MAN, I SUCK AT THIS GAME.
CAN YOU GIVE ME
A FEW POINTERS?

I HATE YOU.

0xA28213A
0x6339392C,
0x7363682E.
Pointers

• Better than sliced bread!
• Similar to Java references
• Java code example:

```java
myX = new X();
```

![Diagram showing a pointer to an object]
Assume the X object instance fits in the box
Java code:

```java
myX.doSomething(13);
```

C equivalent (mostly):

```c
doSomething(13, myX);
```
A pointer is a variable.

A pointer contains the address of another variable.
Two operators you need to know about:

- `&` is called an *addressing operator*. In an expression, it refers to the address of a variable.
- `*` is called the *dereferencing operator*. In an expression, it follows the pointer to the item being pointed to.
Concepts that must be distinguished:

• declaration

• use
pointer declaration:

```c
int x;    /* integer variable */
int *px;  /* pointer variable */
```

- `px` is a variable
- the type of `px` is *pointer to an integer*
- operations on pointer variables are limited
```c
int x;    /* integer variable */
int *px;  /* pointer variable */

x = 13;
px = &x;
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Label or Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>x</td>
</tr>
<tr>
<td>104</td>
<td>px</td>
</tr>
</tbody>
</table>
int x; /* integer variable */
int *px; /* pointer variable */

x = 13;
px = &x;
*px = 56;
/* 4 statements that accomplish the same thing */

1. \( x = x - 12; \)

2. \( \ast px = x - 12; \)

3. \( \ast px = \ast px - 12; \)

4. \( x = \ast px - 12; \)
Allowed Operations on Pointers

```c
int *px, *py;
int x, y;

px = &x;    /* get address */
py = &y;
px = py;    /* assignment; both */
            /* lhs and rhs are same type */
px = NULL;  /* assignment to NULL */
if ( px == NULL)    /* comparison */
    printf("pointer is NULL\n");
```
NOT Allowed on Pointers

```c
int *px, *py, *pz;

px = 12;    /* assignment to */
            /* non-NULL literal */

pz = px + py; /* add or subtract */
            /* two pointer values */

px = py * 2 /* multiply or divide */
     /* of pointer value(s)*/
```
Draw a Diagram

```c
int x, y, z;
int *px, *py, *pz;
px = &x;
py = &y;
pz = &z;

*px = 24;
py = px;
z = 25;
*py = x - z;
```
Arrays

• Designed for speed of access
• Consecutive, same-sized elements

array
(one element per box)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>base</td>
</tr>
<tr>
<td></td>
<td></td>
<td>base + (1 * size)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>base + (2 * size)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>base + (3 * size)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Declare:

```c
int a[5];
```

Use:

```c
a[0] = 1;
a[1] = 2;
```
As allocated within memory:
(one integer fits into a box)
No bounds checking!

```c
int ar[12];
ar[12] = -1;  /* No compiler error! */
ar[300] = ar[300] + 3;  /* OK! */
```

This leads to the question:
Why no bounds checking?
#include <stdio.h>

#define BUFFERSIZE 10

int main(int argc, char *argv[]) {
    int buffer[BUFFERSIZE];
    int i; /*loop induction variable */

    /* place TOO MANY into buffer ! */
    for (i = 0; i < 11; i++) {
        buffer[i] = -825;
    }

    for (i = 0; i < 11; i++) {
        printf("%d ", buffer[i]);
    }

    buffer[36] = 5;

    return 0;
}

prints 11 times: -825

not a compiler error!

Does not crash!
C uses pointer notation with arrays:

\[ \text{a}[i] \text{ is equivalent to } *(\text{a} + i) \]

and

\[ \&\text{a}[i] \text{ is equivalent to } \text{a} + i \]
int a[5];
int *ap;

ap = a;

ap++; /* ap has the address of a[1] */

*ap = 2;

ap++; /* OK. */
a++; /* BAD! NOT ALLOWED! */
/* a is not a pointer! */
#include <stdio.h>

#define MAXARRAYSIZE 5
main()
{
    int intarray[MAXARRAYSIZE];
    int *iap;    /* a pointer to an int */
    int k;    /* loop induction variable */

    /* one implementation */
    iap = intarray;
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        *iap = k + 1;
        iap++;
    }
    iap = intarray;
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        printf("%d\n", *iap);
        iap++;
    }

    /* another implementation */
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        intarray[k] = k + 1;
    }
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        printf("%d\n", intarray[k]);
    }
}
#include <stdio.h>

#define MAXARRAYSIZE 5
main()
{
    int intarray[MAXARRAYSIZE];
    int *iap; /* a pointer to an int */
    int k; /* loop induction variable */

    /* one implementation */
    iap = intarray;
    for (k = 0; k < MAXARRAYSIZE; k++) {
        *iap = k + 1;
        iap++;
    }
    iap = intarray;
    for (k = 0; k < MAXARRAYSIZE; k++) {
        printf("%d\n", *iap);
        iap++;
    }
    /* ... */
}
#include <stdio.h>

#define MAXARRAYSIZE 5

main()
{
    int intarray[MAXARRAYSIZE];
    int *iap; /* a pointer to an int */
    int k; /* loop induction variable */
    /* ... */
    /* another implementation */
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        intarray[k] = k + 1;
    }
    for ( k = 0; k < MAXARRAYSIZE; k++)  {
        printf("%d\n", intarray[k]);
    }
}
From the lecture notes:

