CS 332/532 Systems Programming

Lecture 13

-Unix Files & Directories / 2-

Professor : Mahmut Unan – UAB CS

Agenda

- Open & Read Directories
- Function pointers - recall

Open & Read the directories

- Till now we talked about how filesystem store files and directories information and access details. Now it’s time to learn how to open and read the directories and traverse the file system. To achieve this task, you need to learn about three functions:
- *opendir* - this function will allow us to open a directory with the given path
- *readdir* - this function will read what’s inside the directory
- *closedir* - this will close the open directory

opendir

- opendir, fdopendir - open a directory

```
DIR *opendir(const char *name);
DIR *fdopendir(int fd);
```

The `opendir()` function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

The `fdopendir()` function is like opendir(), but returns a directory stream for the directory referred to by the open file descriptor *fd*. After a successful call to fdopendir(), *fd* is used internally by the implementation, and should not otherwise be used by the application.

https://man7.org/linux/man-pages/man3/opendir.3.html

readdir

- readdir - read a directory

```
struct dirent *readdir(DIR *dirp);
```

The readdir() function returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to by dirp.

On success, readdir() returns a pointer to a dirent structure. (This structure may be statically allocated; do not attempt to free(3) it.)

https://www.man7.org/linux/man-pages/man3/readdir.3.html

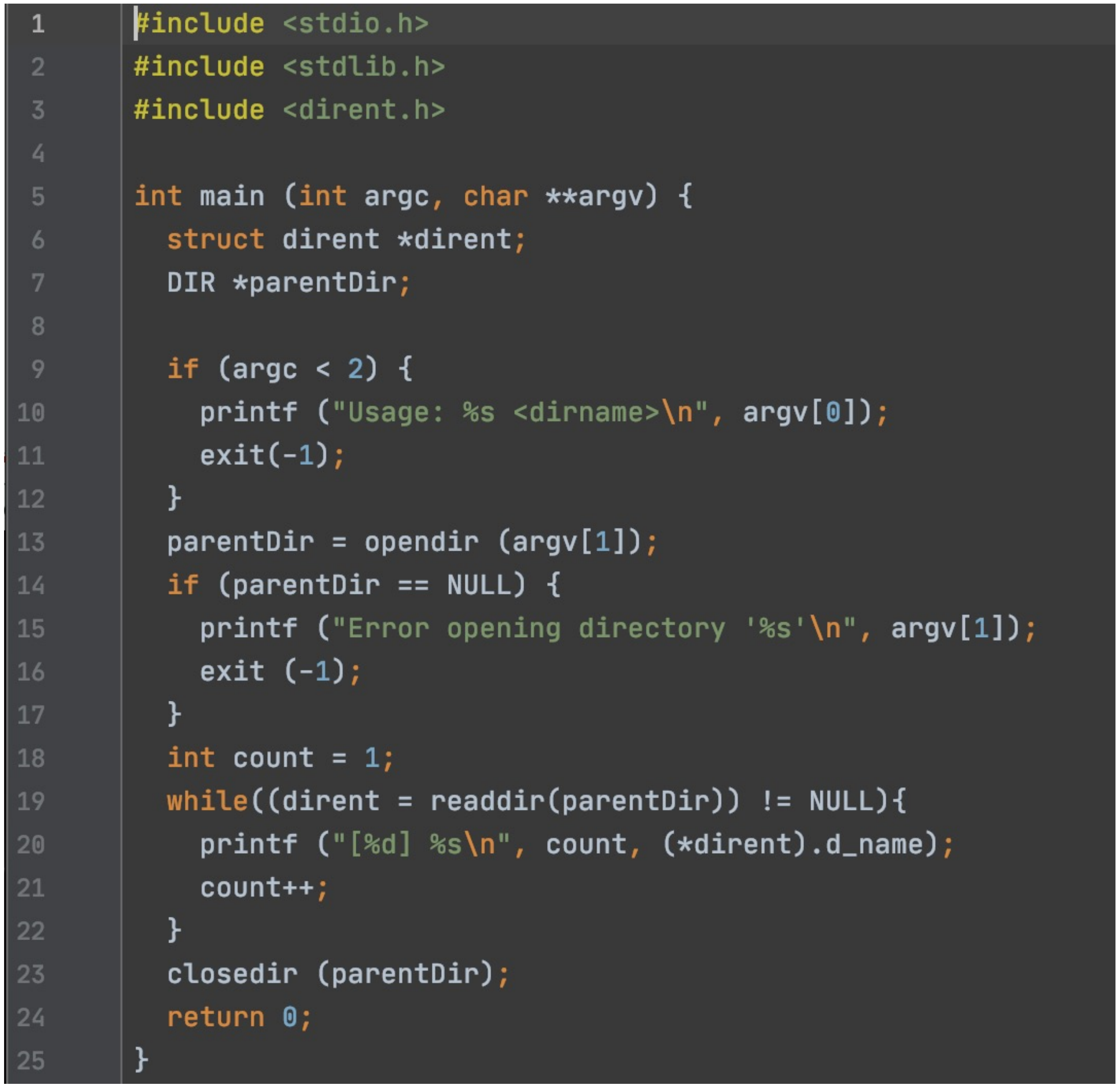
closedir

- closedir — close a directory stream

```
int closedir(DIR *dirp);
```

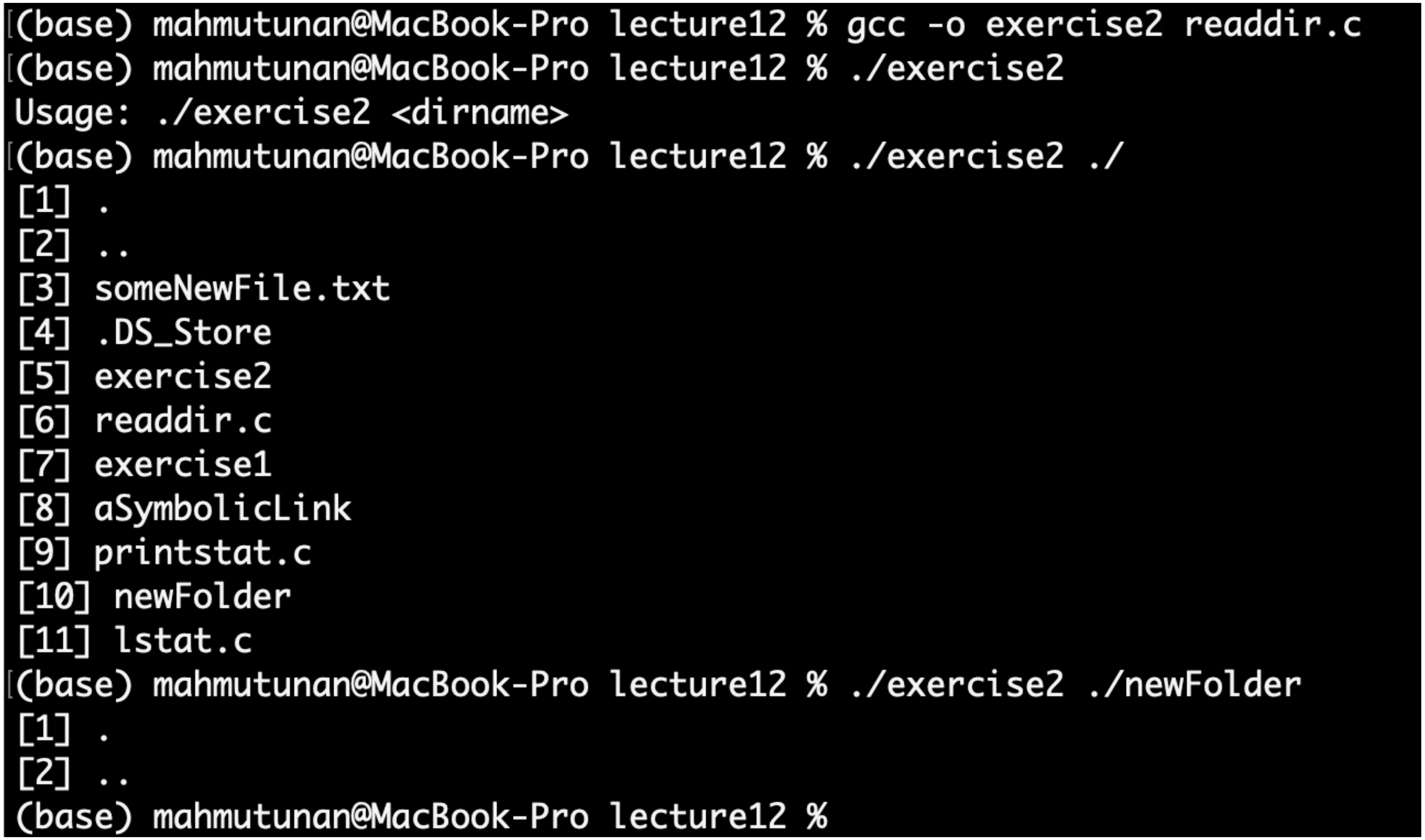
- The closedir() function shall close the directory stream referred to by the argument dirp. Upon return, the value of dirp may no longer point to an accessible object of the type DIR. If a file descriptor is used to implement type DIR, that file descriptor shall be closed.

