

601.220 Intermediate Programming, Spring 2021

Day 5 Arrays and ASCII table

<https://piazza.com/jhu/spring2021/601220>

Outline

- Arrays
- Characters and ASCII table

A few useful linux tips

Several tricks to cut down on the typing

- '*' (asterisk) - wildcard character, helpful with file commands
 - `git add *.c` to add all files ending in `.c` to your staging area
- tab completion - hit a tab as you are typing a name (command, directory or file) and it will complete up to the last unique character
- ! (bang) - repeat the prior command
 - can be used alone or with the start of a command name
 - `> !!` will execute the prior command
 - `> !em` will execute the most recent command that started with "em"
- up and down arrows - cycle through your command history
- `history | grep <keyword>` - search history with keywords

man command

- **man**ual available at the command line
- use with an operation to get all details and options

`man cp`

- can also use with C functions!

`man ispunct`

Array Basics

```
int c[12];
```

An *array* variable is a *collection* of elements laid out consecutively in memory

All elements have the same declared type; `int` in this example

Individual elements accessed with `[]` notation

The actual value of an array variable is a memory address in C, but more on this later...

Array Model

- Illustration of an array declared as `int c[12]` and with particular values

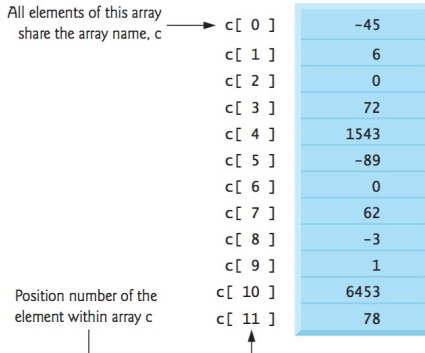


Fig. 6.1 | 12-element array.

Array Declaration & Usage

```
// array_eg_1.c:
#include <stdio.h>
int main() {
    int c[12];
    c[0] = 7; // first element
    c[11] = 1; // last element
    printf("first c=%d, last c=%d\n", c[0], c[11]);
    return 0;
}
```

```
$ gcc array_eg_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
first c=7, last c=1
```

Arrays - wrong declaration

Square brackets go after the variable name, not after the type

- Unlike Java!

```
// array_eg_2.c:
```

```
int main() {  
    int[12] c; // oops  
    return 0;  
}
```

```
$ gcc array_eg_2.c -std=c99 -pedantic -Wall -Wextra
```

```
array_eg_2.c: In function 'main':
```

```
array_eg_2.c:2:8: error: expected identifier or '(' before '[' token
```

```
  2 |      int[12] c; // oops  
    |          ^
```


Array values undefined

Danger: Elements are undefined until explicitly initialized

```
// array_eg_3.c:
#include <stdio.h>
int main() {
    int c[12]; // elements undefined!
    printf("c[0]=%d, c[2]=%d, c[9]=%d\n", c[0], c[2], c[9]);
    return 0;
}
```

```
$ gcc array_eg_3.c -std=c99 -pedantic -Wall -Wextra
```

```
array_eg_3.c: In function 'main':
```

```
array_eg_3.c:4:5: warning: 'c[9]' is used uninitialized in this function
```

```
4 |     printf("c[0]=%d, c[2]=%d, c[9]=%d\n", c[0], c[2], c[9]);
  |     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
array_eg_3.c:4:5: warning: 'c[2]' is used uninitialized in this function
```

```
array_eg_3.c:4:5: warning: 'c[0]' is used uninitialized in this function
```

```
$ ./a.out
```

```
c[0]=947441046, c[2]=-1189641715, c[9]=0
```

Array initialization with loop

```
// array_eg_4.c:
#include <stdio.h>
int main() {
    int c[12]; // elements undefined!
    for(int i = 0; i < 12; i++) {
        c[i] = i; //initialize with value matching index number
    }
    printf("c[4]=%d, c[9]=%d\n", c[4], c[9]);
    return 0;
}
```

```
$ gcc array_eg_4.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
c[4]=4, c[9]=9
```