```c
int a = 3;
int b = 8;
int c = 0;
int *cp;

c = a + b;

cp = &c;

cp++; /* allowed, but probably not reasonable */
```

Contrast this code with:

```c
iap = intarray;
for (k = 0; k < MAX; k++) {
    ..
    ..
    iap++; /* correct and reasonable */
    iap++; /* correct and reasonable */
}
```
```c
#include <stdio.h>
#define BUFFERSIZE 10
int main(int argc, char *argv[]) {
    int y = 9;
    int buffer[BUFFERSIZE];
    int i;
    int x = 7;

    /* place TOO MANY into buffer ! */
    for (i = 0; i < 200; i++) {
        buffer[i] = -825;
    }
    printf("x = %d y = %d\n", x, y);
    return 0;
}
```
% cc ovsimple2.c -o ovsimple2
% ovsimple2
x = 7 y = -825
Segmentation fault
%
/* program to print lines backwards */
#include <stdio.h>

#define BUFFERSIZE 80
int main()
{
    char buffer[BUFFERSIZE]
    char *bp;    /* a pointer to a character */
    int k, j;    /* loop induction variables */

    bp = buffer;
    while ( fgets(buffer, BUFFERSIZE, stdin) != NULL ) {
        /* buffer has one line of input */
        printf("the line backwards:\n");

        /* find the end of the line */
        k = 0;
        while ( *(buffer+k) != '\0' ) k++;
        k--;
        if ( (k >= 0) && (*(buffer+k) == '\n') ) k--;
        /* print relevant characters in reverse order */
        for ( j = k; j >= 0; j-- ) {
            printf("%c", *(buffer + j));
        }
        printf("\n");
    }
    return (0);
}
buffer

0  1  2  3  4  5
x  y  z  \n  \0
#include <stdio.h>

void increment(int a);

main(){
    int x;
    x = 1;
    printf("before call, x = %d\n", x);
    increment(x);
    printf("after call, x = %d\n", x);
}

void increment(int a){
    a++;
}
#include <stdio.h>
void increment(int  a);

main()
{
    int x;
    x = 1;
    printf("before call, x = %d\n", x);
    increment(x);
    printf("after call, x = %d\n", x);
}

void increment(int  a){
    a++;
}
swap(&int1, &int2);

/* function swapmah */
/* interchanges two integer values */
/* parameters: */
/* px pointer to an integer */
/* py pointer to the other integer */
void swap(int *px, int *py) {
  int temp;

  temp = *px;
  *px = *py;
  *py = temp;
}

a call; arguments are the addresses of 2 integers; this code is the caller or parent

when called or invoked, this code is the callee or child
C string

An array of characters, which uses the null character to delimit the end of the string.

'\0' is the null character

"Hi."  

```
'H' 'i' ' ' '\0'
```

"12"  

```
'1' '2' '\0'
```
char *msg;  // pointer to a character used for strings

printf("%s", msg);
Other I/O functions
(in the stdio library)

int putc(int c, FILE *fp);

returns:
character written or
EOF if an error
occurred

character to print

which file
Other I/O functions
(in the stdio library)

```c
int getc(FILE *fp);
```

returns:
character or EOF

which file
Other I/O functions (in the stdio library)

FILE *fopen(char *filename, char *mode);

- returns: file pointer
- NULL if open failed
- string specifying file name
- "r" read
- "w" write
- "a" append
Other I/O functions
(in the stdio library)

```c
int fclose(FILE *fp);
```

`NULL` if OK,
`EOF` if error occurred
Other I/O functions (in the stdio library)

```c
int fputs(char *line, FILE *fp);
```

returns:
Ø if OK
EOF if an error
Other I/O functions
(in the stdio library)

int *fgets(char *line, int maxline, FILE *fp);

returns: line or null on EOF or error

where to place input

at most, maxline-1 characters will be read

which file
Other I/O functions
(in the `stdio` library)

```c
int scanf(char *format, [, *arg1] . . .);
```

- **string:** how input is to appear
- **returns:** number of successfully matched items
- **where to place items**
- **looks at stdin**
scanf("%d %d", &x, &y);

input  12  -85

x gets 12
y gets -85
Other I/O functions
(in the `stdio` library)

```c
int sscanf(char *string, char *format, [, *arg1] . . .);
```

looks at this string, instead of `stdin`

same as `scanf`
Why is this code *really bad*, and asking for trouble?

```c
char buf[60];
scanf("%s\n", buf);
```
Return values

```c
int x( . . . ) {
    return 16;
}
```

expression
int main( ) {
    return 0;
}

#include <stdio.h>
int main( ) {
    exit(0);
}

"all is well"

#include <stdio.h>
int main( ) {
    exit(0);
}

"all is well," and close all files.