Exercise 2 - readdir.c



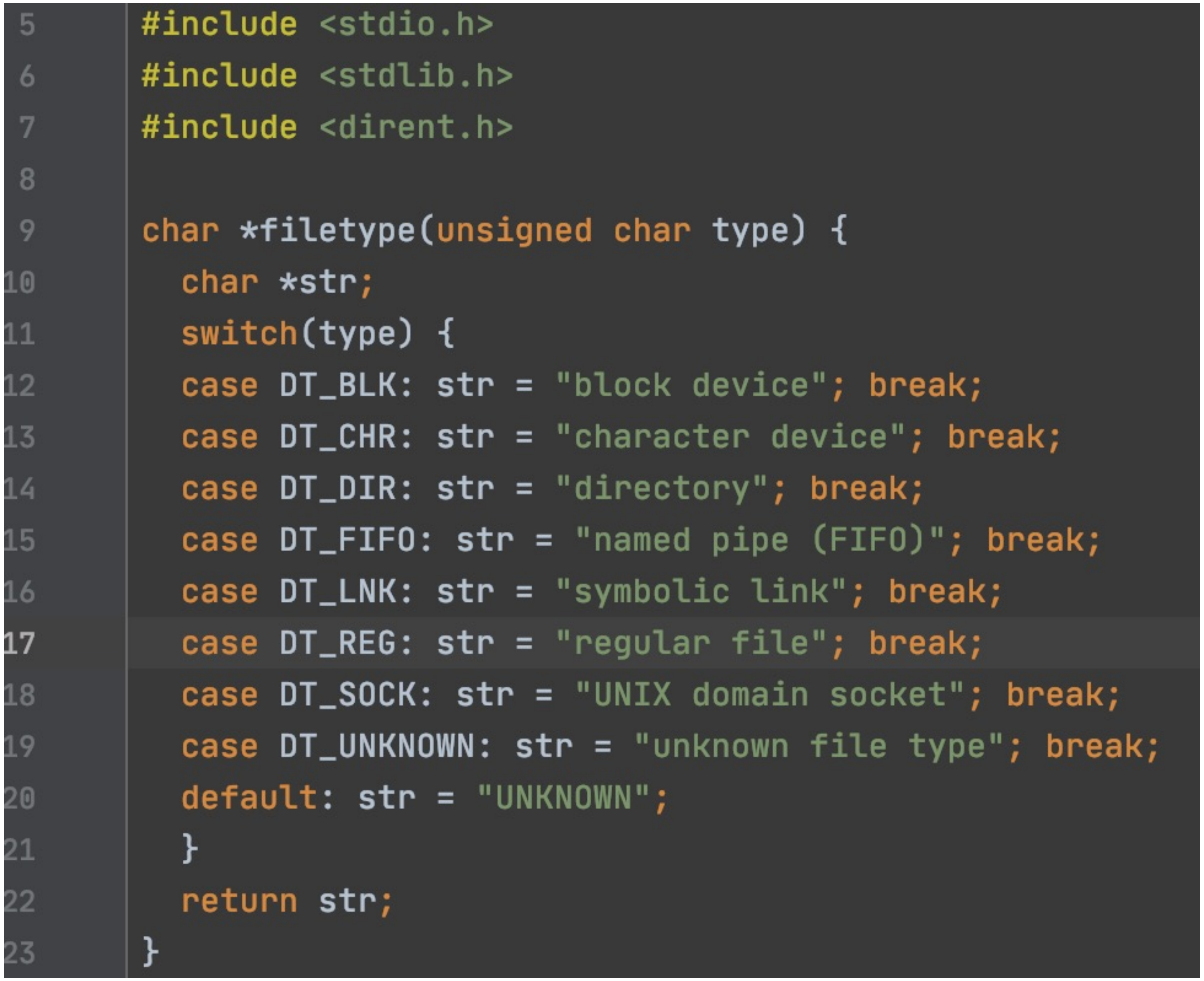
```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

int main (int argc, char **argv) {
  struct dirent *dirent;
  DIR *parentDir;

  if (argc < 2) {
    printf ("Usage: %s <dirname>\n", argv[0]);
    exit(-1);
  }
  parentDir = opendir (argv[1]);
  if (parentDir == NULL) {
    printf ("Error opening directory '%s'\n", argv[1]);
    exit (-1);
  }
  int count = 1;
  while((dirent = readdir(parentDir)) != NULL){
    printf ("[%d] %s\n", count, (*dirent).d_name);
    count++;
  }
  closedir (parentDir);
  return 0;
}
```

