C introduction part 3

pointers, static and dynamic memory allocation, and structures

Objectives

- Write functions that can accept and “return arrays”
  - Review of pointers
  - Static and dynamic arrays

- Use *structures* to store and pass around data

https://sieprog.ch/
https://stakahama.gitlab.io/sie-eng270/C_intro.html

Review: arrays, strings, and pointers

Pointer syntax

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

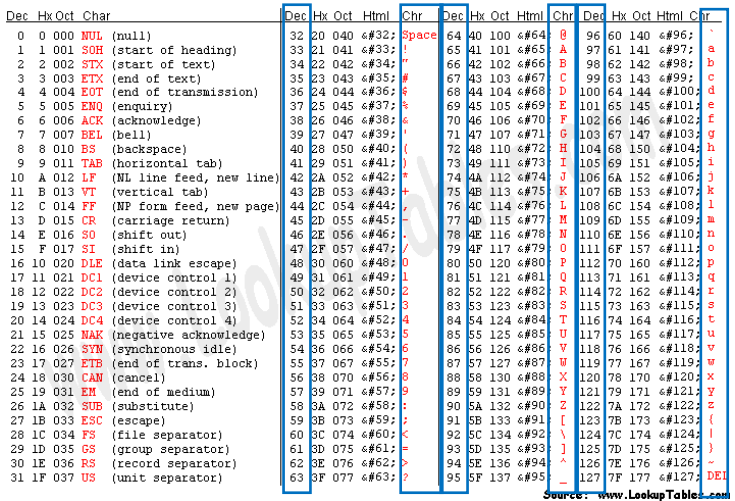
declare pointer of type int*

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
```

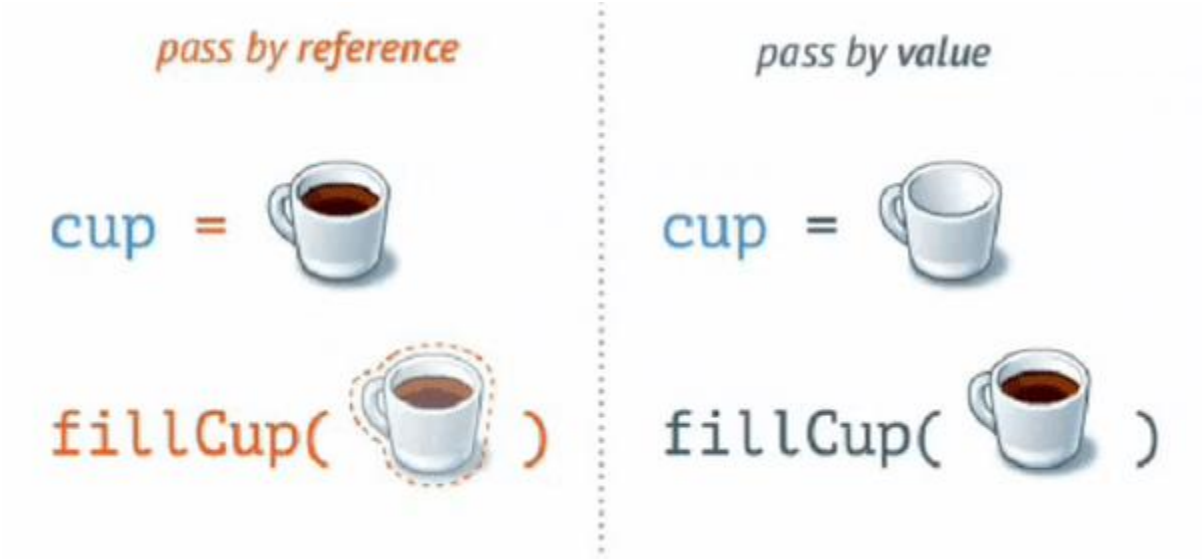
```c
  size_t charsize = sizeof(chararr);
  int charlen = strlen(chararr);
  printf("size of chararr is %lu bytes\n", charsize);
  printf("string length of chararr is %d\n", charlen);
  printf("address of (chararr + %d) is %p\n", pos, chararr + pos);
  printf("string starting at address (chararr + %d) 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

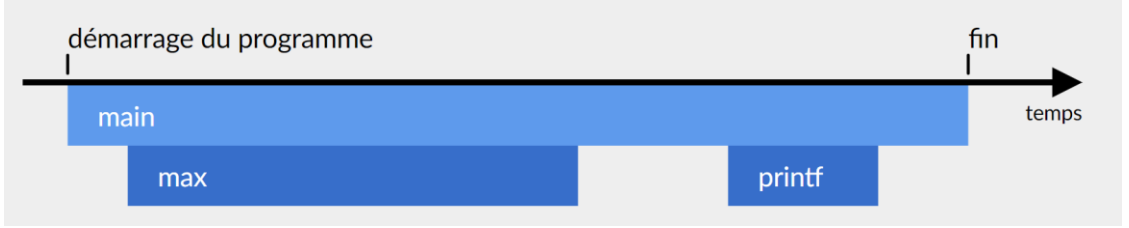
Depends if you clone by reference or by value

