

# CS241

Introduction To C

Lawrence Angrave

# Crash course in C

# Crash

```
char *p;  
p=0;  
*p=4
```

?

```
char * search(char*) {  
    char result[100];  
    ... do stuff  
    return result;  
}
```

?

```
static int debug=1;
```

```
main(...)
```

```
{
```

```
}
```

?

```
int counter() {  
    static int c=1;  
    printf("The value of c is %x",&c);  
    return c++;  
}
```

# Linkage & Scope

# Linkage

Identifiers (variables & functions) visible to another c file?

Main.c = Compilation unit

Myutils.c = Another compilation unit

Static : No!

# util.c

```
static int notthisone;
```

```
int runsuperfast=0;
```

```
void die(char*) {
```

```
    static int privatecounter;
```

```
    fprintf(stderr,mesg);
```

```
    exit(1);
```

```
}
```

# main.c

Refer to runsuperfast and die?

```
extern int runsuperfast;
```

```
void die(char *mesg);
```

# main.c

Refer to runsuperfast and die?

```
void die(char *); // declaration!  
extern int runsuperfast;
```

# Declarations to library functions?

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include "myutils.h"
```

# C Compiler

# gcc Gnu Compiler

Many options!

`man gcc`

Three for the price of one

Preprocessor

Compiler

Linker

# C Variants

C99 - A 'nicer' C

```
/* Multi-line comments */
```

```
// line comment (Not allowed in C89)
```

Mix variable declarations & code inside functions

# Pointers

# Pointers

How to shoot yourself in the foot and not realize it

`int *p`

Q. What is p?

How much storage is required for p?

Where is p in memory?

