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

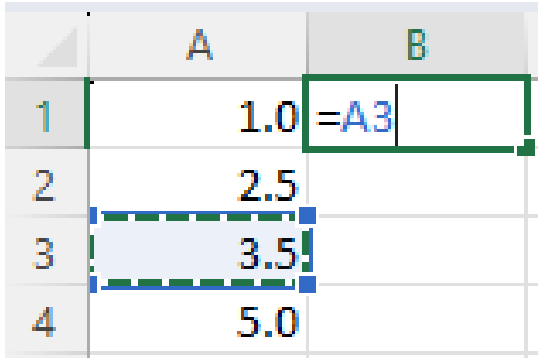
```
import numpy as np

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};
```

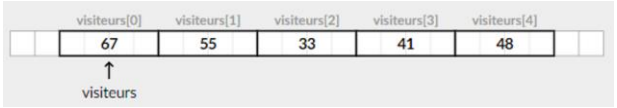
Dans la mémoire de l'ordinateur, les cases sont placées les unes après les autres:

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 – e.g., assign string values to character array

```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.

Review: arrays, strings, and pointers

Pointer syntax

```
int i = 1;
int *ip;
ip = &i;
*ip = 2;
```

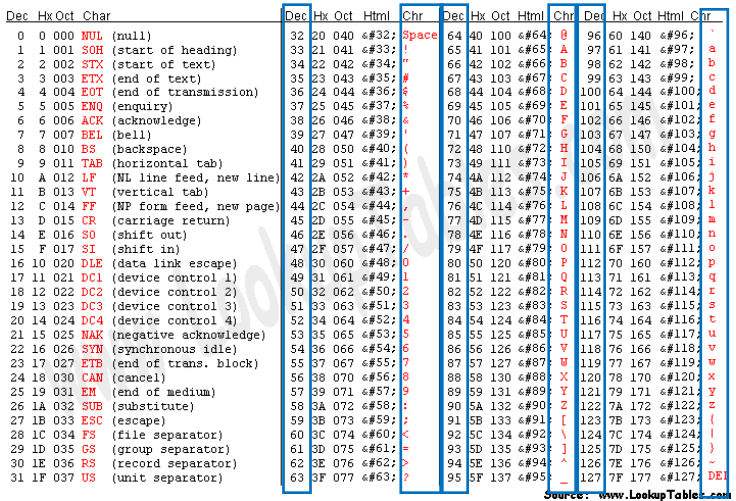
declare pointer of type int*

assign 2 as value to variable at pointer

“*” means different things here but both are used in relation to pointers

Example
values and addresses

- numeric array
- character array (string)
- character array can be converted to and from integers



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
#define LEN 18

int main() {

  int numarr[LEN];
  char chararr[LEN+1];

  numarr[0] = 87;
  numarr[1] = 101;
  numarr[2] = 100;
  numarr[3] = 110;
  numarr[4] = 101;
  numarr[5] = 115;
  numarr[6] = 100;
  numarr[7] = 97;
  numarr[8] = 121;
  numarr[9] = 58;
  numarr[10] = 84;
  numarr[11] = 104;
  numarr[12] = 117;
  numarr[13] = 114;
  numarr[14] = 115;
  numarr[15] = 100;
  numarr[16] = 97;
  numarr[17] = 121;

  // Convert integer to
  // ASCII characters
  // = "Wednesday:Thursday";
  for(int i=0; i<LEN; i++) {
    chararr[i] = (char)numarr[i];
  }
  chararr[LEN] = '\0';
```

```c
    // For printing example
    int pos = 9;

    size_t numsize = sizeof(numarr);
    printf("size of numarr is %lu bytes\n", numsize);
    printf("address of (numarr + %d) is %p\n", pos, numarr + pos);
    printf("value of *(numarr + %d) is %d\n", pos, *(numarr + pos));
    printf("value of numarr[%d] is %d\n", pos, numarr[pos]);
    printf("\n");
```

```
size of numarr is 72 bytes
address of (numarr + 9) is 0x7ffcca8a1cb4
value of *(numarr + 9) is 58
value of numarr[9] is 58
```

