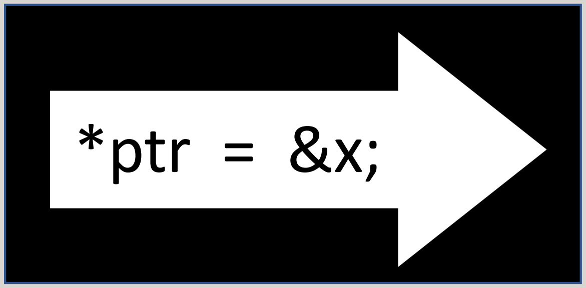


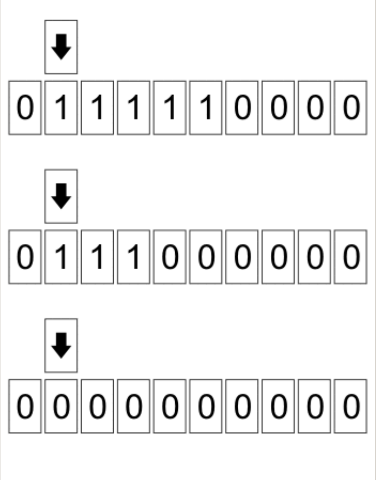
```
*ptr  =  &x;
```

Introduction to Pointers

Michael Liut

Mathematical & Computational Sciences

University of Toronto Mississauga



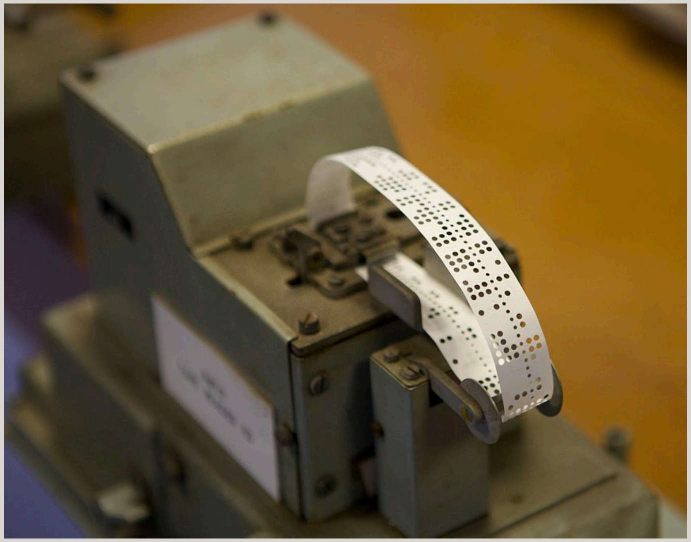
0 1 1 1 1 1 0 0 0 0

0 1 1 1 0 0 0 0 0 0

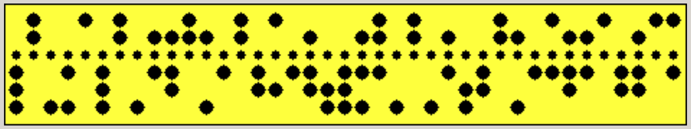
0 0 0 0 0 0 0 0 0 0

Background (Recall: Memory)

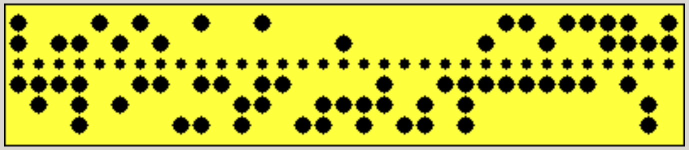
- Anything that can store 0s and 1s can be memory.
- Think of a tape! A tape of memory (cells)!
- We are going to simplify the tape on the right by looking at blocks of memory.



5-hole Punch Tape connected in a physical loop



“Mathematical and Computational Sciences”

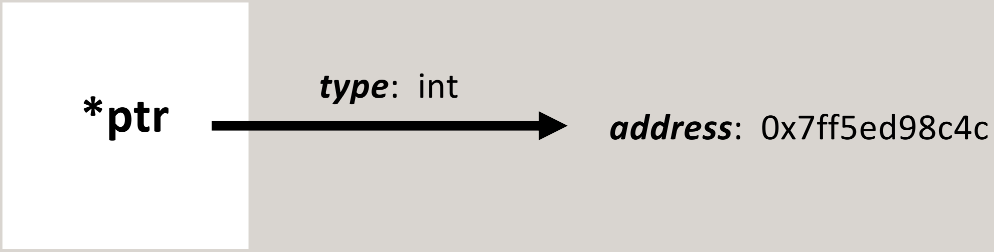


“University of Toronto Mississauga”

What are pointers?

- A means to reference memory.
- Pointers have **two** important pieces of information:
  1. The ***address*** of some “cell” of memory
  2. The ***type*** of that entity

Let’s call our sample pointer ***ptr**
and assume that it is of integer type:

