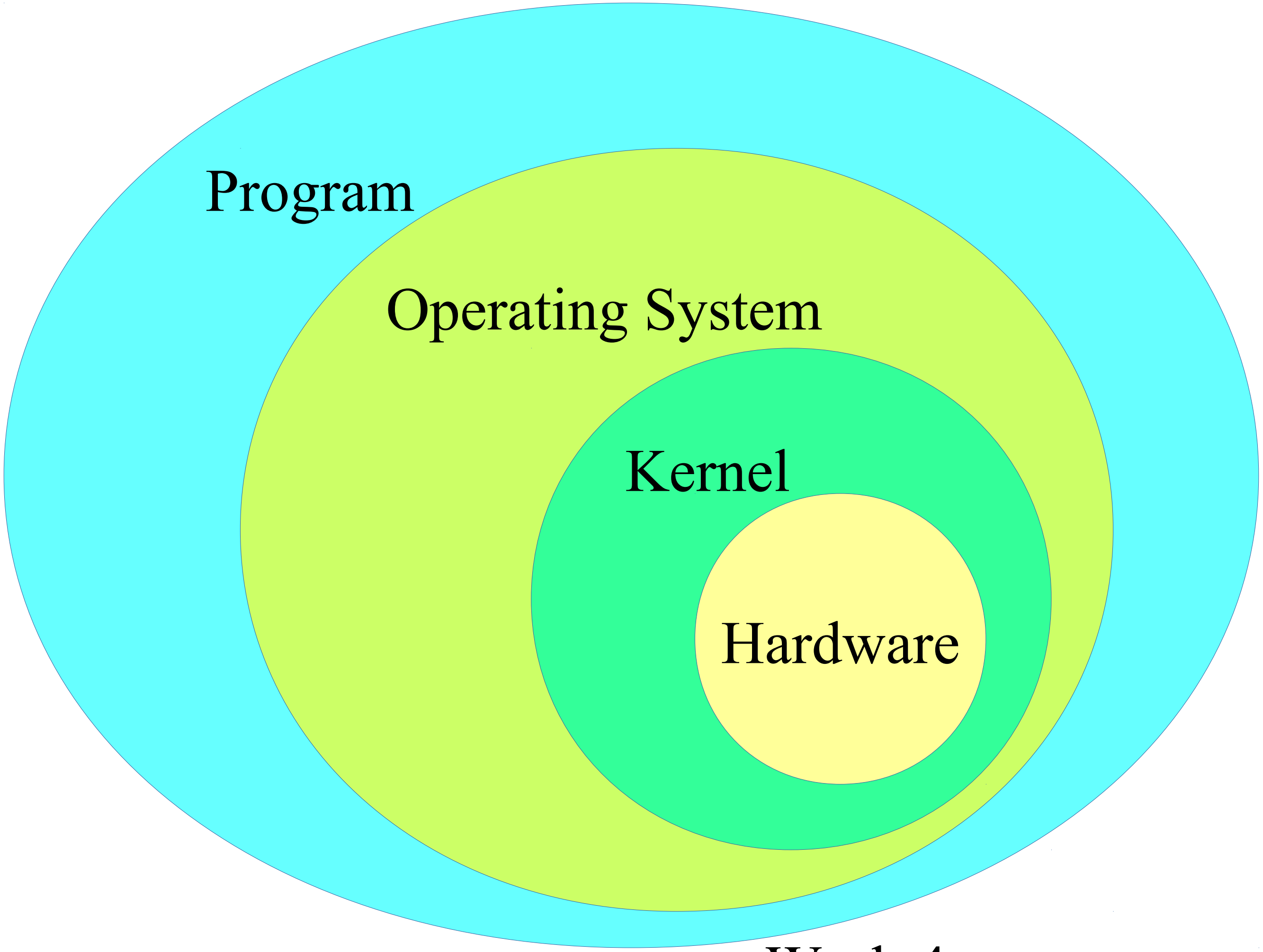
Week 4 Lecture 2

Input/Output Strings

Input and Output

- Input into a computer program and output from a computer program are difficult.
- Fortunately, the Operating System hides most of the complexity