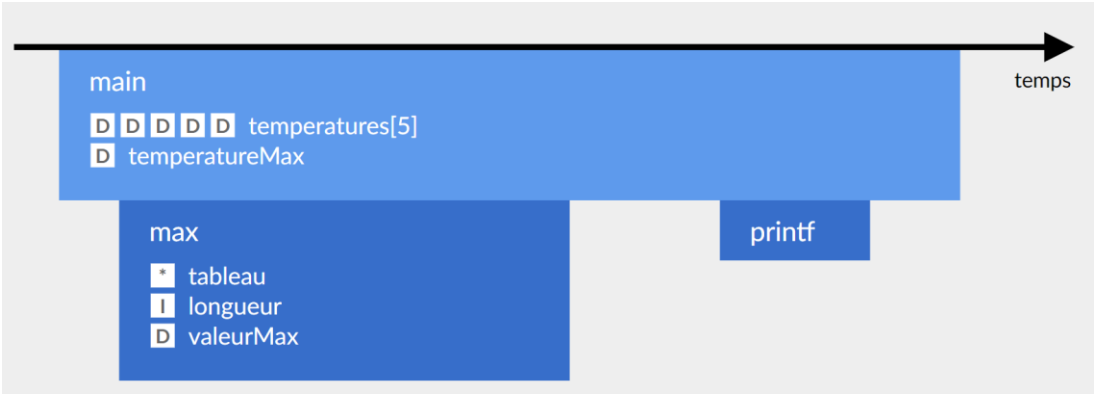
11:04 PM · Nov 3, 2020

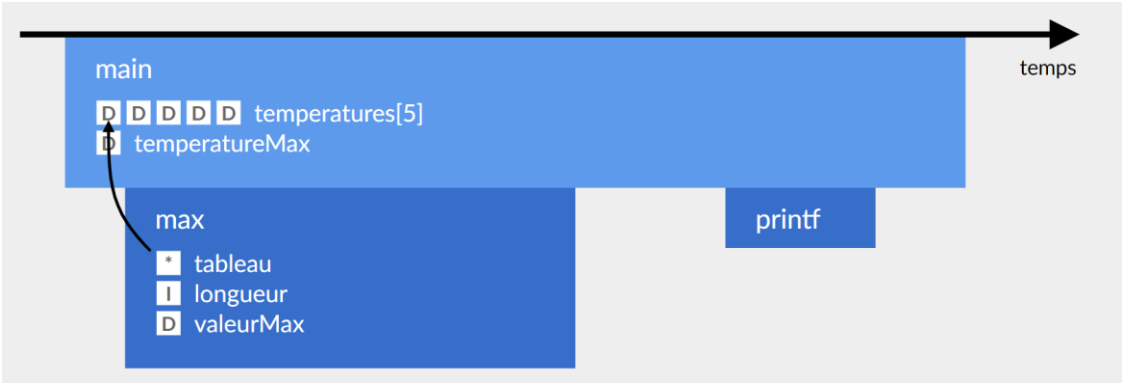
Call stack

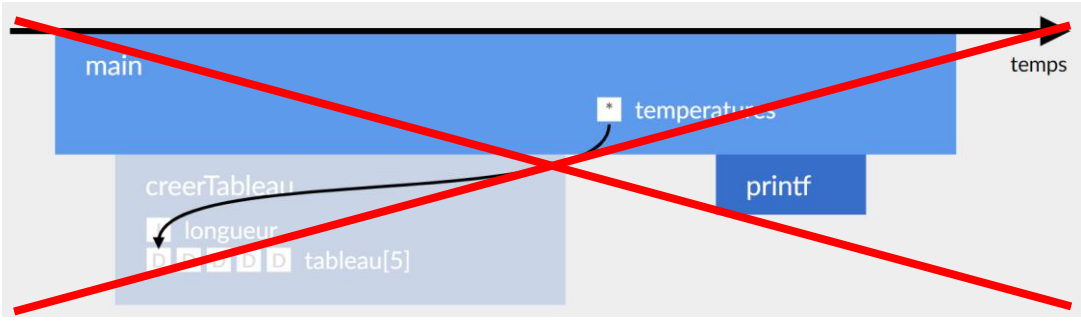
Data structure in memory that manages active functions and flow of execution

```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);
}
```









Three ways to pass arrays or array data out of functions

- use static memory allocation
  - (1) modify value of array passed by reference
  - (2) use *static* declaration and return pointer from function

- use dynamic memory allocation – (3) malloc in function

- declarations
  - static
    - define by size
    - define by values
    - define by size and values
  - dynamic
    - define by size

static (1) modify value of array passed by reference

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

#define NELEM 4

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

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

  return;
}

int main() {

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

  squarearray(NELEM, myarr, out);

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

static (2) use *static* declaration and return pointer from function

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

#define NELEM 4

int *squarearray(int *arr_in) {

  static int arr_out[NELEM];

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

  return arr_out;
}

int main() {

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

  out = squarearray(myarr);

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

  return 0;
}
```

dynamic (3) malloc in function

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

int *squarearray(size_t n, int *arr_in) {

  int *arr_out = malloc(sizeof(int) * n);

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

  return arr_out;
}

int main() {

  int myarr[] = {0, 1, 2, 3};
  int *out;

  // assume we need to estimate the number of elements in the array
  size_t nelem = sizeof(myarr)/sizeof(myarr[0]);
  out = squarearray(nelem, myarr);

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

  free(out);

  return 0;
}
```

you have to use sizeof()
before you pass array to function
why?

