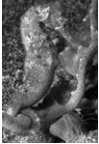
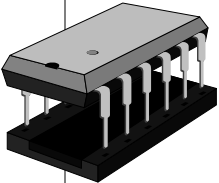
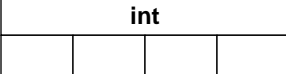
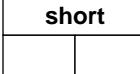
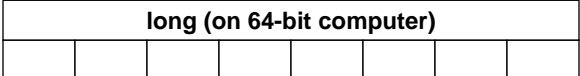
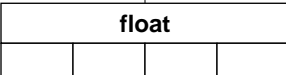
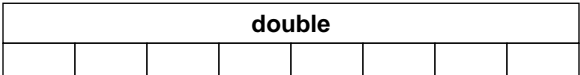



		<h1>CS-034</h1> <h2>Continuing with C</h2>
		
	<p>Thomas Doeppner Pascal Van Hentenryck</p>	
2/9/2005	CS-034: Lecture 2 (twd/pvh)	1

	<h1>Memory</h1>								
	<table border="1"><tr><td>0</td></tr><tr><td>1</td></tr><tr><td>2</td></tr><tr><td>·</td></tr><tr><td>·</td></tr><tr><td>·</td></tr><tr><td>1,073,741,823</td></tr></table>	0	1	2	·	·	·	1,073,741,823	
0									
1									
2									
·									
·									
·									
1,073,741,823									
2/9/2005	CS-034: Lecture 2 (twd/pvh)	2							

Basic Data Types

int  $-2,147,483,648 - 2,147,483,647$	short  $-32,768 - 32,767$
long (on 64-bit computer)  $-9,223,372,036,854,775,808 - 9,223,372,036,854,775,807$	
float  $\sim 10e-44.85 - \sim 10e38.53$, 23-bit mantissa	double  $\sim 10e-323.3 - \sim 10e308.3$, 52-bit mantissa
char  $-128 - 127$	

2/9/2005
CS-034: Lecture 2 (twd/pvh)
3

Characters

- **ASCII**
 - American Standard Code for Information Interchange
 - works for:
 - English
 - not much else
 - Swahili
 - doesn't work for:
 - French
 - Arabic
 - Dutch
 - Sanskrit
 - Spanish
 - Chinese
 - German
 - pretty much everyone else

2/9/2005
CS-034: Lecture 2 (twd/pvh)
4



Who cares!!

2/9/2005 CS-034: Lecture 2 (twd/pvh) 5



**You should care ...
(but not in this course)**

2/9/2005 CS-034: Lecture 2 (twd/pvh) 6

ASCII Character Set

	00	10	20	30	40	50	60	70	80	90	100	110	120
0:	\0	\n		(2	<	F	P	Z	d	n	x	
1:		\v)	3	=	G	Q	[e	o	y	
2:		\f	sp	*	4	>	H	R	\	f	p	z	
3:		\r	!	+	5	?	I	S]	g	q	{	
4:			"	,	6	@	J	T	^	h	r		
5:			#	-	7	A	K	U	_	i	s	}	
6:			\$.	8	B	L	V	`	j	t	~	
7:	\a		%	/	9	C	M	W	a	k	u	DEL	
8:	\b		&	0	:	D	N	X	b	l	v		
9:	\t		'	1	;	E	O	Y	c	m	w		

2/9/2005

CS-034: Lecture 2 (twd/pvh)

7

chars as Integers

```

char tolower(char c) {
    if (c >= 'A' && c <= 'Z')
        return c + 'a' - 'A';
    else
        return c;
}

```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

8

Character Strings

```
char c = 'a';
```

c:

a

```
char *s = "string";
```

s:

○

s	t	r	i	n	g	\0
---	---	---	---	---	---	----

2/9/2005 CS-034: Lecture 2 (twd/pvh) 9

Quiz (1)

Is there any difference between *c1* and *c2* in the following?

```
char c1 = 'a';  
char *c2 = "a";
```

2/9/2005 CS-034: Lecture 2 (twd/pvh) 10

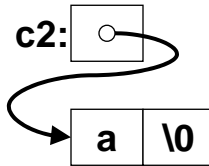
Answer (1)