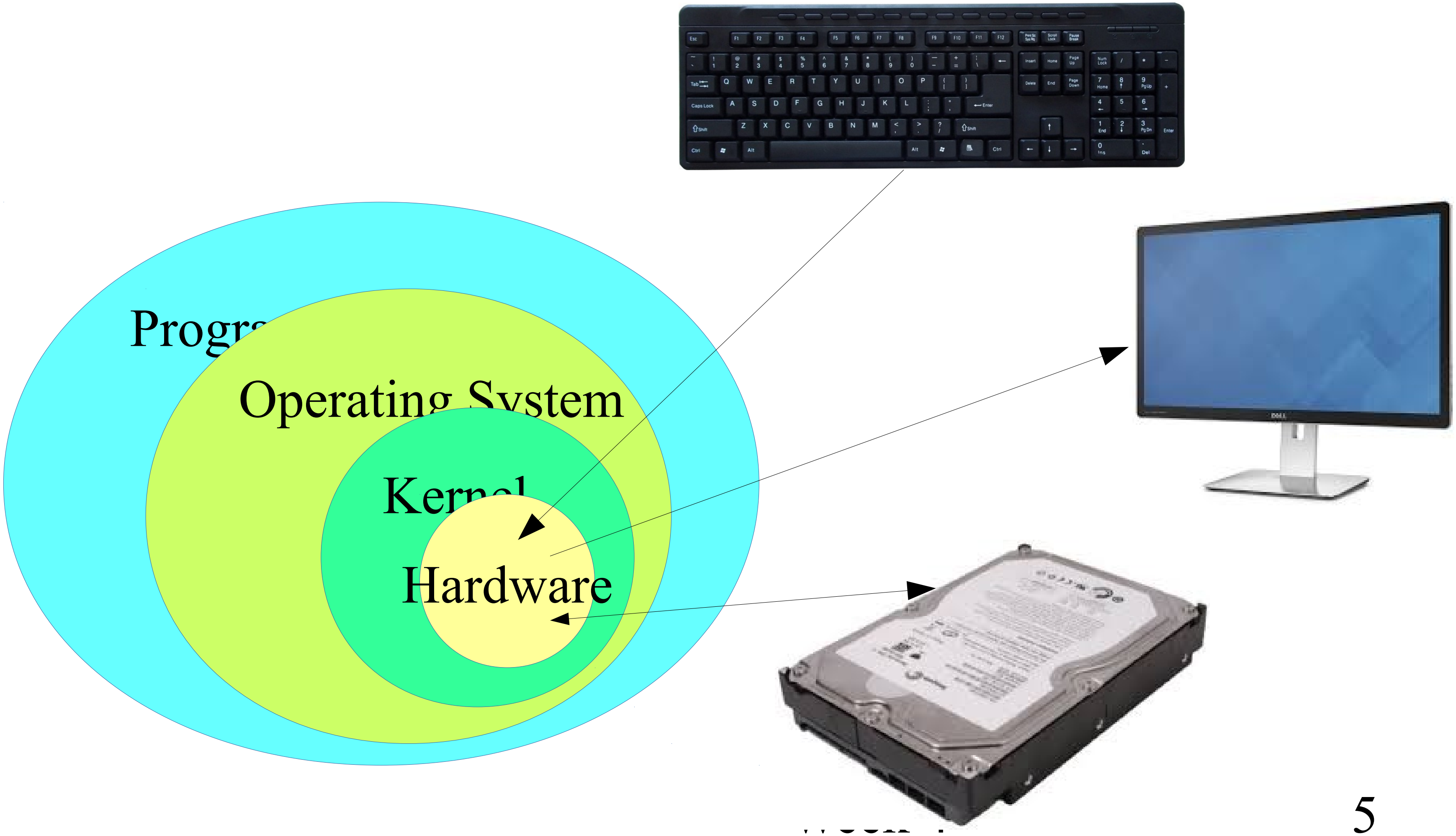
Operating System



Layers

- The hardware performs computations
- The Kernel is protected—special permissions are needed to access it.
  - Keeps programs from interfering with each other
- The rest of the operating system is a set of programs.
- User programs call on the OS programs to perform tasks.