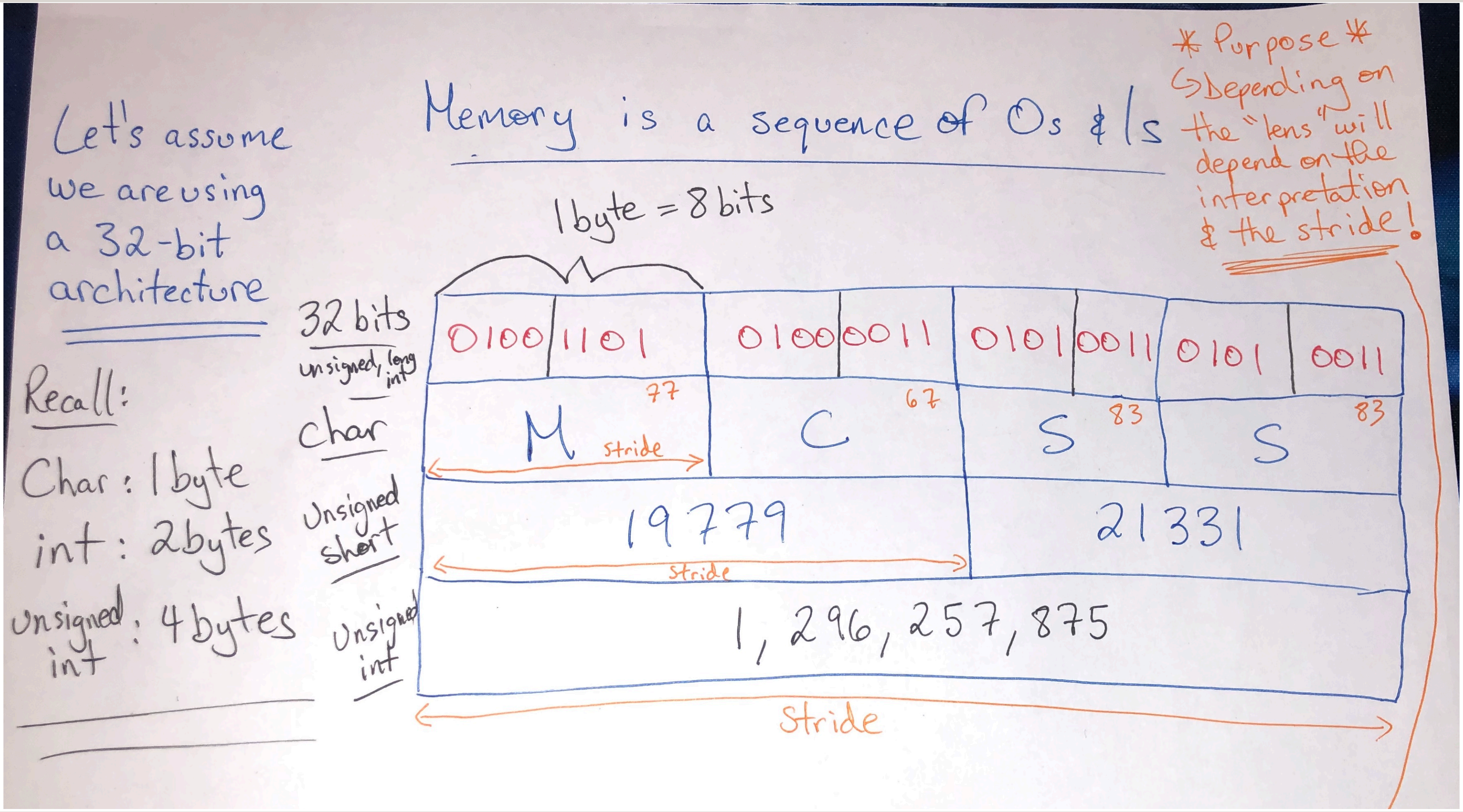


***ptr**

type: int

address: 0x7ff5ed98c4c

Memory and Pointers



Let's assume we are using a 32-bit architecture

Recall:

Char : 1 byte

int : 2bytes

unsigned int : 4 bytes

Memory is a sequence of 0s & 1s

1 byte = 8 bits

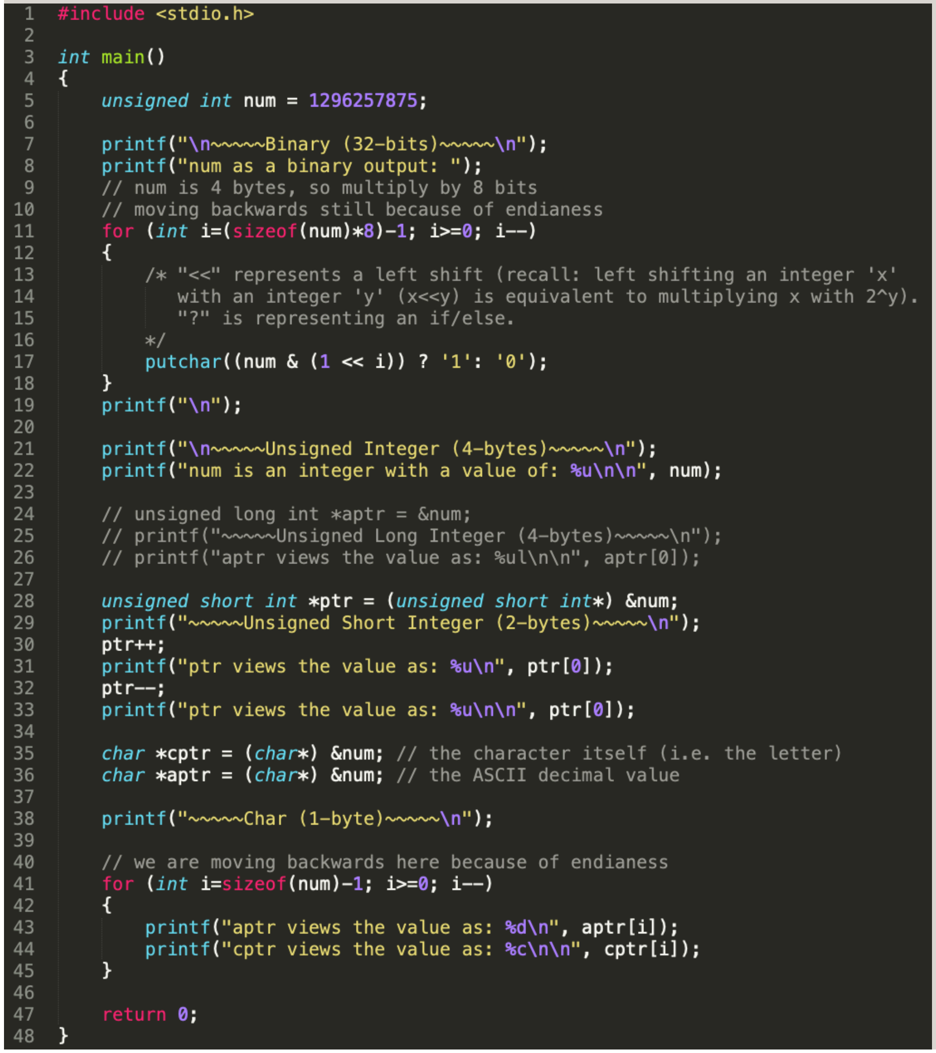
* Purpose *

↳ Depending on the "lens" will depend on the interpretation & the stride!