Other use of dynamic memory allocation

Example directory structure

```
.
├── csv_example_files
│   ├── example_input.csv
│   └── example_output.csv
├── csv_example
└── csv_example.c
```

Contents of csv_example_files/example_input.csv

```
col1,col2,col3
0.0,0.1,0.2
0.3,0.4,0.5
```

csv_example.csv

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

#define NCOL 3

int main(int argc, char *argv[]) {
  // expected syntax:
  //   ./{executable} inputfile.csv outputfile.csv

  int c = EOF;
  char header[1024];
  int i, nlines;

  /* open files for reading and writing */
  FILE *fp = fopen(argv[1], "r");       // "csv_file_example.csv"
  FILE *fout = fopen(argv[2], "w");     // "csv_file_out.csv"

  /* count number of lines */
  nlines=1;
  while ((c=fgetc(fp)) != EOF) {
    if (c=='\n') {
      nlines++;
    }
  }
  printf("%d lines read\n", nlines);

  /* dynamically allocate array */
  double (*arr)[NCOL] = malloc((nlines-1)*NCOL*sizeof(double));

  /* read file  */
  fseek(fp, 0, SEEK_SET);
  fscanf(fp, "%s", header);
  i = 0;
  while(fscanf(fp, "%lf,%lf,%lf", &arr[i][0], &arr[i][1], &arr[i][2]) == 3) {
    i++;
  }

  /* output  */
  fprintf(fout, "col1_squared,col2_squared,col3_squared\n");
  for(i=0; i<nlines-1; i++) {
    fprintf(fout, "%lf,%lf,%lf\n", pow(arr[i][0],2), pow(arr[i][1],2), pow(arr[i][2],2));
  }

  /* deallocate array */
  free(arr);

  /* close files */
  fclose(fout);
  fclose(fp);

  /* exit function */
  return 0;
}
```

Summary - defining array size

- known at time of compilation
  - use array in global scope
  - use static arrays in functions

- determined at runtime
  - use malloc

Structures in C

Why?

- heterogeneous data
- refer to content by name (more understandable code)

Options:

- structure of array(s) – *pass by value*
- array of structures – *pass by reference*

```c
#include <stdio.h>

// Déclarer une structure
struct Sommet {
    char * nom;
    double latitude;
    double longitude;
    float altitude;
};

int main(int argc, char * argv[]) {
    // Créer une variable et remplir les champs
    struct Sommet cervin;
    cervin.nom = "Matterhorn";
    cervin.latitude = 45.97639;
    cervin.longitude = 7.65833;
    cervin.altitude = 4478;

    // ou créer et remplir en même temps
    struct Sommet montBlanc = {"Mont Blanc", 45.832778, 6.865, 4808};

    // Afficher
    printf("%s: %0.0f m alt.\n", cervin.nom, cervin.altitude);
    printf("%s: %0.0f m alt.\n", montBlanc.nom, montBlanc.altitude);
    return 0;
}
```

```c
struct Sommet sommets[10];

sommets[0].nom = "Matterhorn";
sommets[0].latitude = 45.97639;
sommets[0].longitude = 7.65833;
sommets[0].altitude = 4478;

sommets[1].nom = "Weisshorn";
sommets[1].latitude = 46.101667;
sommets[1].longitude = 7.716111;
sommets[1].altitude = 4506;

...
```

Additional syntactic sugar

when we pass a pointer to the structure (note: default is to pass by value)

```c
#include <stdio.h>

// Déclarer une structure
struct Sommet {
    char * nom;
    double latitude;
    double longitude;
    float altitude;
};

void afficherSommet(struct Sommet sommet, char *units) {
  printf("%s: %0.0f %s alt.\n", sommet.nom, sommet.altitude, units);
}

void corrigerAltitude(struct Sommet * sommet) {
    // Convertir mètres en feet
    sommet->altitude = sommet->altitude / 0.3048;
}

int main(int argc, char * argv[]) {

  // declare
  struct Sommet cervin;

  // initialize with values
  cervin.nom = "Matterhorn";
  cervin.latitude = 45.97639;
  cervin.longitude = 7.65833;
  cervin.altitude = 4478;

  afficherSommet(cervin, "meters");

  corrigerAltitude(&cervin);

  afficherSommet(cervin, "feet");

  return 0;
}
```

Vu que *sommet* est un pointeur, on utilise `->` pour accéder aux valeurs de la structure. Ceci est une syntaxe abrégée pour:

```c
(*sommet).altitude = (*sommet).altitude / 0.3048;
```