Peripherals



Progr

Operating System

Kernel

Hardware

Output

- The function printf is an output function
  - It creates behavior outside the computer (i.e. other than add, subtract, jump, etc.)
  - It is a function that is supported by the operating system

Assembly Instructions

```
BITS 64

SECTION .data

Hello:          db "Hello world",10
len_Hello:      equ $-Hello

SECTION .text

global _start

_start:
                mov rax,1                       ; write syscall (x86_64)
                mov rdi,1                       ; fd = stdout
                mov rsi,Hello                   ; *buf = Hello
                mov rdx,len_Hello               ; count = len_Hello
                syscall

                mov rax,60                      ; exit syscall (x86_64)
                mov rdi,0                       ; status = 0 (exit normally)
                syscall
```

Assembly Lang Explanation

- mov rax,1
  - Put the print command in the ax register
- mov rdi,1
  - Put stdout in the di register
- mov rsi,Hello
  - Put the address of the “Hello world” in the si register
- mov rdx,len_Hello
  - Put the length of the “Hello world” string in the dx register
- syscall
  - Call the operating system

Ceci n'est pas un 'H'

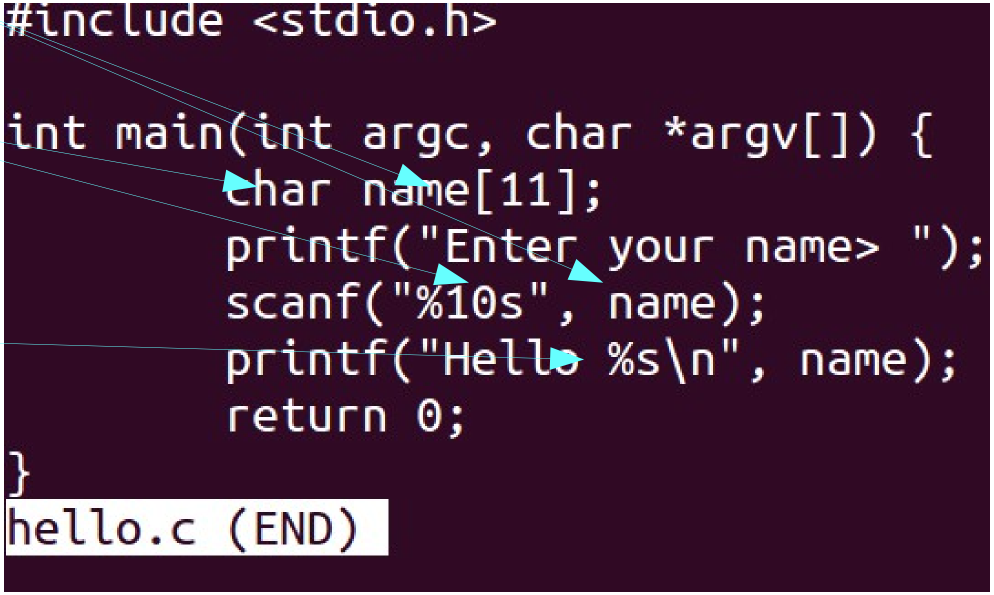
- 'H' is represented by 01001000 or 72
- The image of the 'H', a matrix of dots, is represented by a different binary string.
- This image must be placed in the terminal window.
- The image must be placed in a sequence with the images of the other letters.

Input

- Characters are read from the keyboard.
  - Images corresponding to the letters are also placed in the terminal.
- Binary representations of the characters read, must be placed in a memory location accessible to the program.
  - The computer must know how to interpret the characters placed in memory.