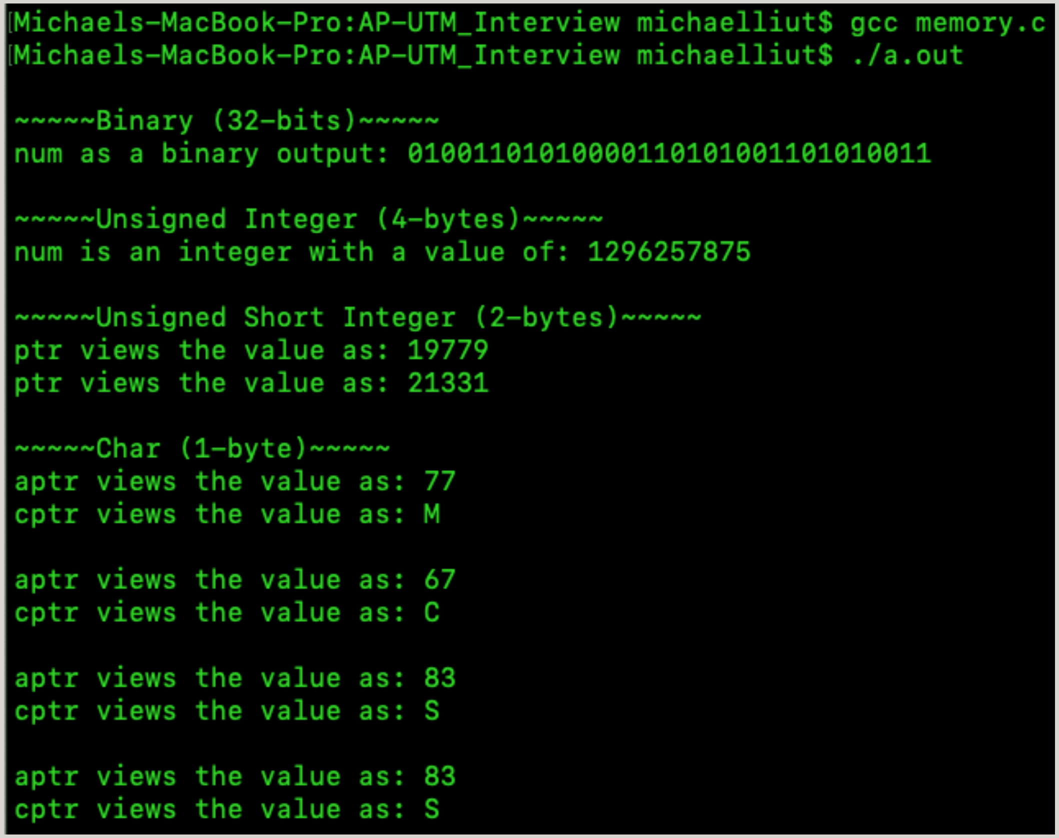
Let’s construct a C program!

It’s coding time! ☺

DEMO of C program



```c
#include <stdio.h>

int main()
{
    unsigned int num = 1296257875;

    printf("\n~~~~~Binary (32-bits)~~~~~\n");
    printf("num as a binary output: ");
    // num is 4 bytes, so multiply by 8 bits
    // moving backwards still because of endianess
    for (int i=(sizeof(num)*8)-1; i>=0; i--)
    {
        /* "<<" represents a left shift (recall: left shifting an integer 'x'
           with an integer 'y' (x<<y) is equivalent to multiplying x with 2^y).
           "?" is representing an if/else.
        */
        putchar((num & (1 << i)) ? '1': '0');
    }
    printf("\n");

    printf("\n~~~~~Unsigned Integer (4-bytes)~~~~~\n");
    printf("num is an integer with a value of: %u\n\n", num);

    // unsigned long int *aptr = &num;
    // printf("~~~~~Unsigned Long Integer (4-bytes)~~~~~\n");
    // printf("aptr views the value as: %ul\n\n", aptr[0]);

    unsigned short int *ptr = (unsigned short int*) &num;
    printf("~~~~~Unsigned Short Integer (2-bytes)~~~~~\n");
    ptr++;
    printf("ptr views the value as: %u\n", ptr[0]);
    ptr--;
    printf("ptr views the value as: %u\n\n", ptr[0]);

    char *cptr = (char*) &num; // the character itself (i.e. the letter)
    char *aptr = (char*) &num; // the ASCII decimal value

    printf("~~~~~Char (1-byte)~~~~~\n");

    // we are moving backwards here because of endianess
    for (int i=sizeof(num)-1; i>=0; i--)
    {
        printf("aptr views the value as: %d\n", aptr[i]);
        printf("cptr views the value as: %c\n\n", cptr[i]);
    }

    return 0;
}
```

