External and static variables

- External variable: declared outside the body of a function
- File scope: visible from the point of the declaration to the end of the file.
- Static storage duration: through the duration of the program.
- External/global variables have file scope and static storage duration.

static variables

```
static int i;

void f(void)
   static int j;
}
```

- static used outside a block means that the variable is only visible in the file in which it is declared
- static used in a block means that the variable lives beyond the duration of the block, and is initialized only once.

Example

```
#include <stdio.h>

int nextvalue()
{
    static int i = 0;
    i++;
    return i;
}

int main()
{
    int i;
    for(i = 10; i > 0; i--) {
        printf("%d\n", nextvalue());
    }
    return 0;
}
```

```
output:
1
2
3
4
5
6
7
8
9
10
```

extern

filea.c

```
extern int i;

void f(void) {
     i++;
}
```

fileb.c

```
int i = 0;
extern void f(void);
void g(void) {
     f();
     printf("%d\n", i);
}
```

- informs the compiler that `i` is an `int` variable, but doesn't cause it to allocate space.

typedef

- You can define new types using `typedef`.
- You have already seen the effects of typedef

```
typedef unsigned int size_t;
```

- Example

```
struct personrec {
   char name[20];
   int age;
};
typedef struct personrec Person;
Person *p = malloc(sizeof(Person));
```

typedef

- You can define new types using `typedef`.
- You have already seen the effects of typedef

```
typedef unsigned int size_t;
```

- Example

```
typedef struct {
   char name[20];
   int age;
} Person;

Person *p = malloc(sizeof(Person));
```

Header files

- When you begin to split up your C program into multiple files, you need header files to store function and type declarations.

main.c

```
void add(int);
int isEmpty();
extern List head;
int main()
{
   add(10);
   isEmpty();
   head = NULL;
}
```

list.c

```
List *head = NULL;
int isEmpty()
{...}
void add(int v)
{...}
void remove(int v)
{...}
```

list.h

```
struct node {
     int value;
     struct node * next;
} ;
typedef struct node List;
extern List *head;
int isEmpty(int);
void add(int);
void remove(int)
```

main.c

```
#include "list.h"

int main()
{
   add(10);
   isEmpty();
   head = NULL;
}
```

list.c

```
#include "list.h"
List *head = NULL;
int isEmpty()
{...}
void add(int v)
{...}
void remove(int v)
{...}
```

list.h

```
struct node {
     int value;
     struct node * next;
} ;
typedef struct node List;
List *head = NULL;
int isEmpty();
void add(int);
void remove(int)
```

Wrong!

main.c

```
#include "list.h"

int main()
{
   add(10);
   isEmpty();
}
```

list.c

```
#include "list.h"

int isEmpty()
{...}
void add(int v)
{...}
void remove(int v)
{...}
```

Wrong!

main.c

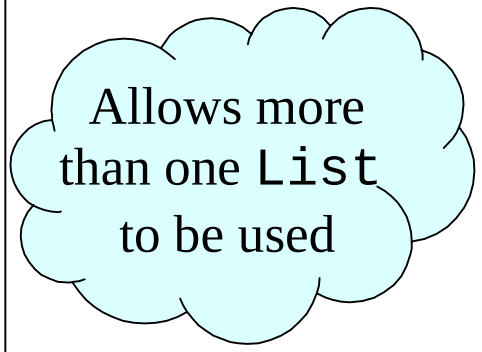
```
#include "list.c"

int main()
{
    add(10);
    isEmpty();
}
```

list.c

```
List *head = NULL;
int isEmpty()
{...}
void add(int v)
{...}
void remove(int v)
{...}
```

list.h

```
struct node {
     int value;
     struct node * next;
} ;
typedef struct node List;

int isEmpty(List *, int);
void add(List *, int);
void remove(List *, int)
```



main.c

```
#include "list.h"

int main()
{
   add(10);
   isEmpty();
}
```

list.c

```
#include "list.h"

int isEmpty(List *h)
{...}
void add(List *h, int v)
{...}
void remove(List *h, int v)
{...}
```

Protecting header files

- Compilation errors may result if a header file is included more than once.
- This causes a problem if the header file defines types.
- Use preprocessor directives to selectively compile.

```
#ifndef LIST_H
#define LIST_H
...(contents of the header file)
#endif
```

Makefiles

- Makefiles were originally designed to support separate compilation of C files.

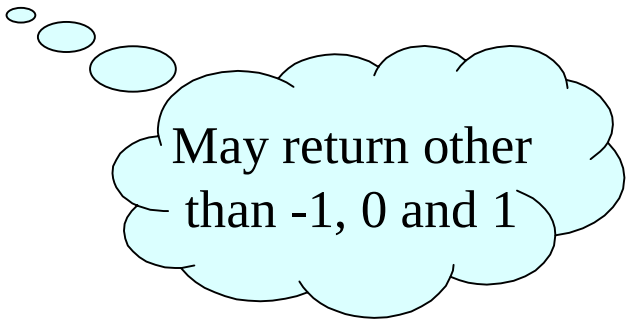
```
CFLAGS= -g -Wall

all : switch1 switch2 switch3

switch1 : reverse.o switch1.o
        gcc ${CFLAGS} -o switch1 $^
switch2 : reverse.o switch2.o
        gcc ${CFLAGS} -o switch2 $^
switch3 : reverse.o switch3.o
        gcc ${CFLAGS} -o switch3 $^
%.o : %.c
        gcc ${CFLAGS} -c $<
```

mystrcmp.c

```
int main(int argc, char **argv)
{
   char *result;
   if (argc != 3) {
      fprintf(stderr, "Usage: %s string1 string2\n", argv[0]);
      exit(1);
   }
   switch (mystrcmp(argv[1], argv[2])) {
   case -1: result = "less than"; break;
   case 0:  result = "equal to"; break;
   case 1:  result = "greater than"; break;
   default: result = "causing a problem comparing to"; break;
   }
   printf("%s is %s %s\n", argv[1], result, argv[2]);
   return 0;
}
```

mystrcmp.c (continued)

```
int mystrcmp (const char *a, const char *b)
{
   while (*a && *b && *a == *b) {
      a++; b++;
   }
   return (*a - *b);
}
```



mystrcmp.c (return -1,0, or 1)

```
int mystrcmp (const char *a, const char *b)
{
   while (*a && *b && *a == *b) {
      a++; b++;
   }
   if (*a < *b)
      return -1;
   else if (*a > *b)
      return 1;
   else /* a same as b */
      return 0;
}
```