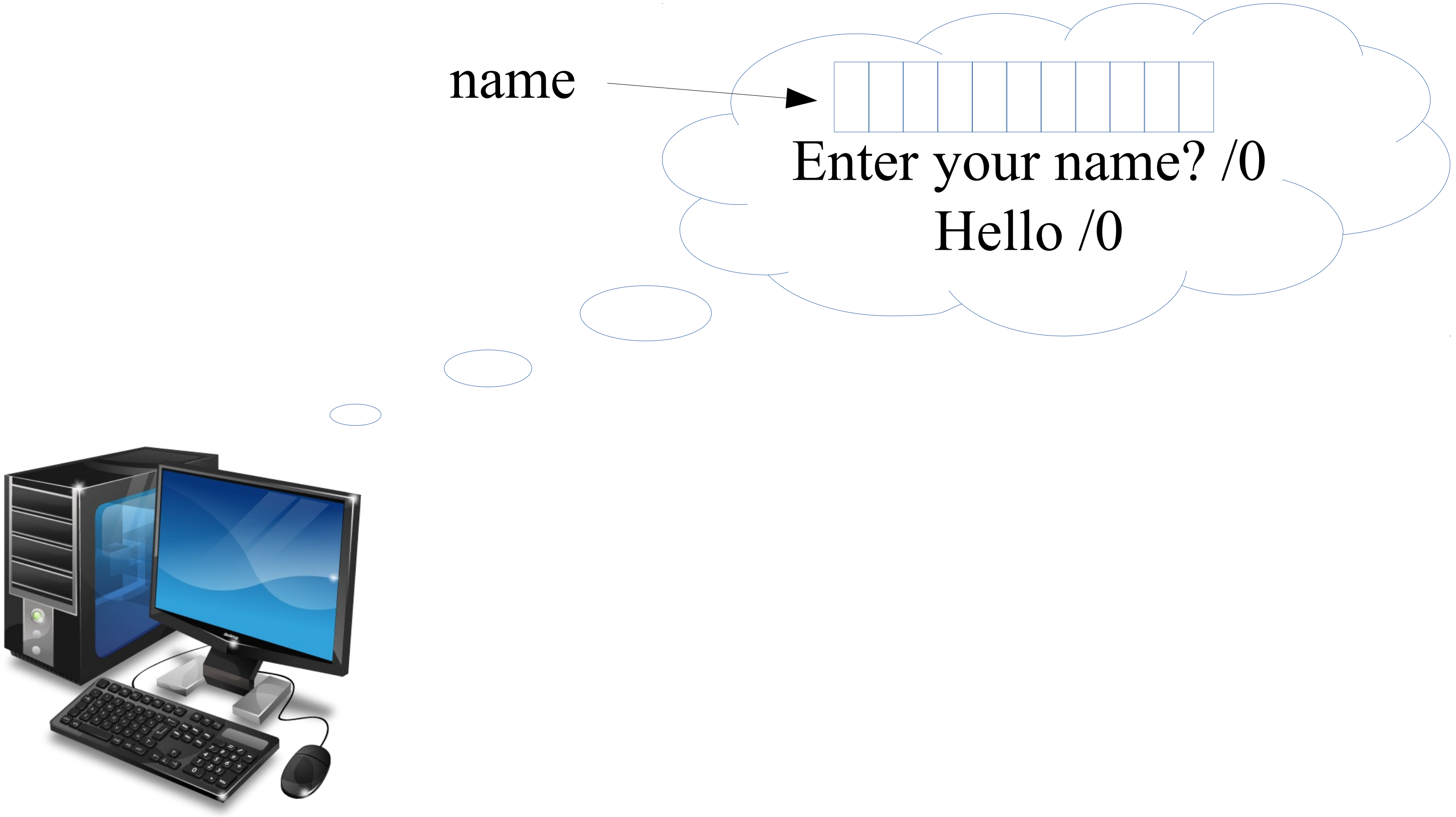
Hello to me

- Place to put input
- Type of input
- How to format input for output



```
#include <stdio.h>

int main(int argc, char *argv[]) {
        char name[11];
        printf("Enter your name> ");
        scanf("%10s", name);
        printf("Hello %s\n", name);
        return 0;
}
hello.c (END)
```