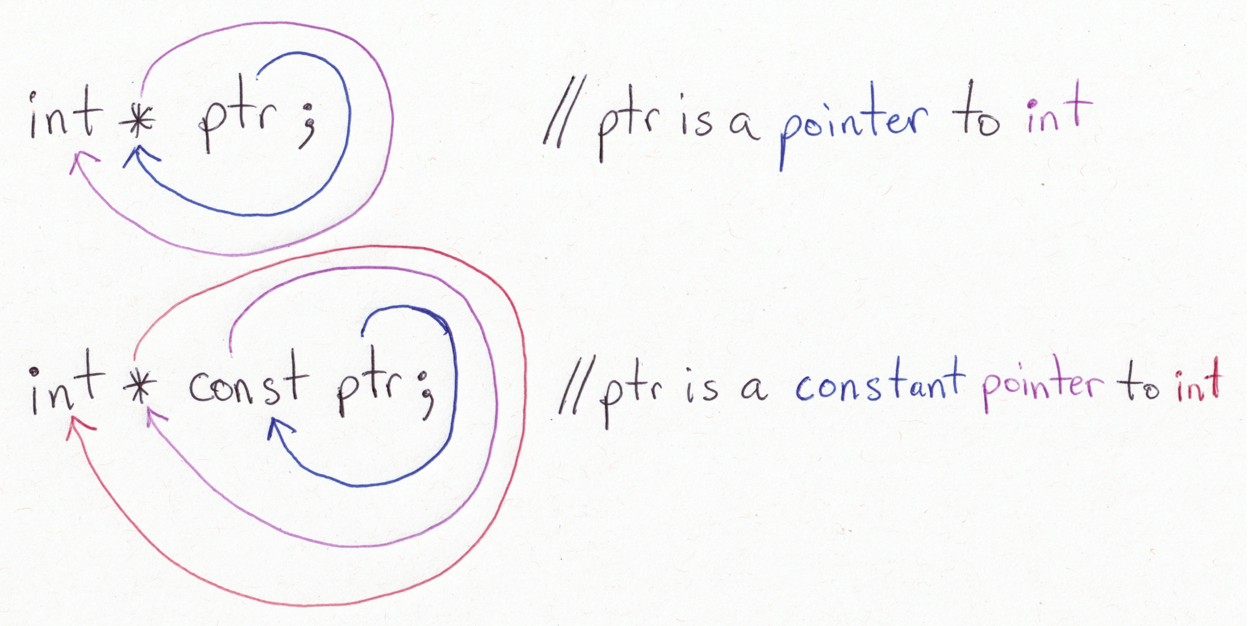


```
[Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ gcc memory.c
[Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ ./a.out

~~~~~Binary (32-bits)~~~~~
num as a binary output: 01001101010000110101001101010011

~~~~~Unsigned Integer (4-bytes)~~~~~
num is an integer with a value of: 1296257875

~~~~~Unsigned Short Integer (2-bytes)~~~~~
ptr views the value as: 19779
ptr views the value as: 21331

~~~~~Char (1-byte)~~~~~
aptr views the value as: 77
cptr views the value as: M

aptr views the value as: 67
cptr views the value as: C

aptr views the value as: 83
cptr views the value as: S

aptr views the value as: 83
cptr views the value as: S
```

This is a live demo and discussion!

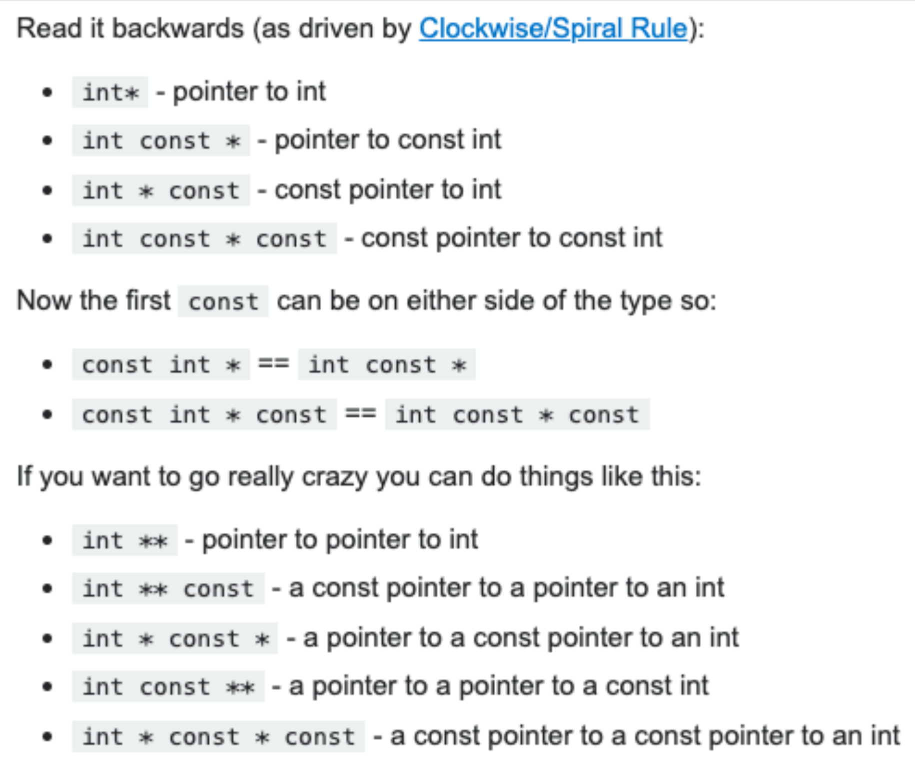
Clockwise/Spiral Rule

- Start from the name of the variable, move ‘clockwisely’ to the next pointer or type. Repeat until the expression ends.



```
int * ptr;          //ptr is a pointer to int

int * const ptr;    //ptr is a constant pointer to int
```

const int * vs. int * const



Read it backwards (as driven by Clockwise/Spiral Rule):