```
int *p
```

What is the value of

```
p == &*p;
```

```
p[0] == *p
```

```
char **argv
```

Explain this to your neighbor

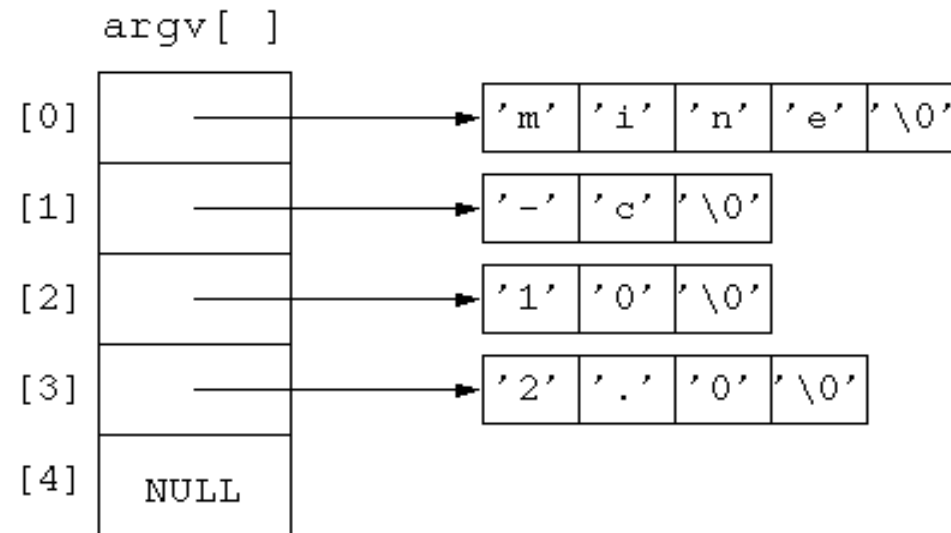
```
char ***argv
```

Explain this to your neighbor

When would you use it?

# Argument Arrays

An argument array is an array of pointers terminated by a NULL pointer.  
Each element of the array is of type `char *` and represents a string.



# Character strings

Basic support in Compiler for string constants and string pointers

Everything else is a library function

strcmp

strncmp

strlen

strcat

# strcmp

strcmp("Hello", "Hello") returns what?

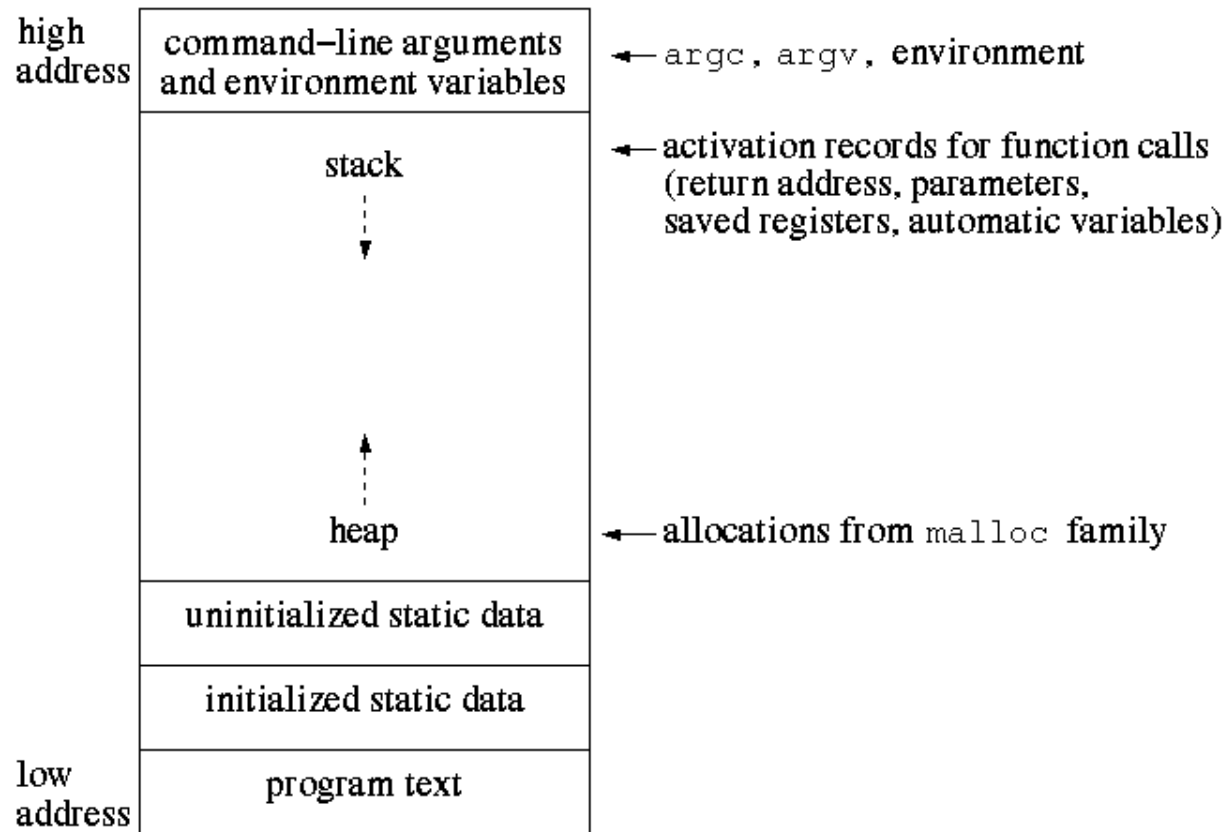
strcmp("1", "2") returns what?

# Why is this *bad*?

```
char *p="Hello";
```

```
strcpy(p," World");
```

# Ans: Understand storage



# Identifies & Linkage Summary

# Identifiers

“Identifier” denotes an object: a function, a tag, member of a structure, etc. We mainly consider identifiers associated with functions and variables

Scope is a region of program text where identifier is used

Identifiers declared more than once may refer to the same object because of linkage

Each identifier has a linkage class of external, internal or none.

An identifier representing an object has a linkage class related to storage class (storage duration).

# Linkage classes

Linkage class determines whether variables can be accessed in files other than the one in which they are declared.

Internal linkage class means they can only be accessed in the file in which they are declared.

External linkage class means they can be accessed in other files.

Variables declared outside any function and function name identifiers have external linkage by default.

They can be given internal linkage with the key word *static*.

Variables declared inside a function are only known inside that function and are said to have no linkage.

# Storage classes: static and automatic

Static storage class refers to variables that, once allocated, persist throughout the execution of a program.

Automatic storage class refers to variables which come into existence when the block in which they are declared is entered and are discarded when the defining block is exited.

Variables declared inside a function have automatic storage class unless they are declared to be static.

These are usually allocated on the program stack.

Variables defined outside any functions have static storage class.

The word *static* has two meanings in C.

One is related to storage class and the other to linkage class.