Variables



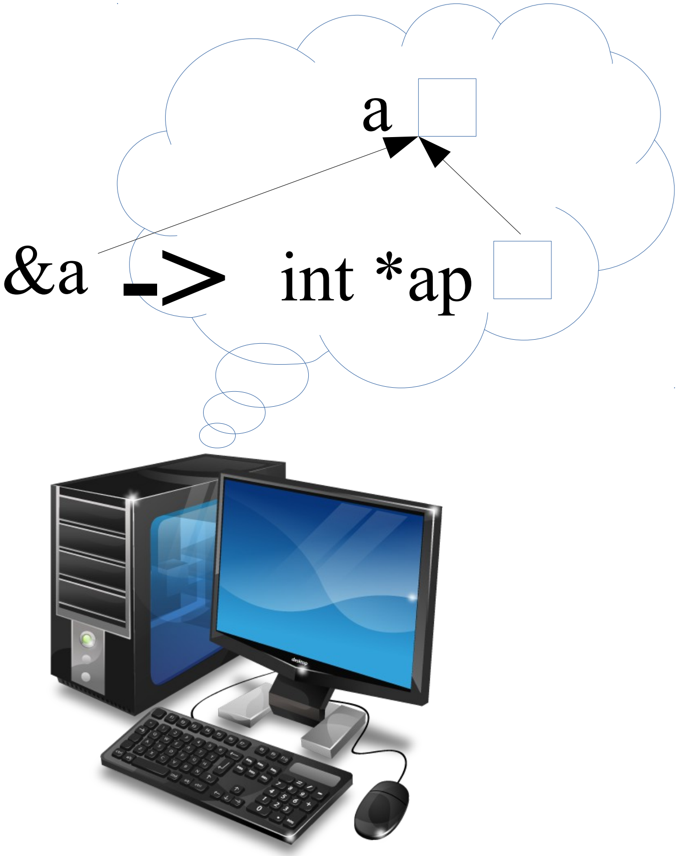
name

Enter your name? /0

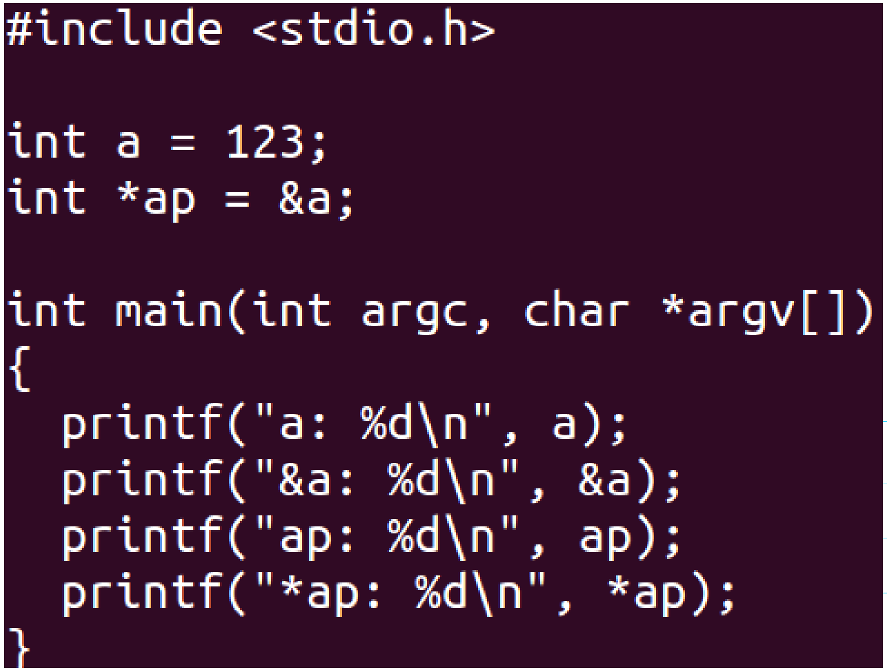
Hello /0

Variables, pointers, and pointer variables

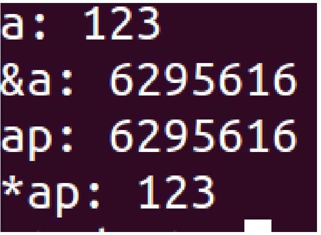
- Variable
  - int a
- Pointer to variable
  - &a
- Variable pointer
  - int *ap
- Assign pointer to pointer variable
  - ap = &a