Yes!!

```
char c1 = 'a';
```

c1: a

```
char *c2 = "a";
```



2/9/2005

CS-034: Lecture 2 (twd/pvh)

11

Quiz (2)

What do `s1` and `s2` refer to after the following is executed?

```
char *s1 = "abcd";
```

```
char *s2 = s1;
```

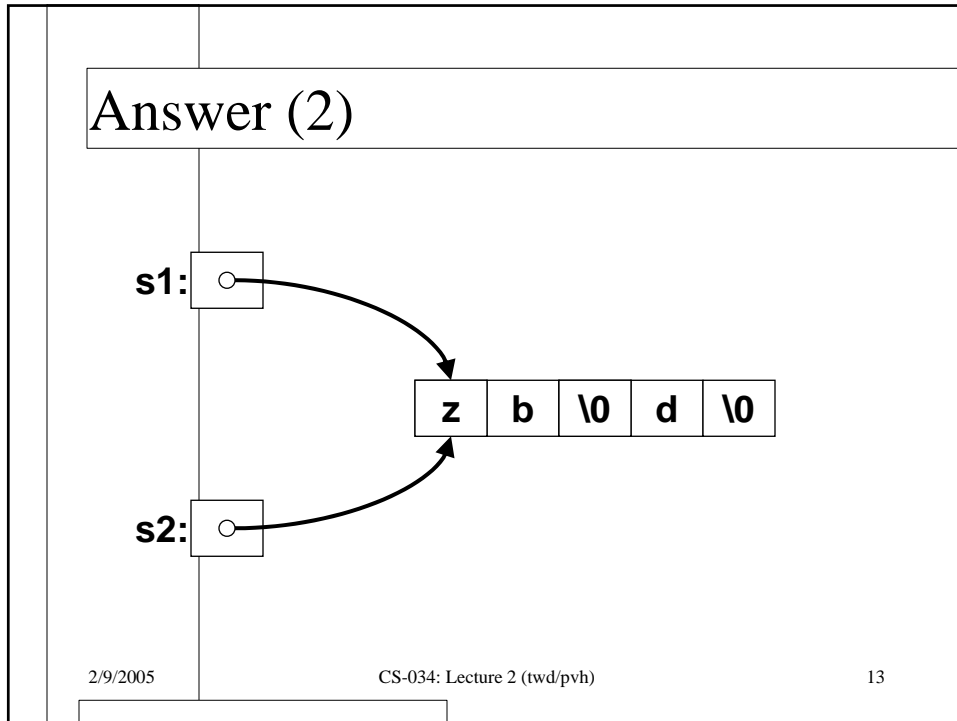
```
s1[0] = 'z';
```

```
s1[2] = '\0';
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

12



Structures

```
struct ComplexNumber {  
    float real;  
    float imag;  
};  
  
struct complex x;  
x.real = 1.4;  
x.imag = 3.65e-10;
```

2/9/2005 CS-034: Lecture 2 (twd/pvh) 14

structs and Functions

```
struct ComplexNumber ComplexAdd(  
    struct ComplexNumber a1,  
    struct ComplexNumber a2) {  
    struct ComplexNumber result;  
    result.real = a1.real + a2.real;  
    result.imag = a1.imag + a2.imag;  
    return result;  
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