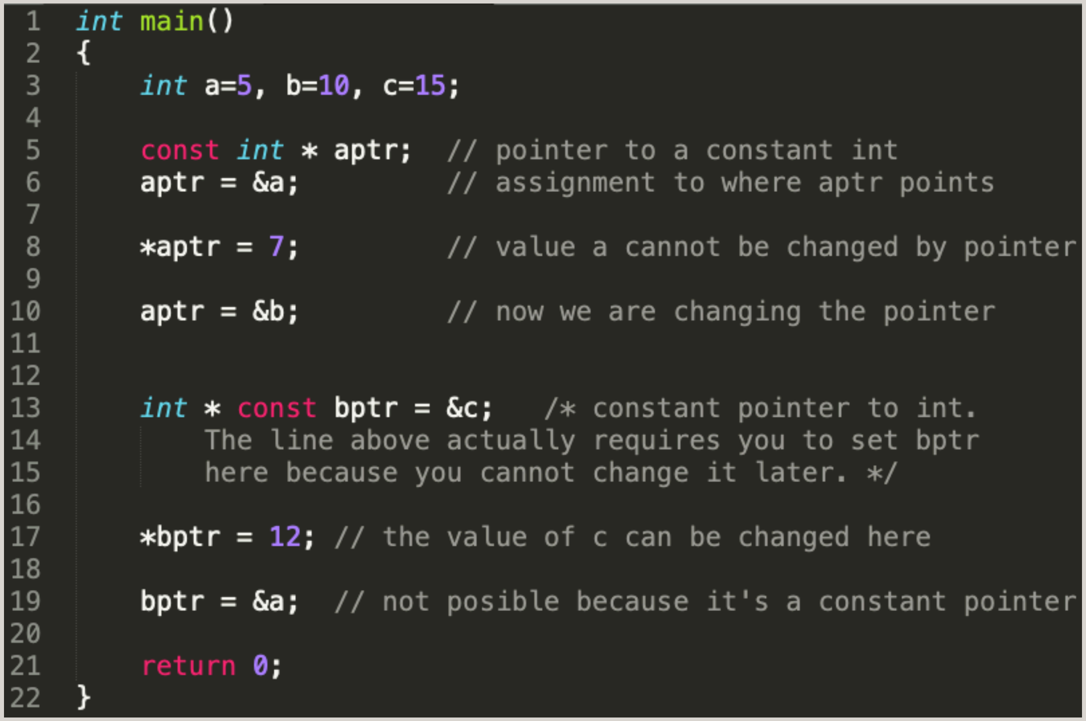
- `int*` - pointer to int
- `int const *` - pointer to const int
- `int * const` - const pointer to int
- `int const * const` - const pointer to const int

Now the first `const` can be on either side of the type so:

- `const int *` == `int const *`
- `const int * const` == `int const * const`

If you want to go really crazy you can do things like this:

- `int **` - pointer to pointer to int
- `int ** const` - a const pointer to a pointer to an int
- `int * const *` - a pointer to a const pointer to an int
- `int const **` - a pointer to a pointer to a const int
- `int * const * const` - a const pointer to a const pointer to an int



```
int main()
{
    int a=5, b=10, c=15;

    const int * aptr;  // pointer to a constant int
    aptr = &a;         // assignment to where aptr points

    *aptr = 7;         // value a cannot be changed by pointer

    aptr = &b;         // now we are changing the pointer


    int * const bptr = &c;   /* constant pointer to int.
        The line above actually requires you to set bptr
        here because you cannot change it later. */

    *bptr = 12; // the value of c can be changed here

    bptr = &a;  // not posible because it's a constant pointer

    return 0;
}
```

Why do we care about pointers?

Benefits:

- **Flexibility**: we can pass by reference (we don't need to replicate the data passed).
- **Power**: permits variable-sized data structures, allocating more memory as needed.
- **Efficiency**: reduces overhead and increases the execution speed of programs (no copying needed ☺).

Programmer
I think I forgot something.
char *
If you forgot, then it wasn't important.
Programmer
malloc()

Who needs pointers?

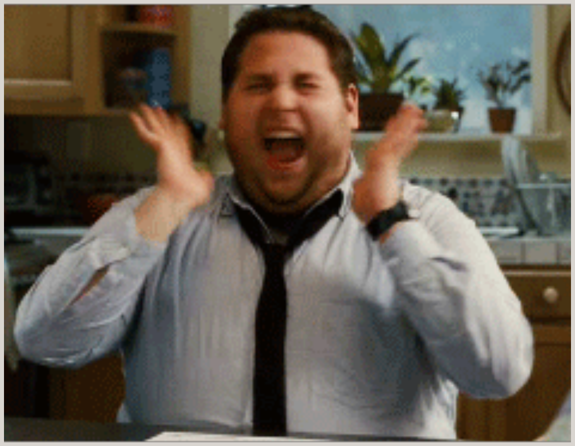
- Those who want a quick means to access/manipulate data!

For example:

