
- Branching (if, switch)
- Loops

Today

Fun as always at least for some

- How to create 1D static arrays
- How do arrays compare to pointers
- Some (strange) consequences of pointer arithmetic
- Functions, and passing arrays to functions
- Basic sorting algorithms
- ———
- Input output operatios on files
- Generating random numbers

1D arrays
Declaration and memory



Syntax:

```
type name[size]
```

- *type* - almost any type, pointer, etc.
- *name* - an identifier
- *size* - **MUST** be known at compilation time

e.g.:

```
//Array of 3 ints
int tabA[3];
//array of 5 doubles
double tabB[5];
```

- Continous in memory
- Occupies *size* x *sizeof(type)* B

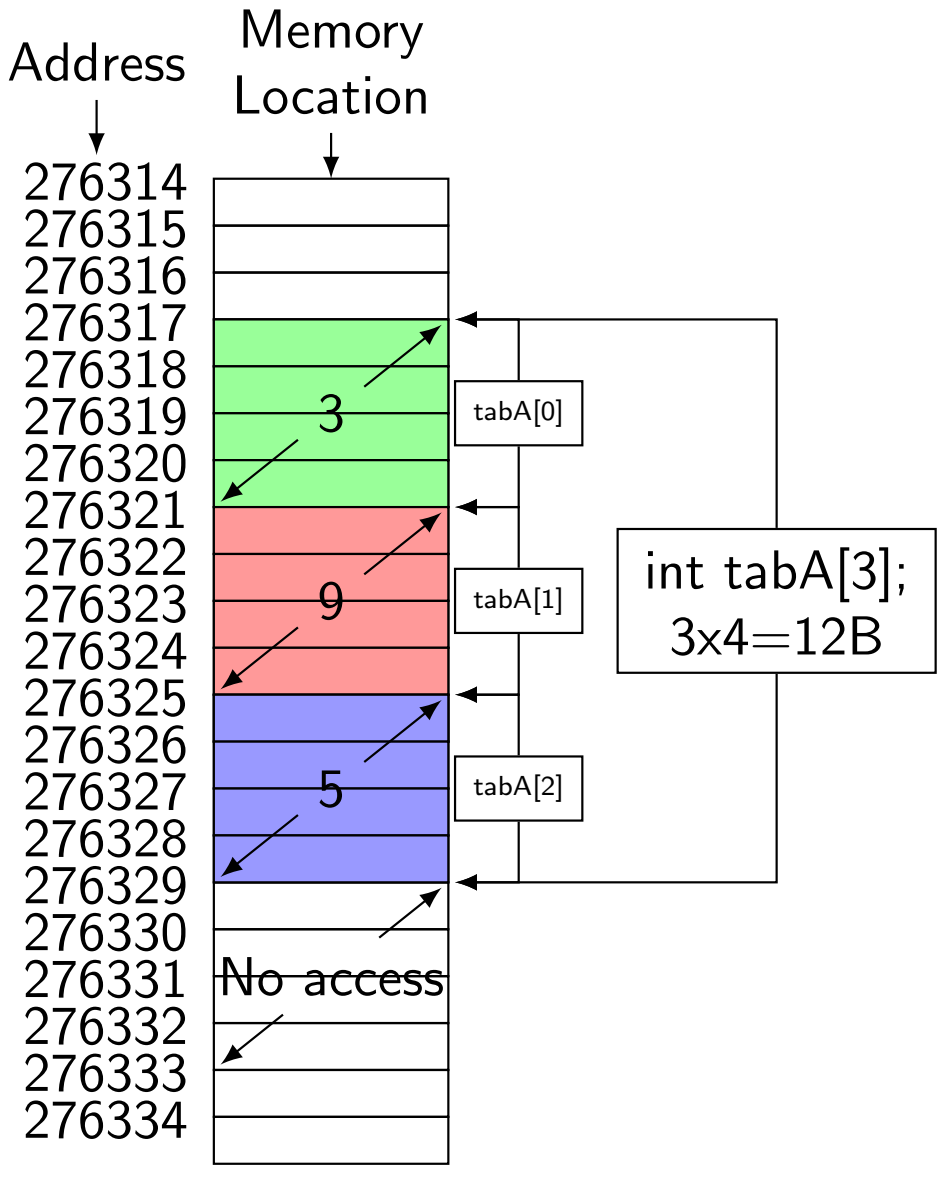
1D arrays
The size MUST be known

You will lose points if you do:

```
int n=8;
int tabA[n];
scanf("%d",&n);
double tabA[n];
```

1D arrays

Acces to elements



- Acces elements with [ ]
- Elements are indexed from 0
- Last element is *size-1*
- Must make sure not to acces out of bounds

