601.220 Intermediate Programming

Arrays and ASCII table

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

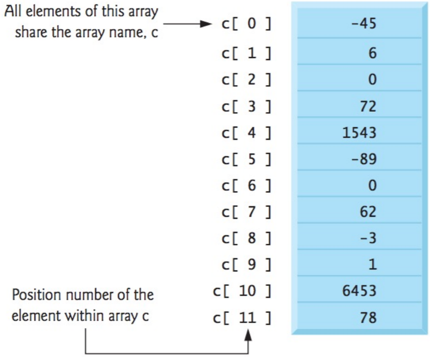
All elements have the same declared type; `int` in this example

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



All elements of this array share the array name, c

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

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!

```c
// 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

```c
// 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 functio
    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 functio
array_eg_3.c:4:5: warning: ‘c[0]’ is used uninitialized in this functio
$ ./a.out
c[0]=393223158, c[2]=-2046139891, 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

```c
// 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);

Checkpoint Question!

What output is printed by the following program?

```c
#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

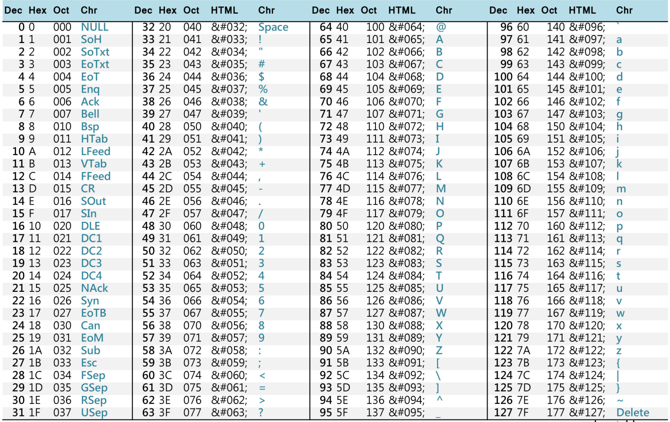
Behind the scenes, `char` is much like `int`

- 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	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	SoH	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	SoTxt	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	EoTxt	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	EoT	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enq	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Ack	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Bsp	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	HTab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	LFeed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	VTab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	FFeed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	CR	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	SOut	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	SIn	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	DLE	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	DC1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	DC2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	DC3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	DC4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	NAck	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Syn	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	EoTB	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Can	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	EoM	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Sub	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Esc	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	FSep	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	GSep	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	RSep	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	USep	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Delete

charstable.com

char/int conversion example

```c
// 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

```c
// 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
```