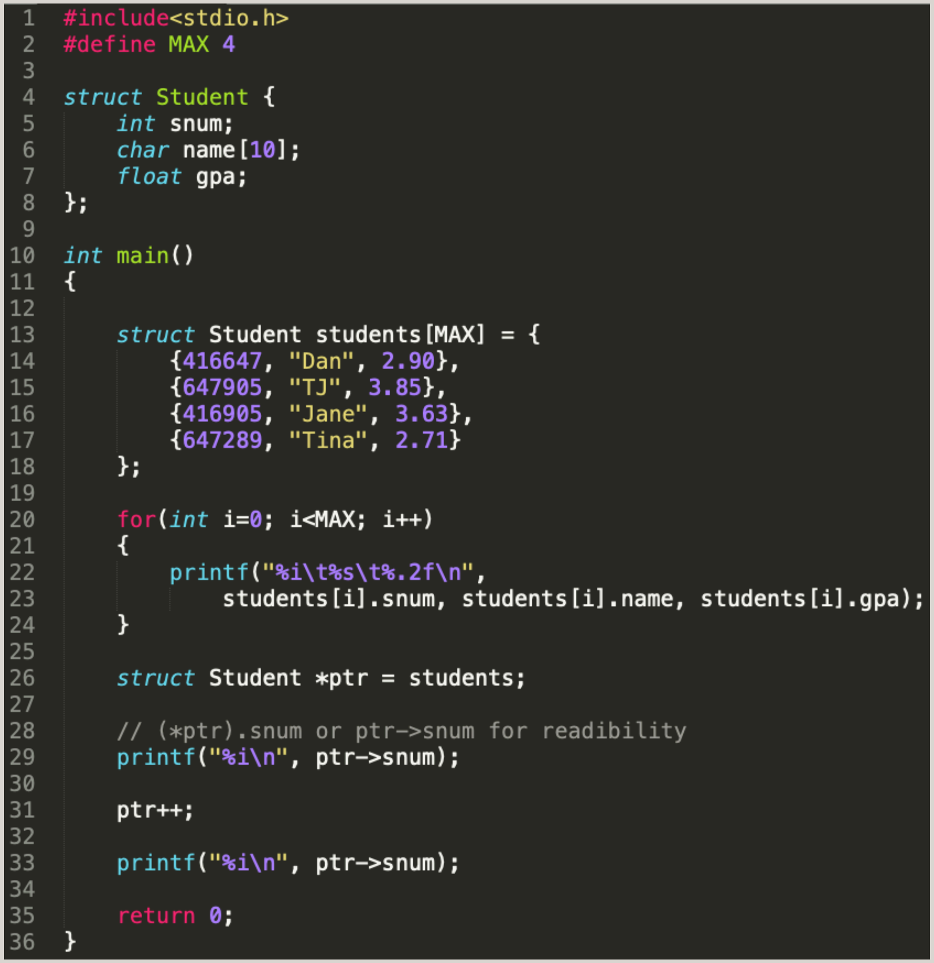
  - Cryptography
  - Data Compression
  - Bioinformatics

Let’s construct another C program!

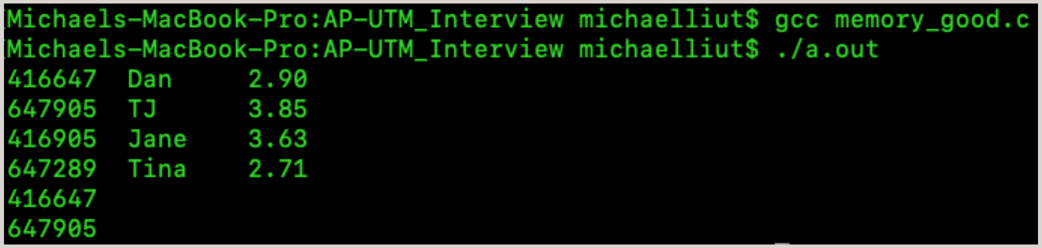
It’s coding time, again! ☺



DEMO of C program #2



```c
#include<stdio.h>
#define MAX 4

struct Student {
    int snum;
    char name[10];
    float gpa;
};

int main()
{

    struct Student students[MAX] = {
        {416647, "Dan", 2.90},
        {647905, "TJ", 3.85},
        {416905, "Jane", 3.63},
        {647289, "Tina", 2.71}
    };

    for(int i=0; i<MAX; i++)
    {
        printf("%i\t%s\t%.2f\n",
            students[i].snum, students[i].name, students[i].gpa);
    }

    struct Student *ptr = students;

    // (*ptr).snum or ptr->snum for readibility
    printf("%i\n", ptr->snum);

    ptr++;

    printf("%i\n", ptr->snum);

    return 0;
}
```



```
Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ gcc memory_good.c
Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ ./a.out
416647  Dan     2.90
647905  TJ      3.85
416905  Jane    3.63
647289  Tina    2.71
416647
647905
```

Thanks for listening! ☺

- I'd like to present some exercises that I believe would make up a regular "1-hour" class session before questions.

This would extend into some active work

Example 1

- Given some sample code fragments, have students show how to actually demonstrate the declarations.

- Given certain declarations, have students draw the clockwise/spiral rule and write the backwards reading.
  - Assuming that there was time to reach this in the first class on *pointers*.

This would extend into some active work

Example 2

- Have students write a program that allocates an array of integers in the main function and passes that array to a function, computing something and returning it.

- Further, extend exercise, by having them illustrate the view of memory immediately before and after this function call.
  - This is similar to what they do in CSC-148 and resonates well with students.

This would extend into some active work

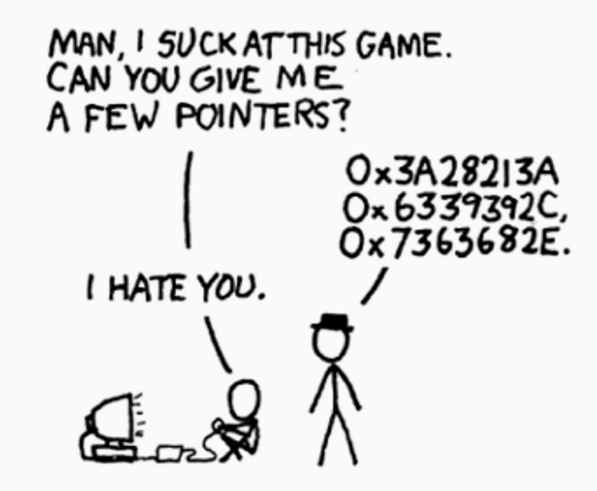
Example 3

- Have them implement a LinkedList, equivalent to that in CSC-148, but now with pointers rather than objects.
  - Discuss the differences between contiguous memory and fragmented memory.

- Further, extend exercise, by having students perform the merging of two LinkedList into a third LinkedList ensuring no duplicates are added.