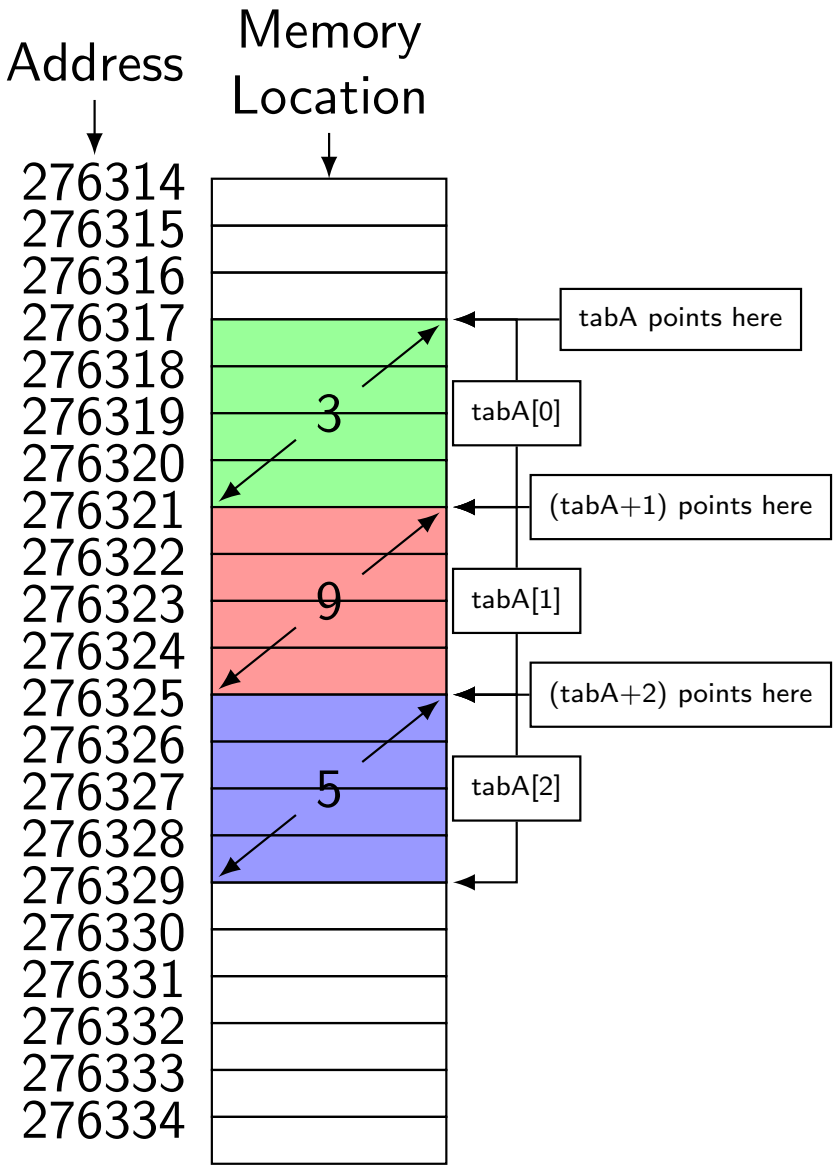
```
int tabA[3];
tabA[0] = 3;
tabA[1] = 9;
tabA[2] = 5;
```

- Out of bounds access:

```
tabA[3]=0; //Error!!
```

1D arrays

Arrays are pointers



- Arrays are pointers
- *tabA* points at the beginning of the array
- &tabA[0] is equivalent to tabA
- pointer arithmetic applies
- (+ means +4B for int)
- * works

```
int tabA[3];
int *p=tabA;// no &
*p;//same as tabA[0]
*(p+1)//same as tabA[1]
*(p+2)//same as tabA[2]
```

- There are some consequences ...

Passing arrays to functions

Syntax:

```
function_type function_name(array_type local_name[], int array_size)
```

e.g.:

```
void FillArray(int A[], int n)
{
  for(int i=0; i<n; ++i)
    A[i]=i;
}
```

Sorting
bubble sort

6 5 3 1 8 7 2 4

- Simple sorting algorithm
- Compares pairs of elements, going through the collection
- easy implementation
- slow and impractical
- Complexity - cost, number of operations
- Worst $\sim n^2$ $O(n^2)$
- Best $\sim n$ $O(n)$
- There are better!

Bubble sort

```
void bubble_sort(int list[], int n)
{
  int c, d, t;

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (list[d] > list[d+1])
      {
        t          = list[d];
        list[d]    = list[d+1];
        list[d+1] = t;
      }
    }
  }
}
```

Random numbers

```
#include <stdlib.h> //for random
#include <time.h> // for time

int my_random_number = rand();
//returns a number from a pseudo random sequence
//from 0 to RAND_MAX

srand(4);
//initialize the pseudo random sequence at some position

//use system time, to get different results at each run
//more rendomness
srand(time(NULL));
```

Files

```
FILE *f; // pointer to a file
f = fopen("name.dat", "wr"); //open in write read

//Use f with fprintf() and fscanf,
//very much like printf and scanf
fprintf(f, "Yhe␣messege␣to␣a␣file\n");
int a;
fscanf(f, "%d", &a); \\read from a file

fclose(f); // for each open there must be an fclose()
```