

Pointers

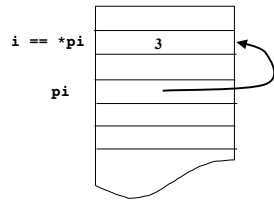
- C relies heavily on pointers:

```
int i;  
int *pi;
```

```
pi = &i;  
*pi = 3;
```

& = unary operator
Determines address of object

* = indirection operator
Determines data at an address

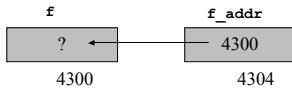


Pointer = variable containing address of another variable

```
float f; /* data variable */  
float *f_addr; /* pointer variable */
```

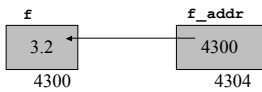
`f` `f_addr` any float
? ? → ?
4300 4304 any address

```
f_addr = &f; /* & = address operator */
```

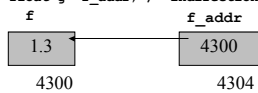


Pointer = variable containing _address of another variable

```
*f_addr = 3.2; /* indirection operator */
```



```
f = 1.3;  
float g = *f_addr; /* indirection: g is now 3.2 */
```



Pointer Example

```
include <stdio.h>

void main(void)
{
    int j;
    int *ptr;

    ptr=&j; /* initialize ptr before using it */
          /* *ptr=4 does NOT initialize ptr */

    *ptr=4; /* j <- 4 */
}

```

Pointers and Arrays

- An array name by itself is a pointer to the first element:

```
int a[100];
*a=5; /* same as a[0] = 5 */
*(a+3)=10; /* same as a[3] = 10; */
          /* Note: *a+3 means something else */

```

- Strings are arrays of char:

```
char s[3];
strcpy(s, "hi"); /* same as following line */
s[0] = 'h'; s[1] = 'i'; s[2] = '\0';

```

Input/Output in C

The C standard library offers a number of I/O functions for reading and writing to the console, files, streams, etc.

- **printf** print a formatted string to the output console (screen)
- **scanf** access data from the input console (keyboard)
- **puts** write a string to the console
- **gets** get a string from the console
- **fprintf** same as printf, but writes to a file
- **fscanf** same as scanf, but reads from a file
- **putchar** write a character to the console
- **getchar** get a character from the console
- **And many more...**

Formatted Output

- Example:

```
int i = 10;
float f = 2.5;
char s [] = "hi";
printf ("Jack's integer is %d\n", i);
printf ("Jill's float is %f\n", f);
printf ("My string s = %s\n", s);
```

- Syntax:

```
printf (string_with_formatting, var1, var2, ...);
```

Formats: %d integer, %f float, %c character, %s string, ...

Note "escape sequences": \n newline, \' quote, \x27, etc.

Format Strings - see printf documentation for full list

%d	Print as decimal integer
%6d	Print as decimal integer at least 6 characters wide
%f	Print as floating point
%6f	Print as floating point, at least 6 characters wide
%.2f	Print as floating point, 2 characters after decimal point
%6.2f	Print as floating point, 6 chars wide, and 2 after decimal point
%c	Print as a single ASCII character
%s	Print an arbitrary length string of ASCII characters
%x	Print as hexadecimal

Example using formatting of floats/doubles

```
#include<stdio.h>
void main(void)
{
    int i;
    float x=3.1415;
    double y=3.1415000000;
    printf("pi = %f\n",x);
    printf("pi = %e\n",x);
    printf("pi = %g\n",x);
    printf("pi = %3.3f\n",x);
    printf("pi = %3.3e\n",x);
    printf("pi = %3.12f\n",y);
    printf("pi = %3.12f\n",x);
}
```

```
csH> gcc -o test2 test2.c
csH> ./test2
pi = 3.141500
pi = 3.141500e+00
pi = 3.1415
-----
pi = 3.141
pi = 3.141e+00
pi = 3.141500000000
pi = 3.1414999996185
```

Formatted Input

•Example:

```
#include <stdio.h>
int i;
float f;
scanf ("%d %f\n", &i, &f); /* inputs an integer and a float */
```

•Syntax:

```
scanf (string_with_formatting, &var1, &var2,...);
```

•Note:

- Note **&** = "address of" operator - more on this later;
- The ampersand **&** is necessary because *scanf* modifies the values stored in the respective variables
- by comparison, *printf* only uses the values, without modifying them.

I/O Example

What does this print?

```
#include <stdio.h>

main ()
{
    int n;
    float x;
    char mark;
    scanf ("%d %f %c", &n, &x, &mark);
    printf ("Of %d %s,\n%f got %c's\n", n, "students", x, mark);
}
```

Type in the following input: 86 85.999 a

Input/Output via files

```
#include<stdio.h>
int main(void)
{
    FILE *handle;
    handle=fopen("elec3730.txt","w+");
    if (handle != NULL)
    {
        fprintf(handle,"Hello World via a file\n");
        fclose(handle);
    }
    else
    {
        printf("Could not open file\n");
    }
}
```