Thanks for listening! ☺

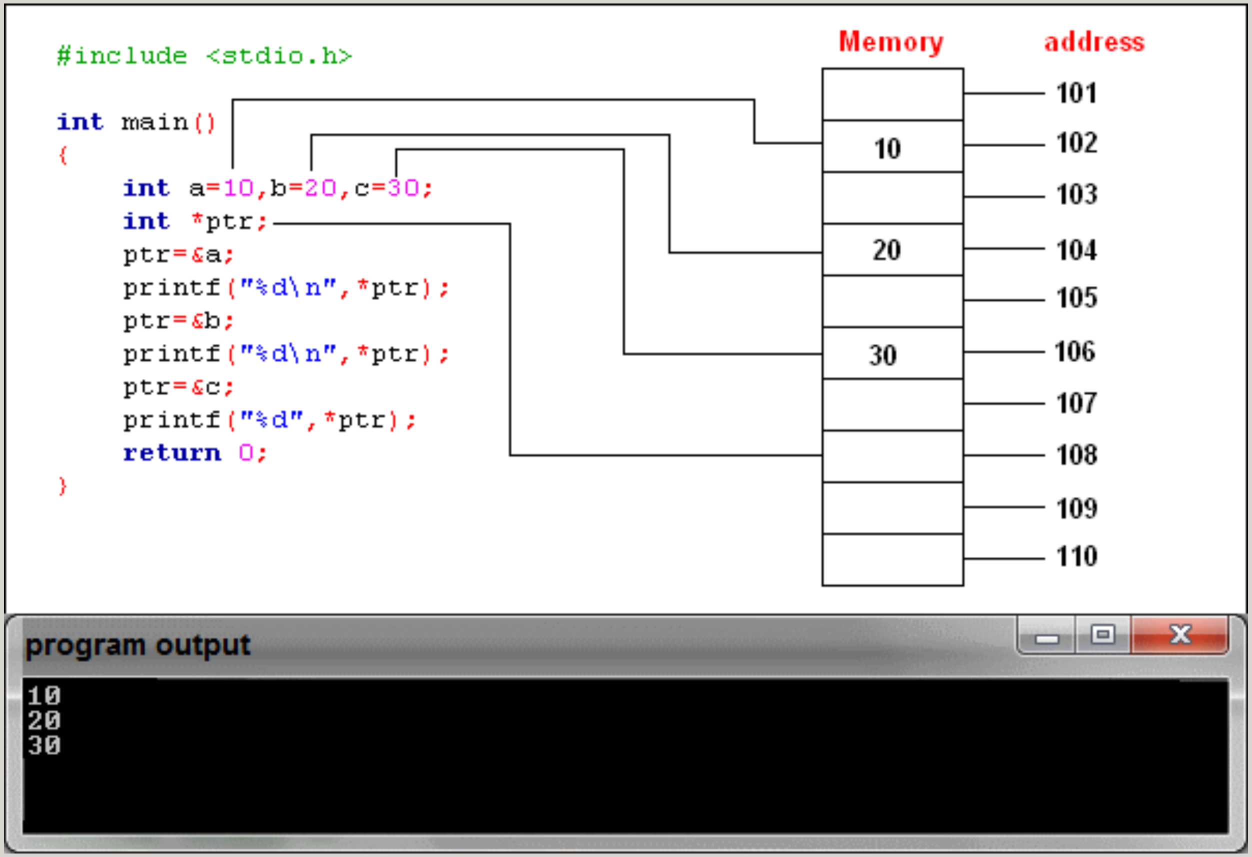
- Does anyone have any questions?


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

#pineapples ripen faster upside down

Supplementary Slides

Let’s look at a simple C program

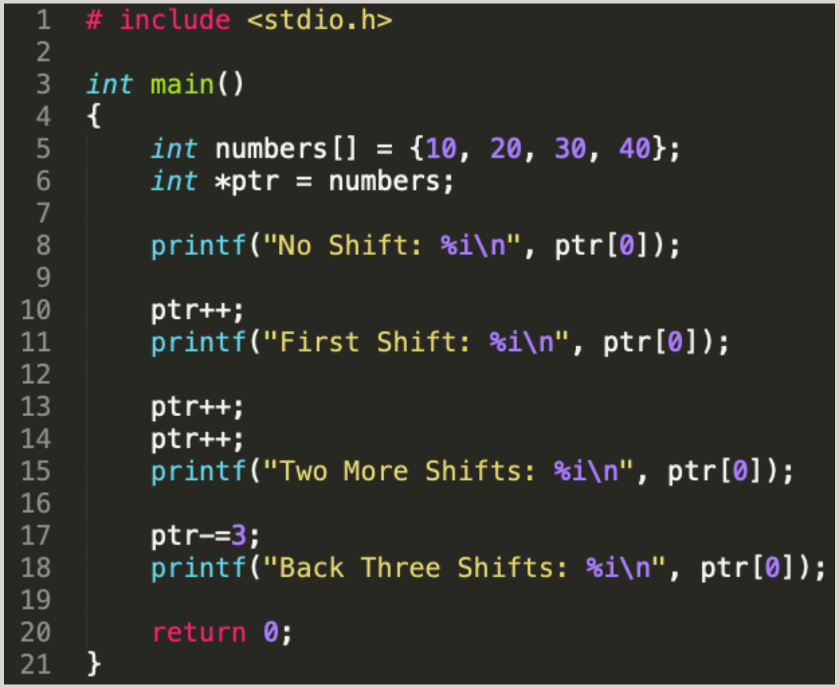


```c
#include <stdio.h>

int main()
{
     int a=10,b=20,c=30;
     int *ptr;
     ptr=&a;
     printf("%d\n",*ptr);
     ptr=&b;
     printf("%d\n",*ptr);
     ptr=&c;
     printf("%d",*ptr);
     return 0;
}
```

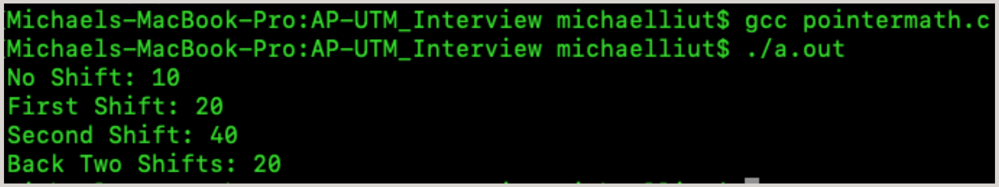
```
10
20
30
```

Pointer Arithmetic

- You may perform arithmetic operations on a pointer, changing the location which it starts to looks at the data.

