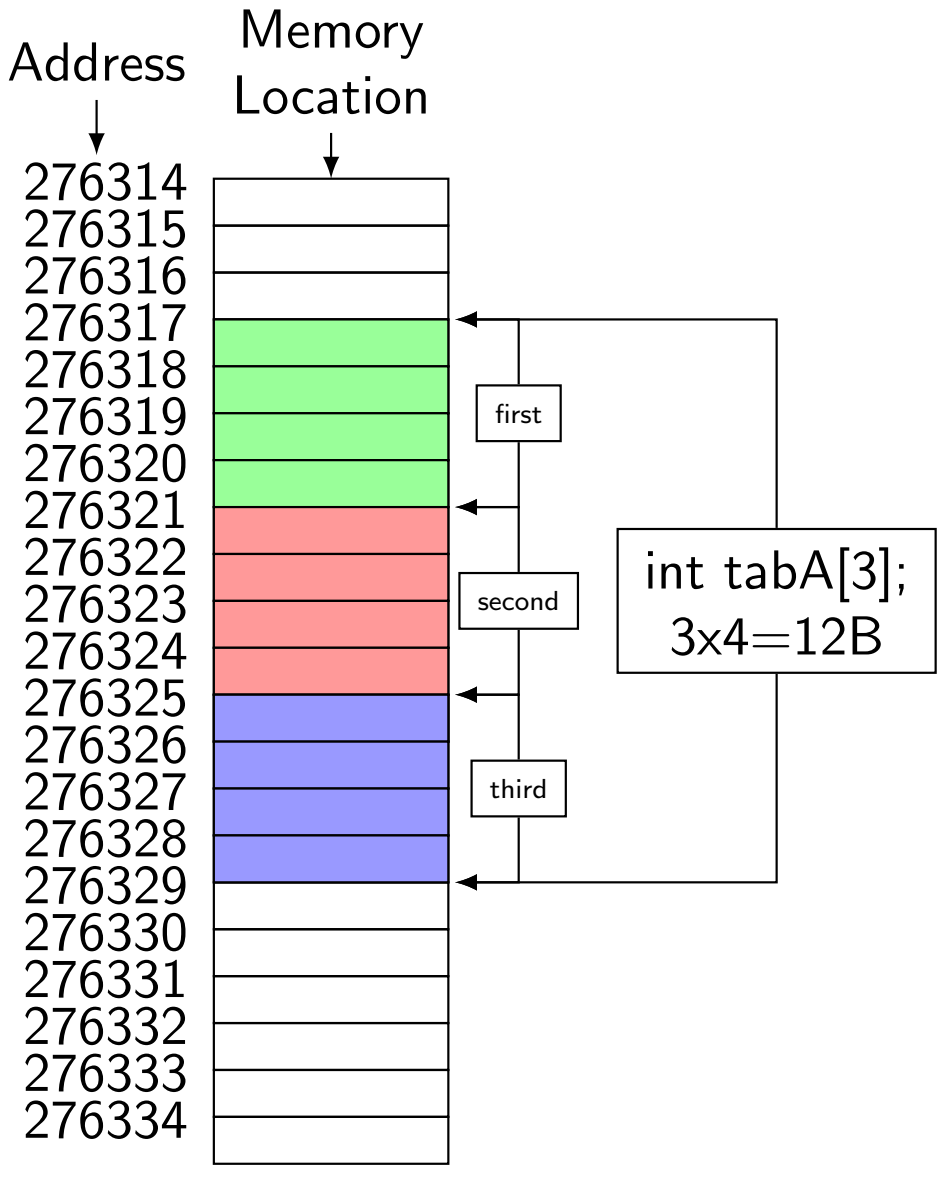
Lecture 7
1D Arrays

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 - bubble sort
- Generating random numbers
- Input output operatios on files
- Examples