15

structs as Arguments

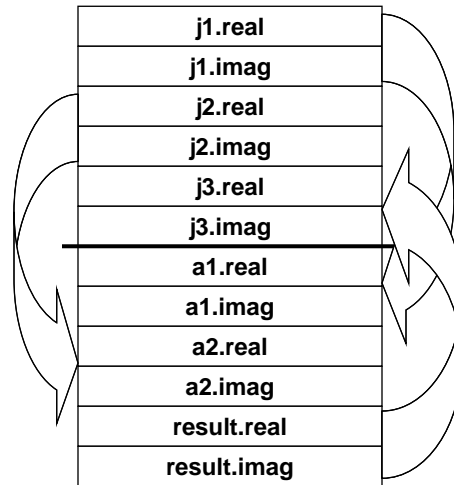
```
struct ComplexNumber j1 = {3.6, 2.125};  
struct ComplexNumber j2 = {4.32, 3.1416};  
struct ComplexNumber j3;  
  
j3 = ComplexAdd(j1, j2);
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

16

The Stack ...



2/9/2005

CS-034: Lecture 2 (twd/pvh)

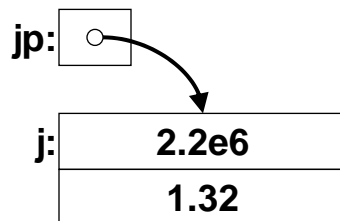
17

Pointers to *structs*

```

struct ComplexNumber j;
struct ComplexNumber *jp = &j;
jp->real = 2.2e6;
jp->imag = 1.32;

```



2/9/2005

CS-034: Lecture 2 (twd/pvh)

18

Passing *structs* More Cheaply ...

```
void ComplexAdd(  
    struct ComplexNumber *a1,  
    struct ComplexNumber *a2,  
    struct ComplexNumber *result) {  
    result->real = a1->real + a2->real;  
    result->imag = a1->imag + a2->imag;  
    return;  
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

19

Using It ...

```
struct ComplexNumber j1 = {3.6, 2.125};  
struct ComplexNumber j2 = {4.32, 3.1416};  
struct ComplexNumber j3;  
  
ComplexAdd(&j1, &j2, &j3);
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

20

The Stack ...

The diagram illustrates the stack memory layout. It shows a vertical stack of memory cells. The top of the stack contains the following variables: `j1.real`, `j1.imag`, `j2.real`, `j2.imag`, `j3.real`, `j3.imag`, `a1`, `a2`, and `result`. Arrows indicate that the stack grows downwards, with the top of the stack being at a higher memory address and the bottom at a lower memory address.

2/9/2005 CS-034: Lecture 2 (twd/pvh) 21

Arrays of *structs*

```
struct ComplexNumber j[10];  
j[0].real = 8.127649;  
j[0].imag = 1.76e18;
```

2/9/2005 CS-034: Lecture 2 (twd/pvh) 22

Arrays, Pointers, and *structs*

```

/* What's this? */
struct ComplexNumber *jp[10];

struct ComplexNumber j0;
jp[0] = &j0;
jp[0]->real = 13.6;

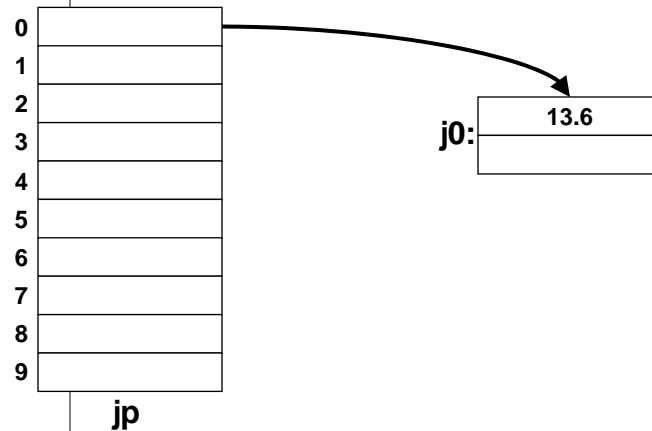
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

23

Memory View



2/9/2005

CS-034: Lecture 2 (twd/pvh)

24

What's Your Sign?

- **Default: signed integers**
- **Unsigned integers: no negatives**
 - unsigned int
 - $0 - 2^{32}-1$ (4,294,967,295)
 - unsigned short
 - $0 - 2^{16}-1$ (65,535)
 - unsigned char
 - $0 - 2^8-1$ (255)

2/9/2005

CS-034: Lecture 2 (twd/pvh)

25

Why Care?