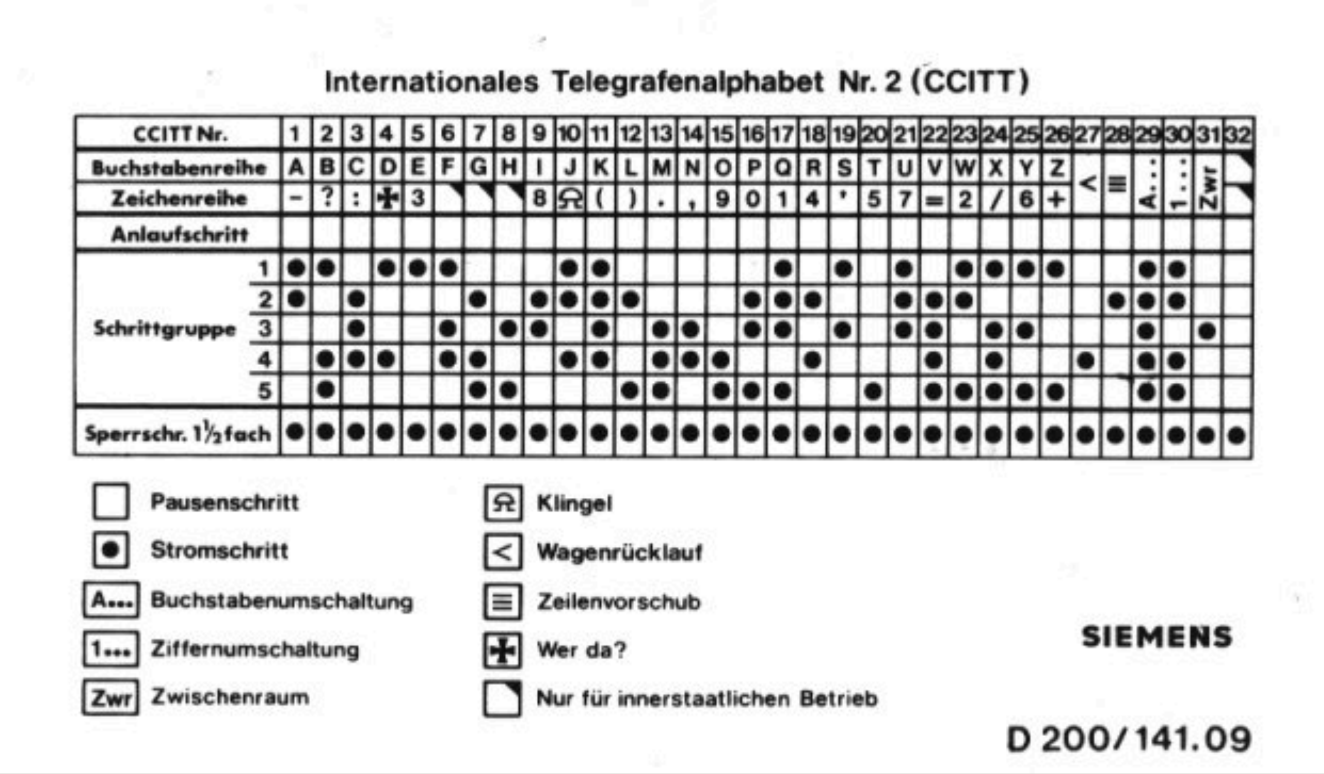


```c
# include <stdio.h>

int main()
{
    int numbers[] = {10, 20, 30, 40};
    int *ptr = numbers;

    printf("No Shift: %i\n", ptr[0]);

    ptr++;
    printf("First Shift: %i\n", ptr[0]);

    ptr++;
    ptr++;
    printf("Two More Shifts: %i\n", ptr[0]);

    ptr-=3;
    printf("Back Three Shifts: %i\n", ptr[0]);

    return 0;
}
```



```
Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ gcc pointermath.c
Michaels-MacBook-Pro:AP-UTM_Interview michaelliut$ ./a.out
No Shift: 10
First Shift: 20
Second Shift: 40
Back Two Shifts: 20
```

International Telegraph Alphabet No. 2 (ITA2 Telex Code) aka CCITT No. 2



Internationales Telegrafenalphabet Nr. 2 (CCITT)

CCITT Nr.		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Buchstabenreihe		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	<	≡	A...	1...	Zwr	◥
Zeichenreihe		–	?	:	✠	3	◥	◥	◥	8	🔔	(	)	.	,	9	0	1	4	'	5	7	=	2	/	6	+						◥
Anlaufschritt																																	
Schrittgruppe	1	●	●		●	●	●				●	●						●		●		●		●	●	●	●			●	●		
	2	●		●				●		●	●	●	●				●	●	●			●	●	●					●	●	●		
	3			●			●		●	●		●		●	●		●	●		●		●	●		●	●				●		●	
	4		●	●	●		●	●			●	●		●	●	●			●				●		●			●		●	●		
	5		●					●	●				●	●		●	●	●			●		●	●	●	●	●			●	●		
Sperrschr. 1½fach		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

□ Pausenschritt

● Stromschritt

A... Buchstabenumschaltung

1... Ziffernumschaltung

Zwr Zwischenraum

🔔 Klingel

< Wagenrücklauf

≡ Zeilenvorschub

✠ Wer da?

◥ Nur für innerstaatlichen Betrieb

SIEMENS

D 200/141.09

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	