C introduction part 2

pointers, arrays, strings

off topic: numeric calculations

more here: https://stakahama.gitlab.io/sie-eng270/C_intro.html#org663baba

Values depend on **both** the *data type of the computed value* and *that of the variable which is used to store the results*.

Examine the following code:

```
#include <stdio.h>

int main() {
  int a = 0;
  double b = 0;
  double c = 0;
  double d = 0;
  a = 1 / 2;
  b = 1 / 2;
  c = 1. / 2;
  d = 1. / (double)2;
  printf("a = %d\n", a);
  printf("b = %f\n", b);
  printf("c = %f\n", c);
  printf("d = %f\n", d);
  return 0;
}
```

If we save this to a file called "numtest.c":

```
gcc -Wall -o numtest numtest.c && ./numtest
```

We get the output:

```
a = 0
b = 0.000000
c = 0.500000
d = 0.500000
```

Why? `double a = 0;` only specifies that the variable `a` should be of storage type `double`; the value that is assigned is actually determined by the operation `1 / 2`, which is an integer type - resulting in a value truncated toward zero (i.e., fractional part discarded) when stored as a floating point number. `1. / 2` casts the integer to the double type (same as `1. / (double) 2`) and returns the floating point value.

Casting:

- implicit (e.g., `1. / 2`, `b = 1 / 2`)
- explicit (e.g., `1. / (double)2`)

pointers

What happens when you pass an array to a function?
MATLAB

```
x = [1, 2];
y = foo(x);

fprintf('first element of x is %d\n', x(1));
fprintf('first element of y is %d\n', y(1));

function y = foo(x)
x(1) = 0;
y = x .^ 2;
end
```

What happens when you pass an array to a function?
Python

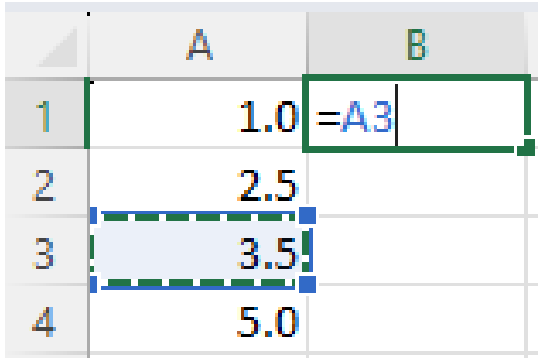
```
import numpy as np

def foo(x):
    x[0] = 0
    y = x ** 2
    return y

x = np.array([1, 2]);
y = foo(x);

print('first element of x is %d\n' % x[0])
print('first element of y is %d\n' % y[0])
```

(note – this behavior is same with lists)

Pointers

- A data type for storing addresses of variables
- Use to pass data around – e.g., into and out of functions –without making copies (memory-efficient)
- “pass by reference” vs “pass by value”

Conceptual illustration with Excel



	Regular variable	Pointer variable
Retrieve value	x = 3.5	*p = 3.5 *"dereferencing"*
Retrieve address	&x = A3	p = A3

Use of pointers

```c
#include <stdio.h>

int main() {

  int *pc = NULL;
  int c = 1;
  printf("%p\n", &c);   // -> 0x7fff34ef4e6c
  printf("%p\n", pc);   // -> (nil)

  pc = &c;
  printf("%p\n", &c);   // -> 0x7fff34ef4e6c
  printf("%p\n", pc);   // -> 0x7fff34ef4e6c

  printf("%d\n", c);    // -> 1
  printf("%d\n", *pc);  // -> 1

  *pc = 2;
  printf("%d\n", c);    // -> 2
  printf("%d\n", *pc);  // -> 2

  return 0;
}
```

Passing pointers to functions

```c
#include<stdio.h>

void div(int a, int b, int *quotient, int *remainder) {
   *quotient = a / b;
   *remainder = a % b;
}

int main() {
   int a = 76, b = 10;
   int q, r;
   div(a, b, &q, &r);
   printf("quotient is %d & remainder is %d\n", q, r); // quotient is 7 & remainder is 6
   return 0;
}
```

Arrays

Tableaux

La ligne suivante crée un tableau de 5 entiers:

```
int visiteurs[5];
```

Les cases de ce tableau peuvent être accédées en indiquant leur indice:

```
visiteurs[0] = 67;
visiteurs[1] = 55;
visiteurs[2] = 33;
visiteurs[3] = 41;
visiteurs[4] = 48;

int somme = 0;
for (int i = 0; i < 5; i++) {
    somme += visiteurs[i];
}

printf("Au total, il y avait %d visiteurs.\n", somme);
```

La numérotation des cases commence par 0. La dernière case porte le numéro 4 (*longueur – 1*).