Exercise 2 - compile & run



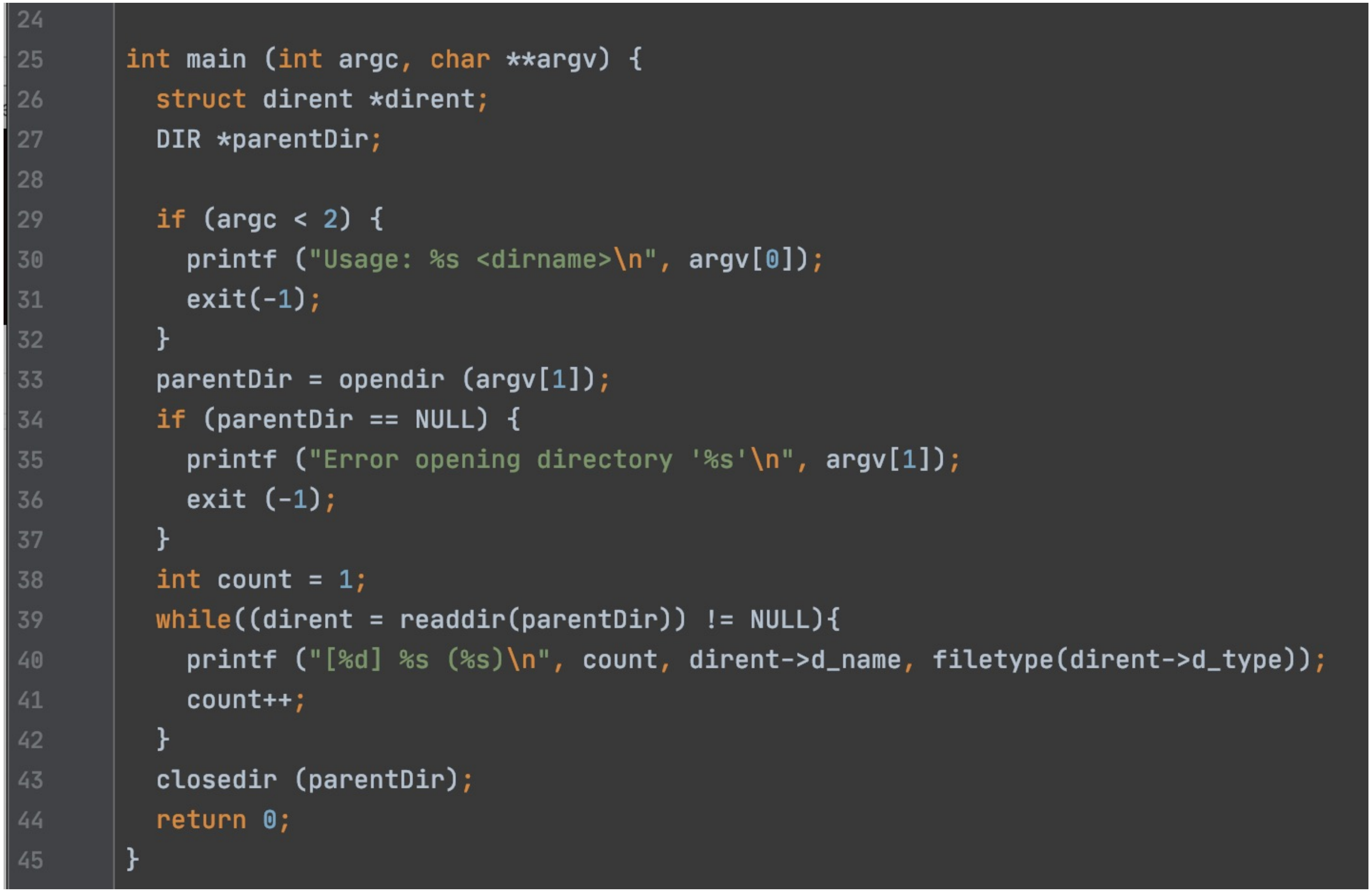
```
(base) mahmutunan@MacBook-Pro lecture12 % gcc -o exercise2 readdir.c
(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2
Usage: ./exercise2 <dirname>
(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2 ./
[1] .
[2] ..
[3] someNewFile.txt
[4] .DS_Store
[5] exercise2
[6] readdir.c
[7] exercise1
[8] aSymbolicLink
[9] printstat.c
[10] newFolder
[11] lstat.c
(base) mahmutunan@MacBook-Pro lecture12 % ./exercise2 ./newFolder
[1] .
[2] ..
(base) mahmutunan@MacBook-Pro lecture12 %
```

Exercise 3 - readdir_v2.c



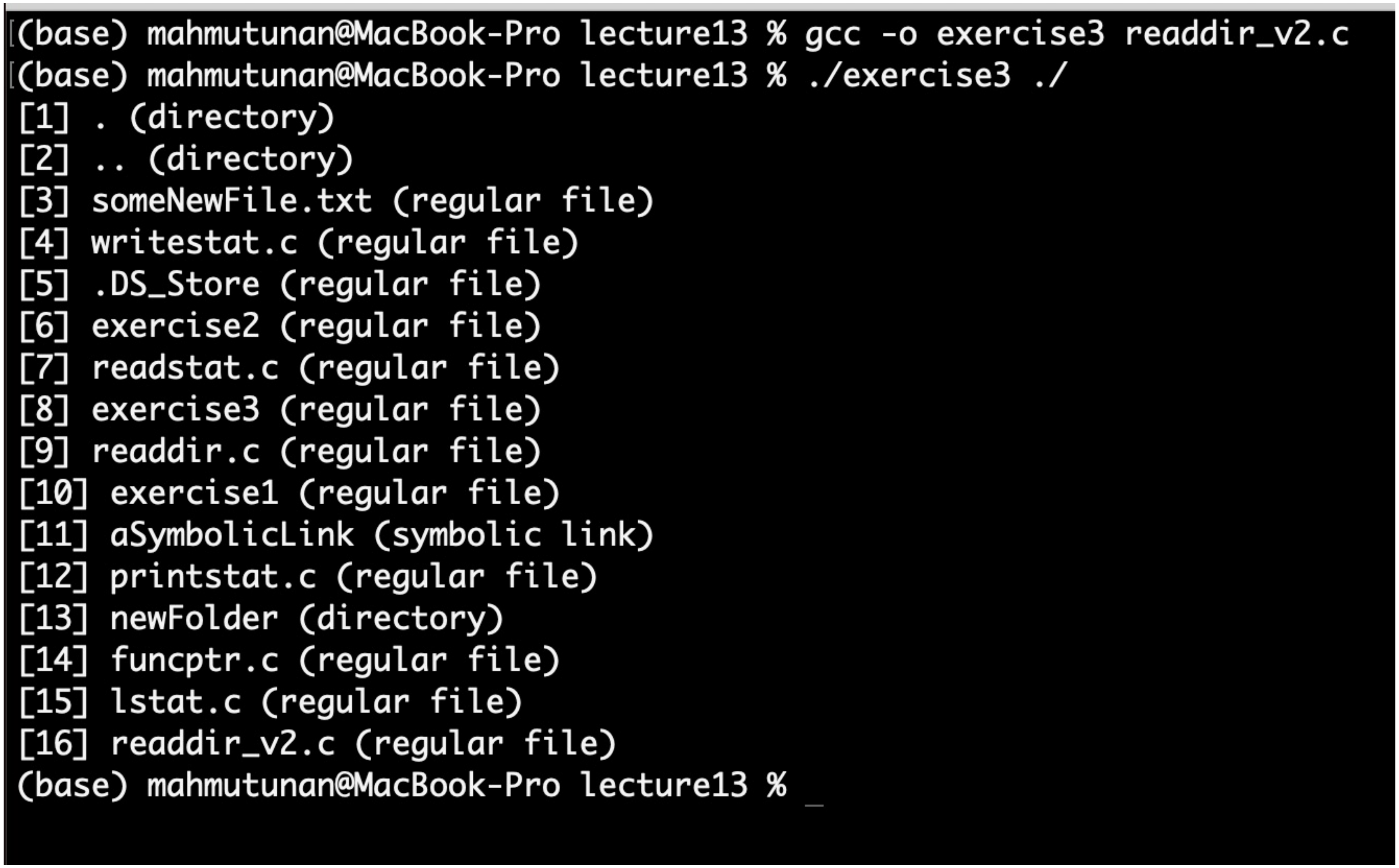
```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>

char *filetype(unsigned char type) {
  char *str;
  switch(type) {
  case DT_BLK: str = "block device"; break;
  case DT_CHR: str = "character device"; break;
  case DT_DIR: str = "directory"; break;
  case DT_FIFO: str = "named pipe (FIFO)"; break;
  case DT_LNK: str = "symbolic link"; break;
  case DT_REG: str = "regular file"; break;
  case DT_SOCK: str = "UNIX domain socket"; break;
  case DT_UNKNOWN: str = "unknown file type"; break;
  default: str = "UNKNOWN";
  }
  return str;
}
```

