This is the new version. I think it will go well.

Recall memory: a big array of bytes "memory cells". Each with an address.
All data and code are stored in memory, so all variables and all
functions have addresses. C allows us to work with the actual
numerical addresses of these memory cells.

Why is this useful?

a. you can put/get data from specific numeric addresses
   - only really useful for system level/hardware control

b. pass by reference: tell a functon WHERE data are stored : USEFUL
c. linked data structures : WHERE is the next link or subtree :USEFUL
d. dynamic memory allocation: for next time.
e. efficient traversal of arrays : not any more -- compilers are smart

Outline for today: What can we store? How can use use their addresses?

Ans: simple variables (storage for one int, a char, a float)
     arrays (a sequence of contiguous storage cells of one type)
     struct (a collection of varied types more or less adjacent)
     functions (a sequence of machine language instructions)

For each of these, we shall see how to obtain their address and
we shall see what we can do with those addresses.

I. Simple variables : ex1.c

```
int main()
{
     int i,j;
     char c;
     float a;

     i = 2;
     j = i;
     if ( i == j )
       c = 't';

     printf("i=%d, j=%d, c=%c, a=%f\n", i,j,c,a)
}
```

Nothing new here.

Q1: But WHERE in memory are these variables?
A: we can ask C where these are stored by using the & operator.

```
#include <stdio.h>        //  ex1pa.c: print addresses

typedef unsigned long ul;

int main()
{
     int i,j;
     char c;
     float a;

     i = 2;
     j = i;
     if ( i == j )
       c = 't';

     printf("i=%d, j=%d, c=%c, a=%f\n", i,j,c,a);
     printf("locations are:\n");
     printf("i=%p, j=%p, c=%p, a=%p\n", &i,&j,&c,&a);
     printf("i at %lu, j at %lu, c at %lu, a at %lu\n",
             (ul)&i,(ul)&j,(ul)&c, (ul)&a);
}
```

Q2: Can we store these addresses?
A2: y: we need a variable type that can hold an address.
the type is a pointer variable, and we create them as follows
and use the values to print

```
/* ex1sa.c: store addresses */

#include <stdio.h>

typedef unsigned long ul;

int main()
{
     int i,j;
     char c;

     int *p;    /* p holds address of an int */
     int *q;    /* q holds address of an int */
     char *cp;

     p = &i;    /* get address and store it  */
     q = &j;    /* get address and store it  */
     cp = &c;

     i = 3;
     j = i;
     if ( i == j )
       c = 't';

     /* now to print them out */

     printf("i=%d, j=%d, c=%c\n", i,j,c);
     printf("locations are:\n");
     printf("i at %p, j at %p, c at %p\n", p, q, cp);
}
```

Discussion: We can draw these variables in our memory diagram as follows. Each pointer is a variable, so it occupies a space in memory. The value IN the variable is the address of a different variable.. We draw that correspondence with an arrow.

Q3: What else can we do with pointer variables?
A3: We can use the address to get back to the original variable. The term for this is 'dereference'. The operator is "*" applied to a pointer variable..

```
/* ex1dp.c: dereference pointers */

#include <stdio.h>

typedef unsigned long ul;

int main()
{
     int i,j;
     char c;

     int *p;    /* p holds address of an int */
     int *q;    /* q holds address of an int */
     char *cp;

     p = &i;    /* get address and store it  */
     q = &j;    /* get address and store it  */
     cp = &c;

     *p = 3; /* same as: i = 3; */
     *q = *p;/* same as: j = i; */
     if ( *p == *q )       /* same as if ( i == j ) */
       *cp = 't' ;  /* same as       c = 't'; */

     /* now to print them out */

     printf("i=%d, j=%d, c=%c\n", i,j,c);
     printf("locations are:\n");
     printf("i at %p, j at %p, c at %p\n", p, q, cp);
}
```

Q4: What OTHER operations can we do on pointers to simple variables?
A4: compare pointers using ==, !=, <, > ...

```
/* ex1cp.c: compare pointers */

#include <stdio.h>

typedef unsigned long ul;

int main()
{
     int i,j;
     char c;

     int *p;    /* p holds address of an int */
     int *q;    /* q holds address of an int */
     char *cp;

     p = &i;    /* get address and store it  */
     q = &j;    /* get address and store it  */
     cp = &c;

     *p = 3;  /* same as: i = 3; */
     *q = *p;/* same as: j = i; */
     if ( *p == *q )       /* same as if ( i == j ) */
       *cp = 't' ;  /* same as       c = 't'; */
     if ( p == q )
       printf("p equals q\n");
     else
       printf("p does not equal q\n");

     /* now to print them out */

     printf("i=%d, j=%d, c=%c\n", i,j,c);
     printf("locations are:\n");
     printf("i at %p, j at %p, c at %p\n", p, q, cp);
     return 0;
}
```

A4b: We can pass pointers to functions:

```
/* ex1pf.c: pass to functions */

#include <stdio.h>

void compare(int *, int *);
void display(int *, char *);

int main()
{
     int i,j;
     char c;

     int *p;    /* p holds address of an int */
     int *q;    /* q holds address of an int */
     char *cp;

     p = &i;    /* get address and store it  */
     q = &j;    /* get address and store it  */
     cp = &c;

     *p = 3;  /* same as: i = 3; */
     *q = *p;/* same as: j = i; */
     if ( *p == *q )
        c = 't';

     compare(p, q);
     display(p, cp);
}
/*
 * compare values AND addrs of two int ptrs
 */
void compare(int *p1, int *p2)
{
     if ( *p1 == *p2 )     /* same as if ( i == j ) */
       printf("values of pointees are equal\n");
     if ( p1 == p2 )
       printf("both point to same place\n");
     else
       printf("point to different places\n");
}
void display(int *ip, char *cp)
{
    printf("values are: %d %c\n", *ip, *cp);
    printf("addrs  are %p %p\n", ip,   cp);
}
```

Q5: a pointer is also a simple, single variable, can we take ITS address?
A5: What do you think? What would be the notation to create one? What does dereferencing do?