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

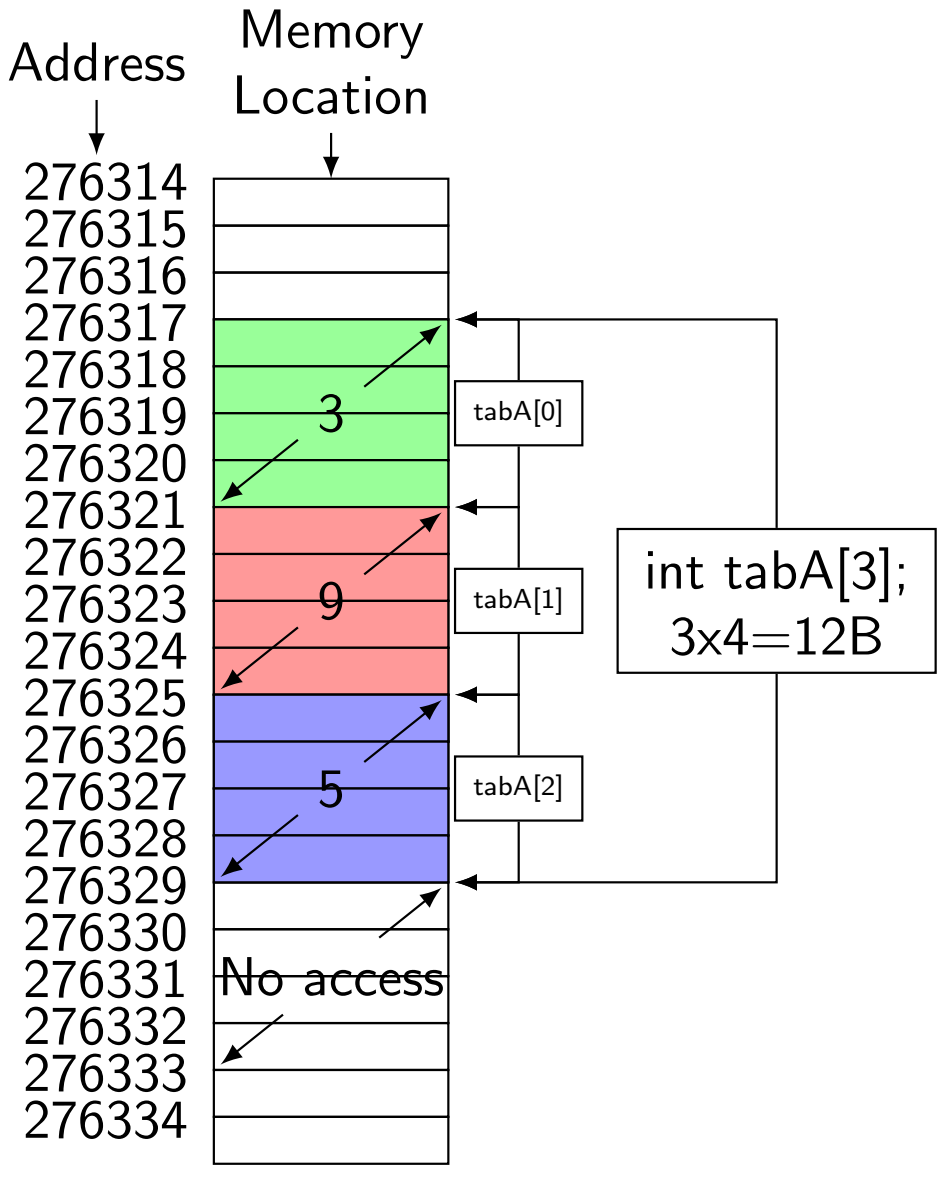
1D arrays
The size MUST be known

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

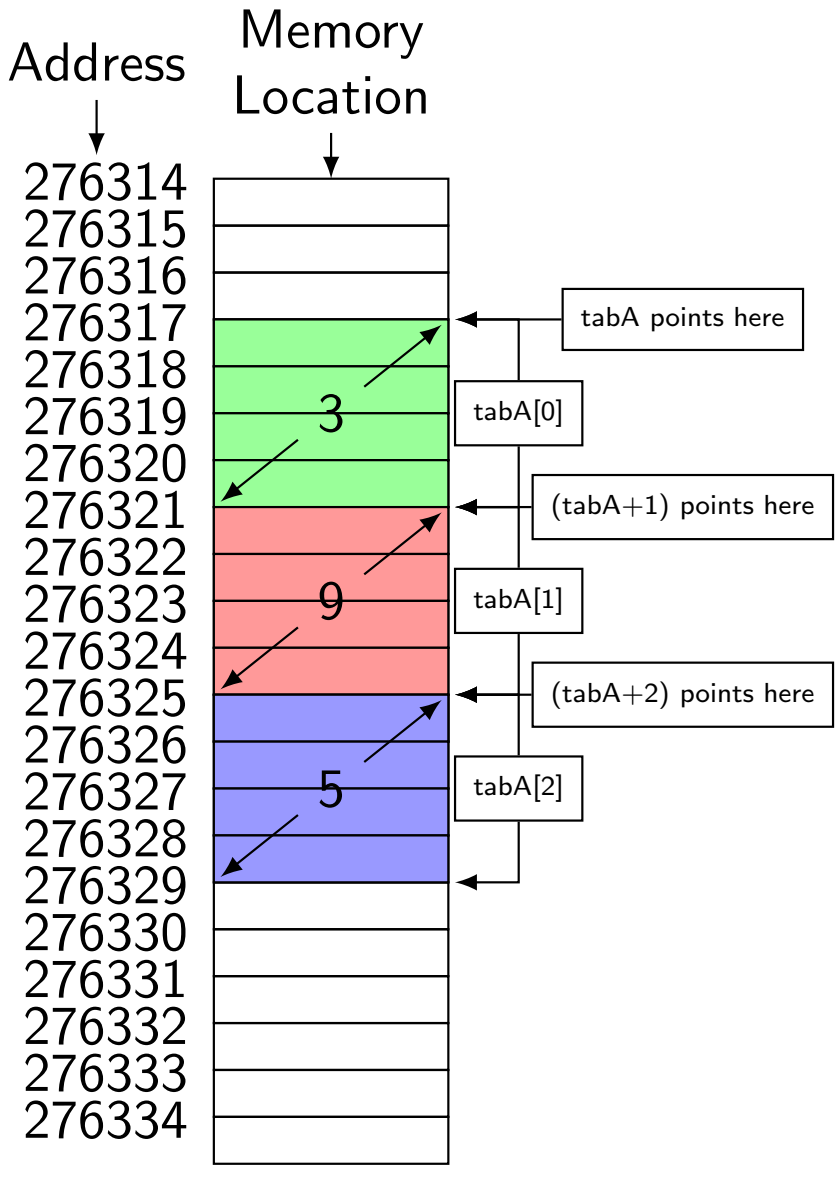
```
int tabA[3];
tabA[0] = 3;
tabA[1] = 9;
tabA[2] = 5;
```

- 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;
}
```

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));
```

Sorting
bubble sort

- 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;
      }
    }
  }
}
```

Files

FILE structure to handle files:

```
FILE *fp;
```

To open a file use *fopen()*:

```
FILE *fopen(const char *filename, const char *mode);
//e.g.:
fp=fopen("c:\\test.txt", "r");
```

To close a file use *fclose()*:

```
int fclose(FILE *a_file);
//e.g.:
fclose(fp);
```

Files

fopen modes

Depending on what we require the file to:

- r - open for reading
- w - open for writing (file need not exist)
- a - open for appending (file need not exist)
- r+ - open for reading and writing, start at beginning
- w+ - open for reading and writing (overwrite file)
- a+ - open for reading and writing (append if file exists)

Files

Reading and writing with fprintf, fscanf

Printing to file:

```
FILE *fp;
fp=fopen("c:\\test.txt", "w");
fprintf(fp, "Testing...\n");

...
fclose(fp);
```

Reading from file:

```
FILE *fp;
fp=fopen("c:\\test.txt", "w");
int a;
fcanf(fp, "%d", &a);
...
fclose(fp);
```

Examples

Use static arrays only.

1. Write a program that writes to a file coordinates to plot $f(x) = sin(x)$ for a range $< 0, 2\pi >$
2. Write program that reads points coordinates from a file and decides if those are in a circle of radius 1.
3. Write a program that generates N random numbers and stores them to a file.
4. Write a program that reads a data file, calculates an average value and finds the number of elements above, and below that average.
5. Write a program that reads values from a file, sorts them and stores them to a new file.
6. Example test questions

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As the
was some unprocessed data that should have been added to the final
page this extra page has been added to receive it.
If you rerun the document (without altering it) this surplus page will g
away, because LaTeX now knows how many pages to expect for this
document.