Exercise 3 - readdir_v2.c / 2



```c

int main (int argc, char **argv) {
  struct dirent *dirent;
  DIR *parentDir;

  if (argc < 2) {
    printf ("Usage: %s <dirname>\n", argv[0]);
    exit(-1);
  }
  parentDir = opendir (argv[1]);
  if (parentDir == NULL) {
    printf ("Error opening directory '%s'\n", argv[1]);
    exit (-1);
  }
  int count = 1;
  while((dirent = readdir(parentDir)) != NULL){
    printf ("[%d] %s (%s)\n", count, dirent->d_name, filetype(dirent->d_type));
    count++;
  }
  closedir (parentDir);
  return 0;
}
```

Exercise 3 - readdir_v2.c / compile&run



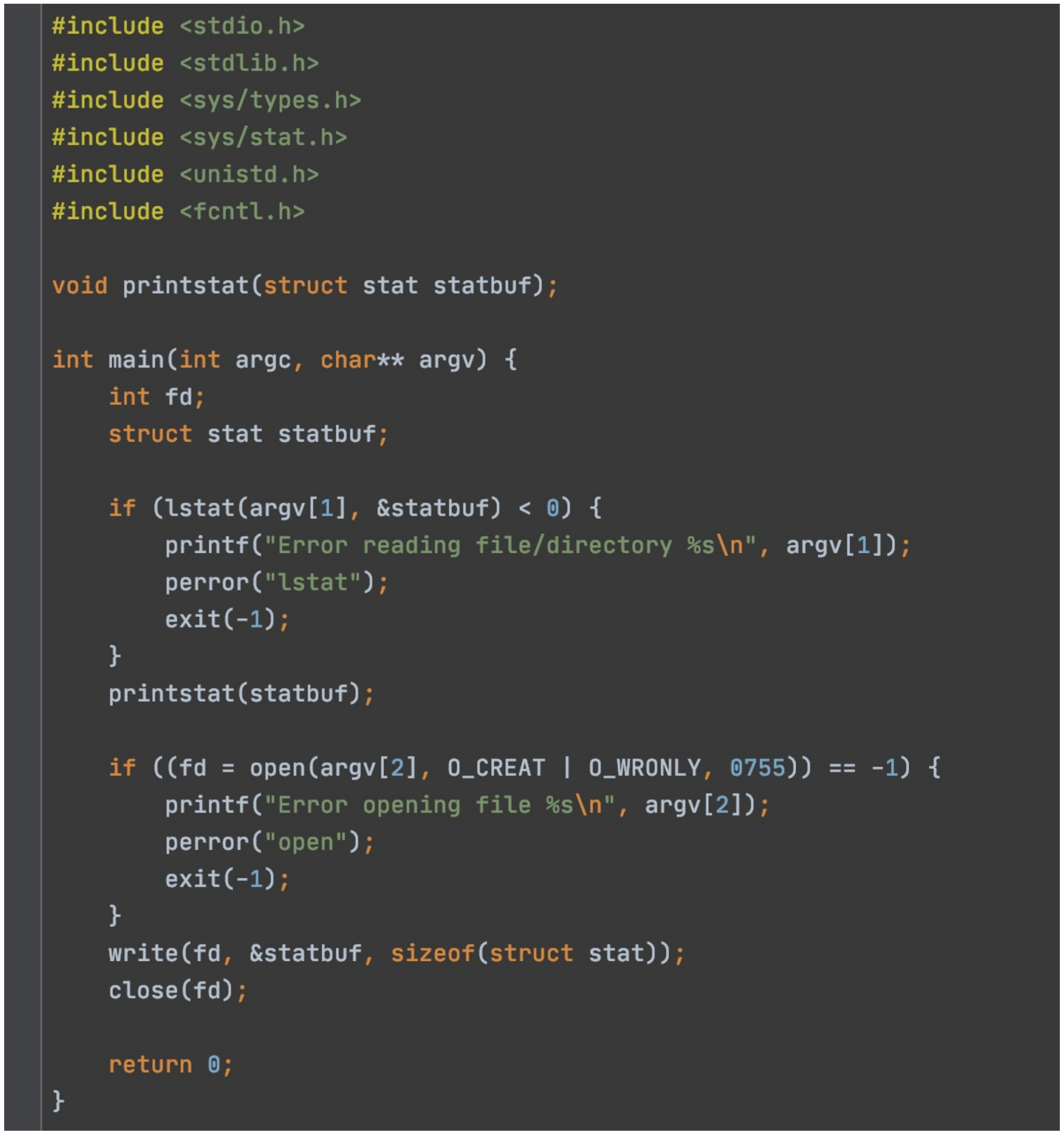
```
(base) mahmutunan@MacBook-Pro lecture13 % gcc -o exercise3 readdir_v2.c
(base) mahmutunan@MacBook-Pro lecture13 % ./exercise3 ./
[1] . (directory)
[2] .. (directory)
[3] someNewFile.txt (regular file)
[4] writestat.c (regular file)
[5] .DS_Store (regular file)
[6] exercise2 (regular file)
[7] readstat.c (regular file)
[8] exercise3 (regular file)
[9] readdir.c (regular file)
[10] exercise1 (regular file)
[11] aSymbolicLink (symbolic link)
[12] printstat.c (regular file)
[13] newFolder (directory)
[14] funcptr.c (regular file)
[15] lstat.c (regular file)
[16] readdir_v2.c (regular file)
(base) mahmutunan@MacBook-Pro lecture13 % _
```