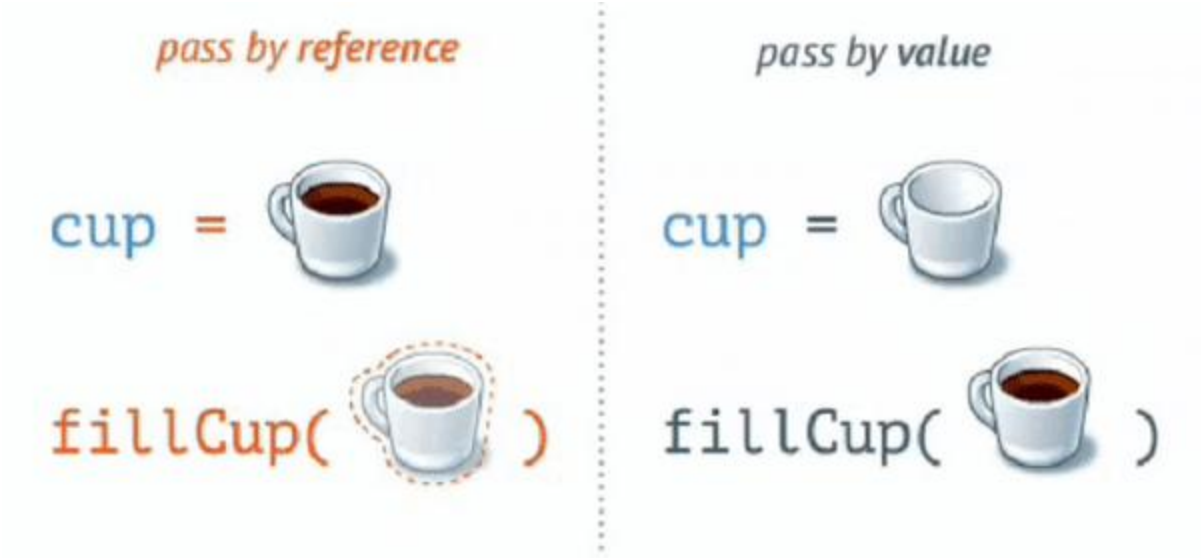
```c
    size_t charsize = sizeof(chararr);
    int charlen = strlen(chararr);
    printf("size of chararr is %lu bytes\n", charsize);
    printf("string length of chararr is %d\n", charlen);
    printf("address of (chararr + %d) is %p\n", pos, chararr + pos);
    printf("string starting at address (chararr + %d) is \"%s\"\n", pos, chararr + pos); // same as &chararr[pos]
    printf("full string of chararr is \"%s\"\n", chararr);
    printf("full string at &chararr[0] \"%s\"\n", &chararr[0]); // same as &(*chararr) or &(*(chararr + 0))
    printf("value of *(chararr + %d) is '%c'\n", pos, *(chararr + pos));
    printf("value of chararr[%d] is '%c'\n", pos, chararr[pos]);

    return 0;
}
```

```
size of chararr is 19 bytes
string length of chararr is 18
address of (chararr + 9) is 0x7ffcca8a1ce9
string starting at address (chararr + 9) is ":Thursday"
full string of chararr is "Wednesday:Thursday"
full string at &chararr[0] is "Wednesday:Thursday"
value of *(chararr + 9) is ':'
value of chararr[9] is ':'
```

Scope

- “region” of program where a particular set of variables are defined and used (where they have meaning)
- scopes in C:
  - global – variables can be used by all functions
  - local within each function (including main())

```c
#include <stdio.h>

double max(double * tableau, int longueur) {
    double valeurMax = tableau[0];
    for (int i = 1; i < longueur; i++) {
        if (tableau[i] > valeurMax) valeurMax = tableau[i];
    }
    return valeurMax;
}

int main(int argc, char * argv[]) {
    double temperatures[] = {24.2, 26.5, 27.4, 28.1, 26.9};
    double temperatureMax = max(temperatures, 5);
    printf("Température maximale: %0.1f\n", temperatureMax);
}
```



source: https://dev.to/erraghavkhanna/ ?

¯_(ツ)_/¯ @SeanTAllen · Nov 3, 2020

If you clone a person, are they the same person?

Yes	20%
No	**80%**

556 votes · Final results

15 15 24

Liam Hammett
@LiamHammett

Depends if you clone by reference or by value

11:04 PM · Nov 3, 2020