Example

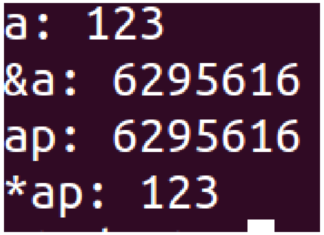


```
#include <stdio.h>

int a = 123;
int *ap = &a;

int main(int argc, char *argv[])
{
  printf("a: %d\n", a);
  printf("&a: %d\n", &a);
  printf("ap: %d\n", ap);
  printf("*ap: %d\n", *ap);
}
```



```
a: 123
&a: 6295616
ap: 6295616
*ap: 123
```

Pointers

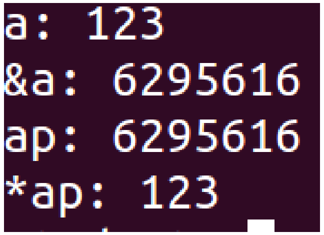


```
a: 123
&a: 6295616
ap: 6295616
*ap: 123
```

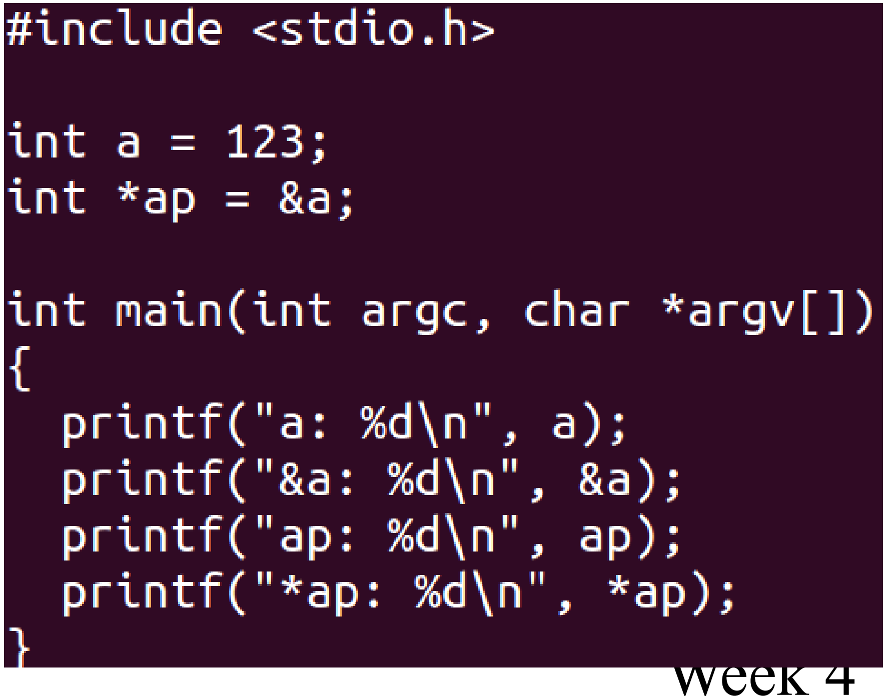
- The variable a is initialized to one hundred and twenty three.
- The address of the variable a is 6295616
- The variable ap is initialized to 6295616
- The contents at the address 6295616 is one hundred and twenty three.