Exercise 4

- We can use the read/write system calls from Lab-04 to write the stat structure and read the stat structure.

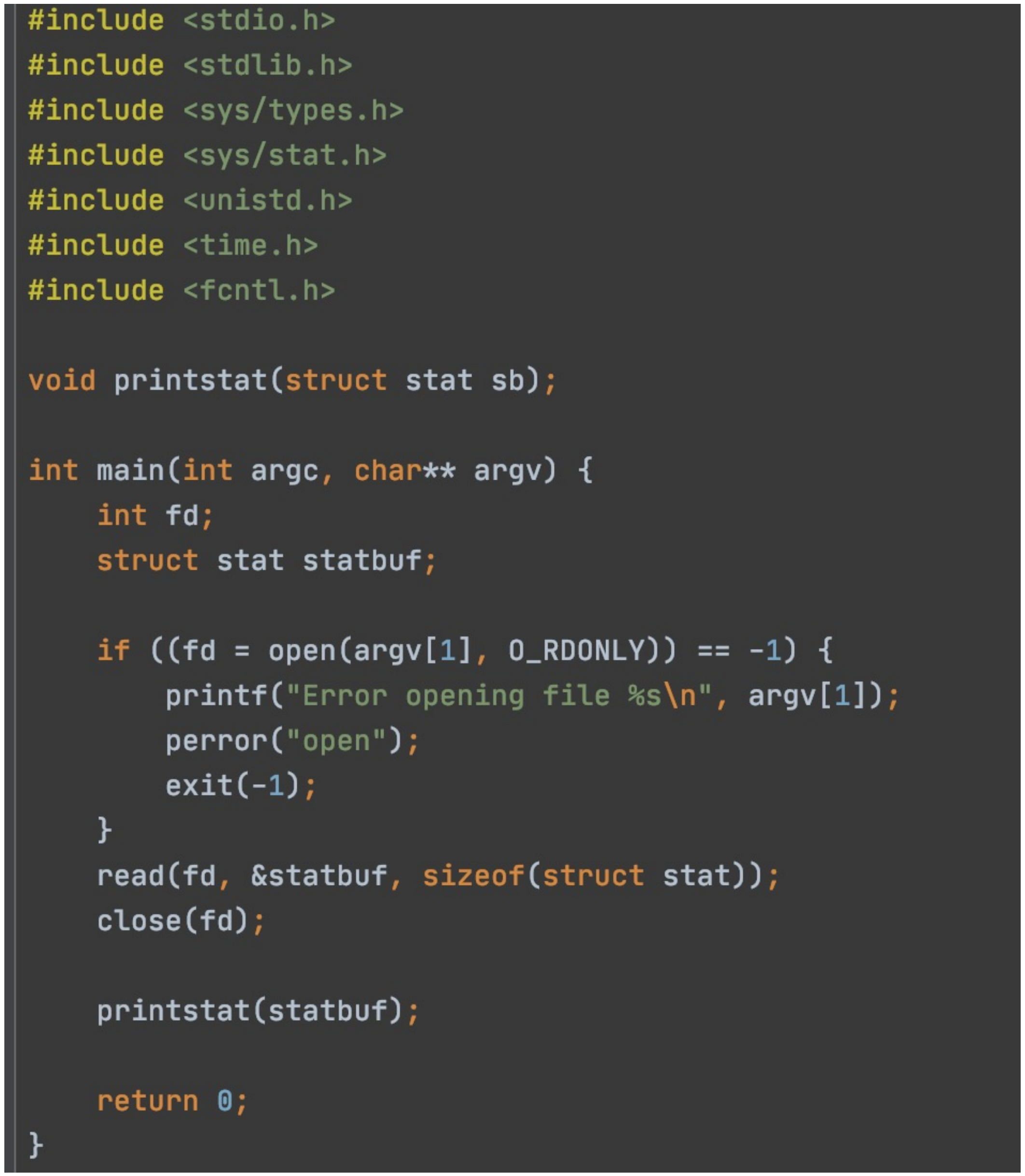
- Note that data is written as a binary file.