Alternativement, un tableau peut être créé avec une liste de valeurs:

```
int visiteurs[] = {67, 55, 33, 41, 48};
```

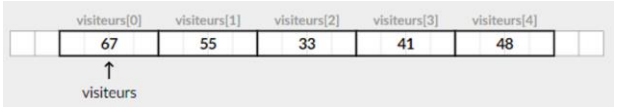
Dans la mémoire de l'ordinateur, les cases sont placées les unes après les autres:

Pointer arithmetic

```c
#include <stdio.h>

int main() {
  int visiteurs[] = {67, 55, 33, 41, 48};

  int somme = 0;
  for (int i = 0; i < 5; i++) {
    somme += *(visiteurs + i);
  }

  printf("Au total, il y avait %d visiteurs.\n", somme);

  return 0;
}
```

array indexing notation adds syntactic sugar

```c
#include <stdio.h>

int main() {
  int visiteurs[] = {67, 55, 33, 41, 48};

  int somme = 0;
  for (int i = 0; i < 5; i++) {
    somme += visiteurs[i];
  }

  printf("Au total, il y avait %d visiteurs.\n", somme);

  return 0;
}
```



integers (int) are 32 bits
visiteurs + i points to the memory address visiteurs + i*32 bits ahead

to retrieve the value at this address, write *(visiteurs + i), or visiteurs[i]

Arrays “decay” to pointers in many operations

Don’t need to explicitly pass address with &

Assignment to pointer

```c
#include <stdio.h>
#define SIZE 3

int main() {
  size_t i = 0;
  int *p = NULL;
  int a[SIZE];

  p = a;
  /* Setting up the values to be i*i */
  for(i = 0; i < SIZE; i++) {
    a[i] = i * i;                                   // assign
    printf("a[i] = %d; *(p+i) = %d\n", a[i], *(p+i)); // read (they are both the same)
  }

  /* Reading the values using pointers */
  for(p = a; p < a + SIZE; p++) {
    printf("*p = %d\n", *p);
  }

  return 0;
}
```

```
a[i] = 0; *(p+i) = 0
a[i] = 1; *(p+i) = 1
a[i] = 4; *(p+i) = 4
*p = 0
*p = 1
*p = 4
```

Pass to function

```c
#include <stdio.h>  // for printf
#include <math.h>   // for pow

#define NELEM 4

void squarearray(int *arr_in, int *arr_out) {

  for(int i=0; i < NELEM; i++) {
    arr_out[i] = pow(arr_in[i], 2);
  }

}

int main() {

  int arr[NELEM] = {0, 1, 2, 3};
  int out[NELEM] = {0};

  squarearray(arr, out);

  for(int i=0; i < NELEM; i++) {
    printf("original myarr[%d] = %d; squared out[%d]: %d\n", i, arr[i], i, out[i]);
  }

  return 0;
}
```

```
original myarr[0] = 0; squared out[0]: 0
original myarr[1] = 1; squared out[1]: 1
original myarr[2] = 2; squared out[2]: 4
original myarr[3] = 3; squared out[3]: 9
```

Strings

- Array of characters

- However it needs to have a null character at the end (integer value of 0, or ‘\0’), so length of strings need to be one unit longer than number of “visible” characters

```c
#include <stdio.h>

int main() {

  char string[] = "abc";

  size_t size = sizeof(string)/sizeof(string[0]);

  printf("length of \"string\" is %d\n", size); // -> length of "string" is 4

  return 0;
}
```

elements of “string”

Index	Char
0	‘a’
1	‘b’
2	‘c’
3	‘\0’

Use of single and double quotes

- Python and MATLAB – can define characters and strings with single or double quotes

- C
  - single quotes for character: ‘x’ is a single character
  - double quotes for string (includes ‘\0’ at end): "x" is a two-character array that contains {'x', '\0'}

Many ways to define strings
immutable and mutable

```c
#include <stdio.h>

int main() {

  char a[] = {'a', 'b', 'c', '\0'};            // mutable
  char b[] = "abc";                            // mutable
  char const c[] = {'a', 'b', 'c', '\0'};      // immutable
  char *d = "abc";                             // immutable

  a[1] = 'a';
  b[1] = 'a';

  printf("%s\n", a); // -> aac
  printf("%s\n", b); // -> aac
  printf("%s\n", c); // -> abc
  printf("%s\n", d); // -> abc

  return 0;
}
```

String functions <string.h>

https://sieprog.ch/#c/string

More functions

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

int main () {
   int day, year;
   char weekday[20], month[20], dtm[100];

   strcpy( dtm, "Saturday March 25 1989" );
   sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );

   return(0);
}
```

Pointer arithmetic with strings

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

int main () {
  char str[] = "column1;column2";
  char ch = ';';
  char *ret;

  ret = strchr(str, ch);        // search
  printf("String after \"%c\" is \"%s\"\n", ch, ret+1);

  *ret = ',';              // replacement
  printf("New string is \"%s\"\n", str);

  return(0);
}
```

String after ";" is "column2"

New string is "column1,column2"

ret is the location of the semicolon (;) in the string, so to retrieve the text after this point, we need to use ret+1

we *do not* need to dereference ret+1 for printing the value of the string (note difference with integer array).

we *do* need to derference ret to make an assignment.