Pointer Names

- *ap is an int
- ap is an int *
  - I.e., a pointer to and int

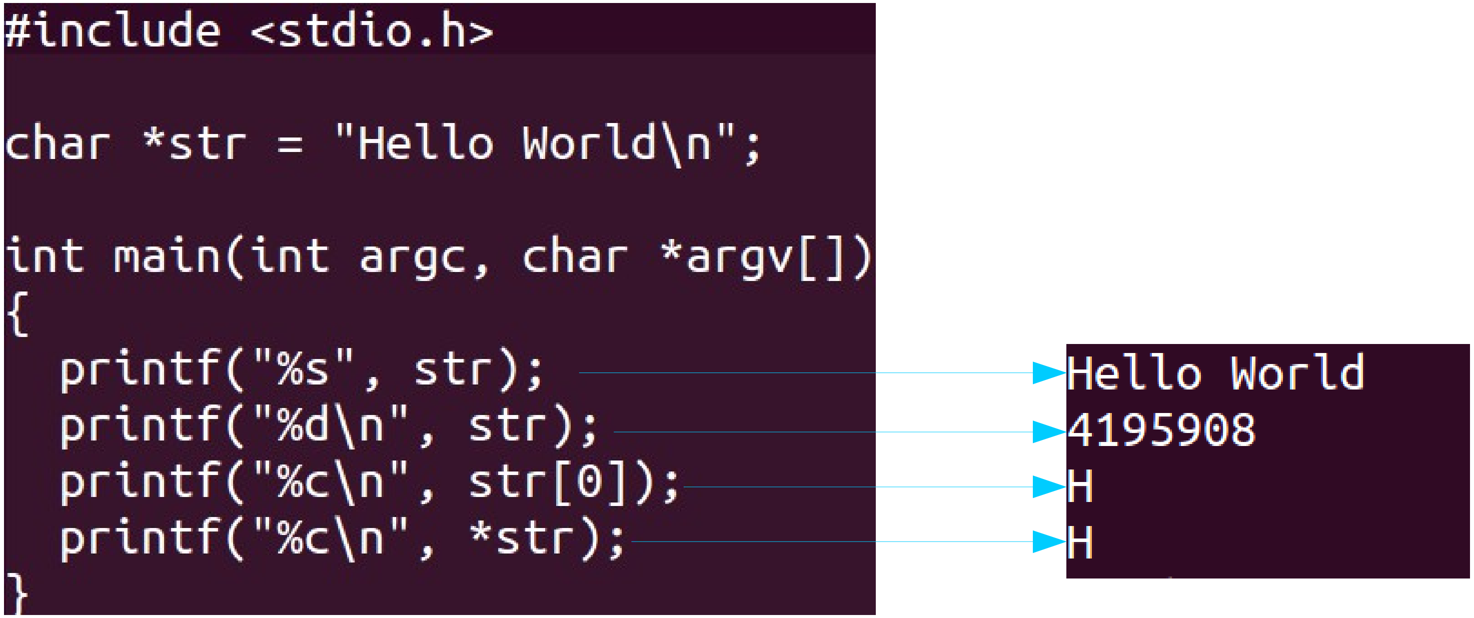


```
a: 123
&a: 6295616
ap: 6295616
*ap: 123
```



```c
#include <stdio.h>

int a = 123;
int *ap = &a;

int main(int argc, char *argv[])
{
  printf("a: %d\n", a);
  printf("&a: %d\n", &a);
  printf("ap: %d\n", ap);
  printf("*ap: %d\n", *ap);
}
```

Strings again

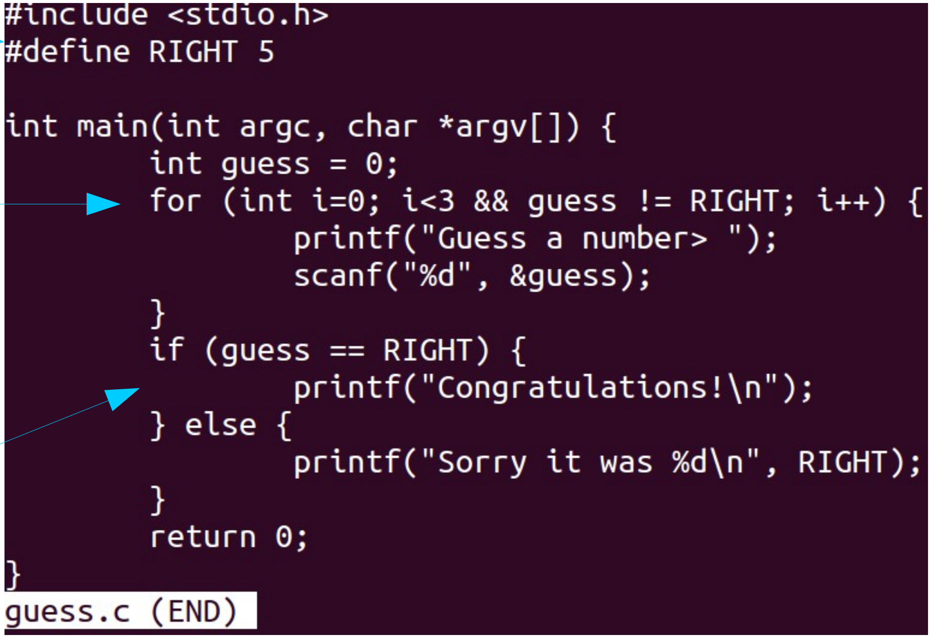


```c
#include <stdio.h>

char *str = "Hello World\n";

int main(int argc, char *argv[])
{
  printf("%s", str);
  printf("%d\n", str);
  printf("%c\n", str[0]);
  printf("%c\n", *str);
}
```

```
Hello World
4195908
H
H
```