writestats.c



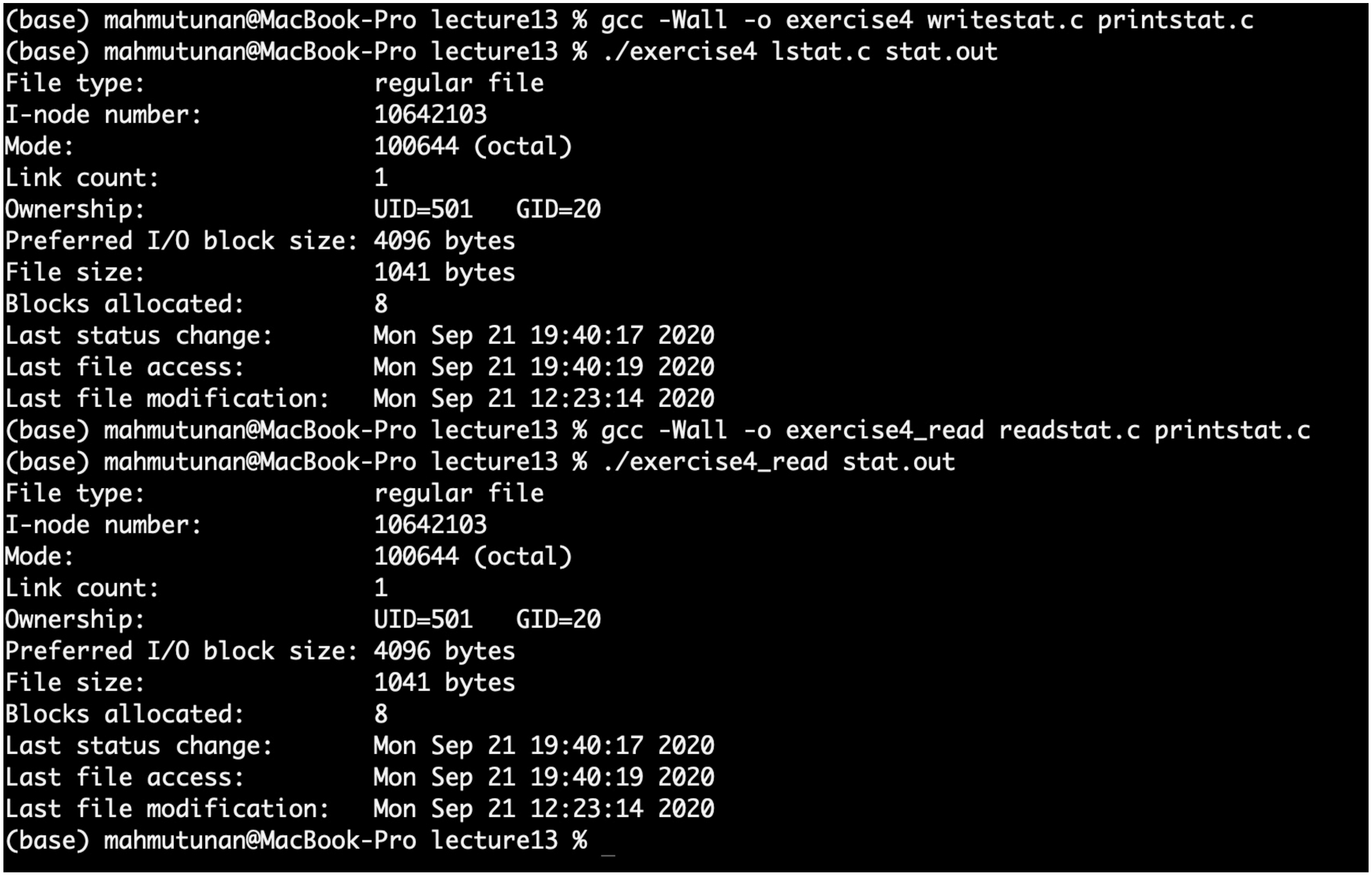
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

void printstat(struct stat statbuf);

int main(int argc, char** argv) {
    int fd;
    struct stat statbuf;

    if (lstat(argv[1], &statbuf) < 0) {
        printf("Error reading file/directory %s\n", argv[1]);
        perror("lstat");
        exit(-1);
    }
    printstat(statbuf);

    if ((fd = open(argv[2], O_CREAT | O_WRONLY, 0755)) == -1) {
        printf("Error opening file %s\n", argv[2]);
        perror("open");
        exit(-1);
    }
    write(fd, &statbuf, sizeof(struct stat));
    close(fd);

    return 0;
}
```

readstats.c



```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <fcntl.h>

void printstat(struct stat sb);

int main(int argc, char** argv) {
    int fd;
    struct stat statbuf;

    if ((fd = open(argv[1], O_RDONLY)) == -1) {
        printf("Error opening file %s\n", argv[1]);
        perror("open");
        exit(-1);
    }
    read(fd, &statbuf, sizeof(struct stat));
    close(fd);

    printstat(statbuf);