Part II: Pointers and Arrays:

An array is a contiguous sequ of memory cells, all containing values of the same type. Each cell has an address, the array has a starting address. We say that the address of the array is the address of the first element.

```
/* ex2.c: arrays */

#include <stdio.h>

int main()
{
        char   m[20] = "hello";
        char   n[]   = "how are you?";

        char   *p, *q;
        p = &m[0];    /* OR p = m */
        q = n;

        if ( *p == *q )
                printf("values are same\n");
        if ( p == q )
                printf("addresses are same\n");
        else
                printf("addresses differ\n");

        printf("the char at m is %c\n", *m);
        printf("the string at m is %s\n", m);
        printf("the address of m is %p\n", m);
}
```

Q1: How can we use the pointer to get other elements in the array?
A1: use normal indexing notation.

```
/* ex2ia.c index into arrays */

#include <stdio.h>

int main()
{
        char   m[20] = "hello";
        char   n[]   = "how are you?";

        char   *p, *q;
        p = &m[0];    /* OR p = m */
        q = n;

        printf("chars in m[2] and m[4] are %c %c\n", p[2], p[4]);
        *n = p[2];
        printf("string at n is %s\n", q);
}
```

Note: we used n with a *. But I thought that * is for pointers!
Note: * is for addresses. The name of an array is an address, so that is ok.

FACT: address[index] MEANS the item 'index' spots from address
FACT: *address MEANS the thing at address.
Address can be an array name or a pointer var value

Q2: What other operations can we do with pointers to arrays?
A2: assign, comparison, increment, decrement, +, -

Look at this code: What do you think it prints out?

```
/* ex2ao.c arithmetic operations */

#include <stdio.h>

int main()
{
        char   m[20] = "hello";
        char   n[]   = "how are you?";

        char   *p, *q;
        p = &m[0];    /* OR p = m */
        q = n;

        printf("chars in m[2] and m[4] are %c %c\n", p[2], p[4]);
        printf("string at n is %s\n", q);

        p++;
        q = q + 3;
        printf("chars in m[2] and m[4] are %c %c\n", p[2], p[4]);
        printf("string at n is %s\n", q);
}
```

Here is another example to ponder:

```
/* ex2ae.c arithmetic exercises */

#include <stdio.h>

int main()
{
        int    t[5];
        int    *p;
        int    i;

        p = t;
        for(i=0; i<5; i++){
                *p = i * 2;
                p++;
        }
        for(i=0; i<5; i++){
                printf("t[%i] = %d\n", i, t[i]);
        }
        p = t - 1;
        printf("t is at %p, p holds value %p\n", t, p);
        printf("p points to value\n", *p);
        printf("(p+2)[0] = %d\n", (p+2)[0]);
        printf("(p-2)[4] = %d\n", (p-2)[4]);
        printf("(p+10)[-8] = %d\n", (p+10)[-8]);
}
```

Using pointers to process arrays of chars: strcpy

Part III: Pointers and Structs

Structs occupy memory, therefore a struct has an address. We use the &
to get the address and the special notation -> to select members from
a pointer: (ex3.c)

```
/* ex3.c: pointers to structs */
#include <stdio.h>
#include <string.h>

struct time
{
       int    hr, mn;
};

struct tstop
{
       char   stn[20];
       char   dir;
       struct time when;
};

void print_event(struct tstop *);

int main()
{
       struct tstop s1, s2;
       struct tstop *p1, *p2;

       p1 = &s1;
       p2 = &s2;

       strcpy(s1.stn, "lynn");
       s1.dir = 'i';
       s1.when.hr = 9;
       s1.when.mn = 23;

       strcpy(p2->stn, "salem");
       p2->dir = 'i';
       p2->when.hr = 9;
       p2->when.mn = 12;

       print_event( &s1 );
       print_event( p2  );
       return 0;
}

/*
 * print out an event
 */
void print_event(struct tstop *p)
{
       printf("station: %s\n", p->stn);
       printf("    dir: %c\n", p->dir);
       printf("   when: %d:%d\n", p->when.hr, p->when.mn);
}
```

IV: Arrays of pointers, pointers to arrays of arrays

Draw a picture and trace this code: (ex4.c)

```
#include <stdio.h>

int main()
{
	char   food[4][20];
	int    i;

	strcpy(food[0], "peas");
	strcpy(food[1], "carrots");
	strcpy(food[2], "kale");
	strcpy(food[3], "lettuce");

	for(i=0; i<4; i++)
	{
		printf("item %d is %s\n", i, food[i]);
	}
	what_do_i_do(food);
}

void what_do_i_do(char a[4][20])
{
	char   *p[4];
	int    i;

	for(i=0 ; i<4 ; i++)
	{
		p[i] = &a[3-i];
	}
	for(i=0 ; i<4 ; i++ )
	{
		printf("in array p, item %d is %s\n", i, p[i]+i);
	}
}
```