- **You need twice as many positive values**
- **Negative values don't make sense**
 - a negative size?!
- **It's not really supposed to be a number**
 - a bit string

2/9/2005

CS-034: Lecture 2 (twd/pvh)

26

Numeric Conversions

```
short a;  
int b;  
float c;  
b = a; /* always works */  
a = b; /* sometimes works */  
c = b; /* sort of works */  
b = c; /* sometimes works */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

27

Explicit Conversions: Casts (1)

```
float x, y=2.0;  
int i=1, j=2;  
  
x = i/j + y;  
/* what's the value of x? */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

28

Explicit Conversions: Casts (2)

```
float x, y=2.0;
int i=1, j=2;
float a, b;

a = i;
b = j;
x = a/b + y;
/* now what's the value of x? */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

29

Explicit Conversions: Casts (3)

```
float x, y=2.0;
int i=1, j=2;

x = (float)i/(float)j + y;
/* and now what's the value of x? */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

30

Bitwise Operations

```
int x;
int y;
x = y & 0xff000;
/* extract the middle bits */
x = x | 0731;
/* turn on certain bits */
x = x ^ y;
/* compute the bitwise exclusive or */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

31

More Bitwise Operations

```
int x;
int y;

x = x << 3;
/* left shift 3 bits (multiply by 8?) */
y = x >> 4;
/* right shift 4 bits (divide by 16) */
x = ~017;
/* x is set to the one's complement of
   octal 17 (0xffffffff0) */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

32

Bit Strings

- **Use bits to represent status**

```
char device = 0;
const char PowerOn = 0x1;
const char Ready = 0x2;
const char Go = 0x4;
const char Line1 = 0x8;
const char Line2 = 0x10;
const char Line3 = 0x20;
const char Line4 = 0x40;
const char Line5 = 0x80;
device = PowerOn|Ready;
device = device | Go | Line5;
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

33

Transition

- **Things have been straightforward**
- **Now for the fun stuff ...**



2/9/2005

CS-034: Lecture 2 (twd/pvh)

34

Fun with Functions (1)

```
void ArrayDouble(int A[], int len) {
    int i;
    for (i=0; i<len; i++)
        A[i] = 2*A[i];
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

35

Fun with Functions (2)

```
void ArrayBop(int A[],
              unsigned int len,
              int (*func)(int)) {
    int i;
    for (i=0; i<len; i++)
        A[i] = (*func)(A[i]);
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

36

Fun with Functions (3)

```
int triple(int arg) {
    return 3*arg;
}

int main() {
    int A[20];
    ... /* initialize A */
    ArrayBop(A, 20, triple);
    return 0;
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

37

Swap, Revisited

```
void swap (int *i, int *j) {
    int *tmp;
    tmp = j; j = i; i = tmp;
}
/* can we make this generic? */
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

38

Casts, Revisited

- **Two purposes**

- **coercion**

```
int i, j;
float a;
a = (float i)/(float j);
```

- **intimidation**

```
float x, y;
swap((int *)&x, (int *)&y);
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

39

Nothing, and More ...

- ***void* means, literally, nothing:**

```
void NotMuch(void) {
    printf("I do nothing\n");
}
```

- **What does *void ** mean?**

- **it's a pointer to anything you feel like**
 - a generic pointer

2/9/2005

CS-034: Lecture 2 (twd/pvh)

40

Rules

- **Use with other pointers**

```
int *x;
void *y;
x = y; /* legal */
y = x; /* legal */
```

- **Dereferencing**

```
void *z;
*z /* illegal!*/
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

41

An Application: Generic Swap

```
void gswap (void *p1, void *p2,
            unsigned int size) {
    int i;
    for (i=0; i <= size; i++) {
        char tmp;
        tmp = ((char *)p1)[i];
        ((char *)p1)[i] = ((char *)p2)[i];
        ((char *)p2)[i] = tmp;
    }
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

42

Using Generic Swap

```

short a, b;
gswap(&a, &b, sizeof(short));

int x, y;
gswap(&x, &y, 4);

int A[] = {1, 2, 3}, B[] = {7, 8, 9};
gswap(A, B, sizeof(A));