Array initialization with literal values

Can initialize to a specified sequence of values

Comma separated within { ... }:

```
// array_eg_5.c:
#include <stdio.h>
int main() {
    int c[5] = {2, 4, 6, 8, 10};
    printf("c[1]=%d, c[3]=%d\n", c[1], c[3]);
    return 0;
}
```

```
$ gcc array_eg_5.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
c[1]=4, c[3]=8
```

Initialized Array Sizes

When initializing with { ... }, array size can be omitted

Compiler figures it out for you

```
// array_eg_6.c:
#include <stdio.h>
int main() {
    int c[ ] = {2, 4, 6, 8, 10};
    //      ^ no size
    printf("c[1]=%d, c[3]=%d\n", c[1], c[3]);
    return 0;
}

$ gcc array_eg_6.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
c[1]=4, c[3]=8
```

Arrays working together

```
// array_eg_7.c:
#include <stdio.h>
int main() {
    int data[10] = {2, 1, 1, 1, 2, 0, 1, 2, 1, 0};
    int freq[3] = {0, 0, 0};
    for(int i = 0; i < 10; i++) {
        freq[data[i]]++;
    }
    printf("%d, %d, %d\n", freq[0], freq[1], freq[2]);
    return 0;
}

$ gcc array_eg_7.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
2, 5, 3
```

What would happen if some elements of data were 3?

Whole Array Operations (NOT)

- Can't assign one array to another using =
 - Need loop to copy elements from one array to another
- Unlike Python, no "slicing" in C
 - E.g. can't access several elements at once using `ra[1:4]`
- Can't print an entire array (except char arrays which are strings)
 - E.g. no `printf("%a", ra);`
- Can't read an entire array (except char arrays which are strings)
 - E.g. no `scanf("%a", ra);`

Zoom poll!

What output is printed by the following program?

```
#include <stdio.h>
int main(void) {
    int a[] = { 6, 8, 5 };
    int sum = 0;
    for (int i = 1; i <= 3; i++) {
        sum += a[i];
    }
    printf("sum=%d\n", sum);
    return 0;
}
```

- A. 0
- B. 13
- C. 19
- D. Some other specific integer value
- E. Impossible to predict

More on characters

We said a `char` variable holds a single character

- `char digit = '4';`
- `char bang = '!';`
- These *must* be single quotes; double quotes are for strings only

Behind the scenes, `char` is much like `int`

- This is valid: `char digit = '4' - 1;`
- `digit` now contains the character `'3'`

`printf` and `scanf` format string for `char` is `%c`

ASCII

- ASCII or a similar standard governs the mapping between characters and integers

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SoH	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	SoTxt	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	EoTxt	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EoT	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enq	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Ack	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Bsp	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	HTab	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LFfeed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VTab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FFfeed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SOut	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SIn	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAck	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Syn	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	EoTB	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Can	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EoM	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Sub	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Esc	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FSep	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GSep	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RSep	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	USEp	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		Delete

charstable.com

char/int conversion example

```
// convert_digit_0.c:
#include <stdio.h>

// Convert decimal character into corresponding int
int main() {
    char char_0 = '0';
    int int_0 = char_0 - '0';
    printf("Character printed as character: %c\n", char_0);
    printf("Character printed as integer: %d\n", char_0);
    printf("Integer printed as integer: %d\n", int_0);
}
```

```
$ gcc convert_digit_0.c -std=c99 -pedantic -Wall -Wextra
```

```
$ ./a.out
```

```
Character printed as character: 0
```

```
Character printed as integer: 48
```

```
Integer printed as integer: 0
```

another char/int conversion example

```
// convert_digit_7.c:  
#include <stdio.h>  
  
// Convert decimal character into corresponding int  
int main() {  
    char char_7 = '7';  
    int int_7 = char_7 - '0';  
    printf("Character printed as character: %c\n", char_7);  
    printf("Character printed as integer: %d\n", char_7);  
    printf("Integer printed as integer: %d\n", int_7);  
}
```

```
$ gcc convert_digit_7.c -std=c99 -pedantic -Wall -Wextra
```

```
$ ./a.out
```

```
Character printed as character: 7
```

```
Character printed as integer: 55
```

```
Integer printed as integer: 7
```