Guess a Number

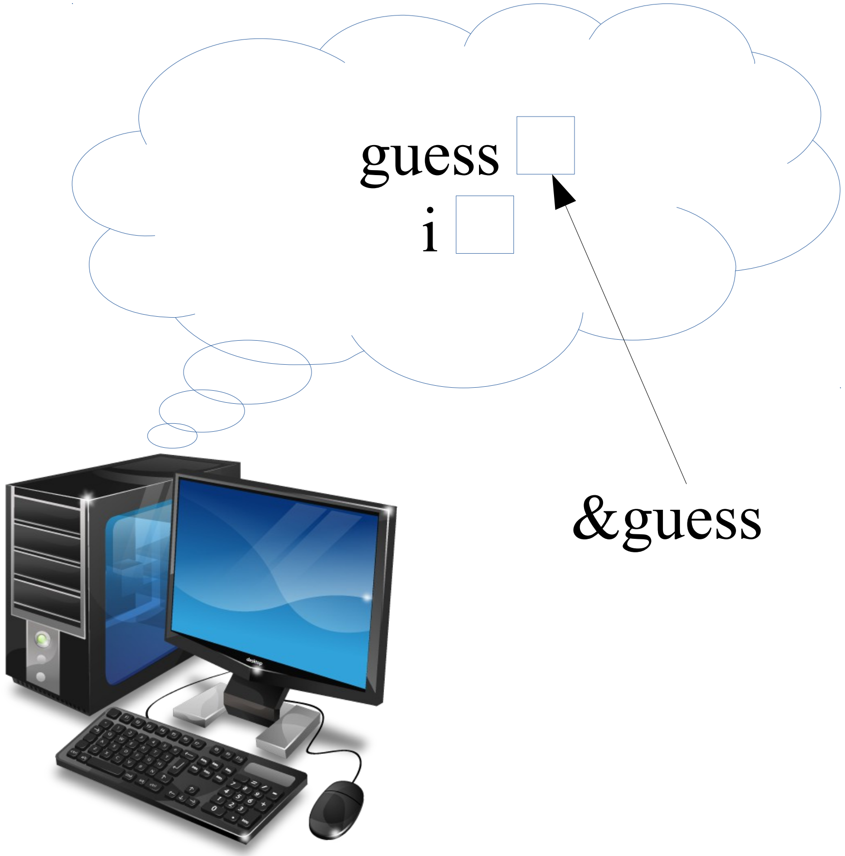
- Constant
- Accept guesses until correct or three tries
- Congratulations or Sorry



```
#include <stdio.h>
#define RIGHT 5

int main(int argc, char *argv[]) {
        int guess = 0;
        for (int i=0; i<3 && guess != RIGHT; i++) {
                printf("Guess a number> ");
                scanf("%d", &guess);
        }
        if (guess == RIGHT) {
                printf("Congratulations!\n");
        } else {
                printf("Sorry it was %d\n", RIGHT);
        }
        return 0;
}
guess.c (END)
```

Variables and Locations

- The variable guess is the name of a place where an int can be stored
- The variable &guess is the name of the location of a place where an int can be stored



L-value and R-value

- Consider the statement: `i = i + 1;`
- The `i` on the left refers to a location where a value will be stored
  - It is an l-value
- The `i` on the right refers to the value of the location
  - It is an r-value

Variables and Address

- The variable guess
  - There are numbers in guess
- The address of the variable guess
  - To put numbers in guess you need the address

- int guess;
- &guess

What about name?

- The address of the variable *name.
  - It is declared as an address
- The variable *name

- char name[11];
- *name

Strings are addresses

- Strings are the address of the first character in the string.
- They extend until there is a null character (i.e., '/0')
- They are constructed this way because each string has a different length.