```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

43

sizeof

```

int main( ) {
  int A[] = {1, 2, 3};
  char s1[] = "hello";
  char *s2 = "goodbye";

  printf("%d, %d, %d\n",
        sizeof(A),
        sizeof(s1),
        sizeof(s2));
}

```

```

% gcc sizeof.c -o sizeof
% ./sizeof
12, 6, 4

```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

44

More ...

```
int main( ) {
    void func(int arg[]); /* declared before
                           defined */

    int A[10];
    func(A);
    return 0;
}

void func(int arg[]) {
    printf("%d\n", sizeof(arg));
}
```

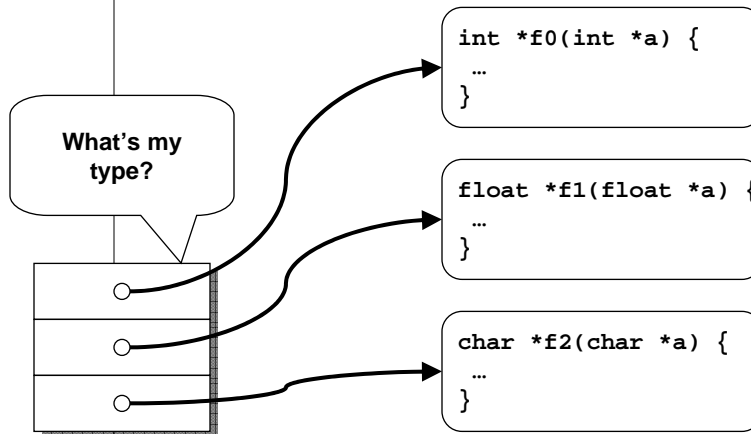
```
% gcc sizeof2.c -o sizeof2
% ./sizeof2
4
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

45

For Our Next Trick ...



2/9/2005

CS-034: Lecture 2 (twd/pvh)

46

Working Our Way There ...

- **An array of 3 ints**
 - `int A[3];`
- **An array of 3 int *s**
 - `int *A[3];`
- **A func returning an int *, taking an int ***
 - `int *f(int *);`
- **A pointer to such a func**
 - `int *(*f)(int *);`

2/9/2005

CS-034: Lecture 2 (twd/pvh)

47

There ...

- **An array of func pointers**
 - `int *(*f[3])(int *);`
- **An array of generic func pointers**
 - `void *(*f[3])(void *);`

2/9/2005

CS-034: Lecture 2 (twd/pvh)

48

Using It

```

int *f0(int *a) { ... }
float *f1(float *a) { ... }
char *f2(char *a) { ... }
int main() {
    int x = 1;
    int *p;
    void *(*f[3])(void *);
    f[0] = (void *(*)(void *))f0;
    f[1] = (void *(*)(void *))f1;
    f[2] = (void *(*)(void *))f2;
    p = f[0](&x);
}

```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

49

Casts, Yet Again

- They tell the C compiler:
“shutup, I know what I’m doing”

- Sometimes true

```
f[0] = (void *(*)(void *))f0;
```

- Sometimes false

```

int f = 7;
(void (*)(int))f(2);

```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

50

Laziness ...

- **Why type the declaration**

```
void *(*f)(void*, void *);
```

- **You could, instead, type**

```
MyType f;
```

- **(If, of course, you can somehow define *MyType* to mean the right thing)**

2/9/2005

CS-034: Lecture 2 (twd/pvh)

51

typedef

- **Allows one to create new names for existing types**

```
typedef int *IntP;
```

```
int *x;
```

– means the same as

```
IntP x;
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

52

More typedefs

```
typedef struct {  
    float real;  
    float imag;  
} complex_t;  
  
complex_t I, *IP;
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

53

And ...

```
typedef void *(*MyFunc)(void *, void *);  
  
MyFunc f;  
  
/* you must do its definition the long  
   way */  
  
void *f(void *a1, void *a2) {  
    ...  
}
```

2/9/2005

CS-034: Lecture 2 (twd/pvh)

54

And Finally ...		
● What's a possible use of ...		
void **		
?		
2/9/2005	CS-034: Lecture 2 (twd/pvh)	55