    return 0;
}
```

Exercise 4 - compile&run



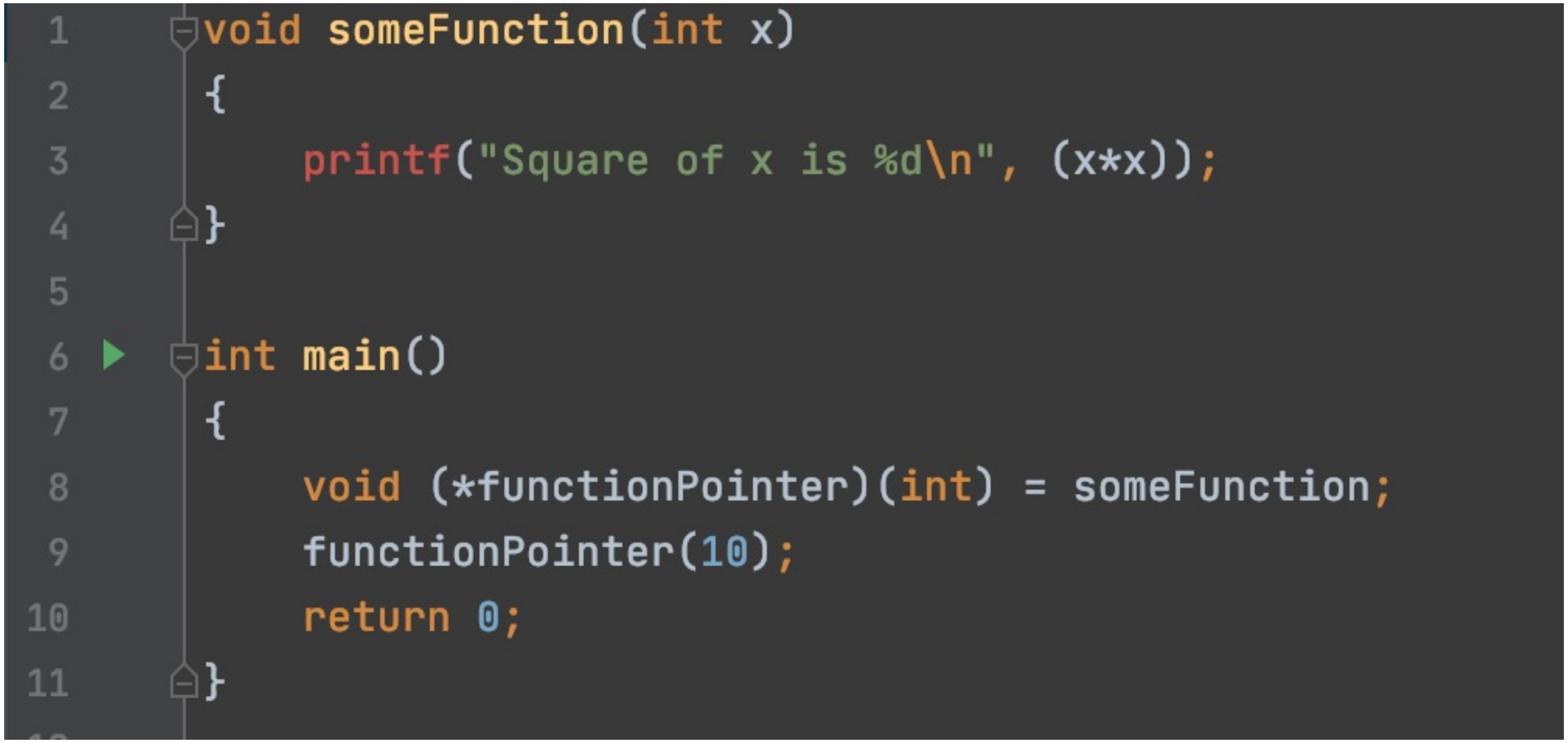
```
(base) mahmutunan@MacBook-Pro lecture13 % gcc -Wall -o exercise4 writestat.c printstat.c
(base) mahmutunan@MacBook-Pro lecture13 % ./exercise4 lstat.c stat.out
File type:                regular file
I-node number:            10642103
Mode:                     100644 (octal)
Link count:               1
Ownership:                UID=501   GID=20
Preferred I/O block size: 4096 bytes
File size:                1041 bytes
Blocks allocated:         8
Last status change:       Mon Sep 21 19:40:17 2020
Last file access:         Mon Sep 21 19:40:19 2020
Last file modification:   Mon Sep 21 12:23:14 2020
(base) mahmutunan@MacBook-Pro lecture13 % gcc -Wall -o exercise4_read readstat.c printstat.c
(base) mahmutunan@MacBook-Pro lecture13 % ./exercise4_read stat.out
File type:                regular file
I-node number:            10642103
Mode:                     100644 (octal)
Link count:               1
Ownership:                UID=501   GID=20
Preferred I/O block size: 4096 bytes
File size:                1041 bytes
Blocks allocated:         8
Last status change:       Mon Sep 21 19:40:17 2020
Last file access:         Mon Sep 21 19:40:19 2020
Last file modification:   Mon Sep 21 12:23:14 2020
(base) mahmutunan@MacBook-Pro lecture13 % _
```

Hints for the HW2

- You can find a more elaborate example in Figure 4.22 in the textbook.
- This program takes as input a directory name, traverses a file hierarchy, counts the different types of files in the given file hierarchy, and the prints the summary (as shown in Figure 4.4).
- This program uses function pointers i.e., you can pass a function as an argument to a function similar to passing variables of different type.
- This enables us to perform different operations on a file as we traverse the file hierarchy.

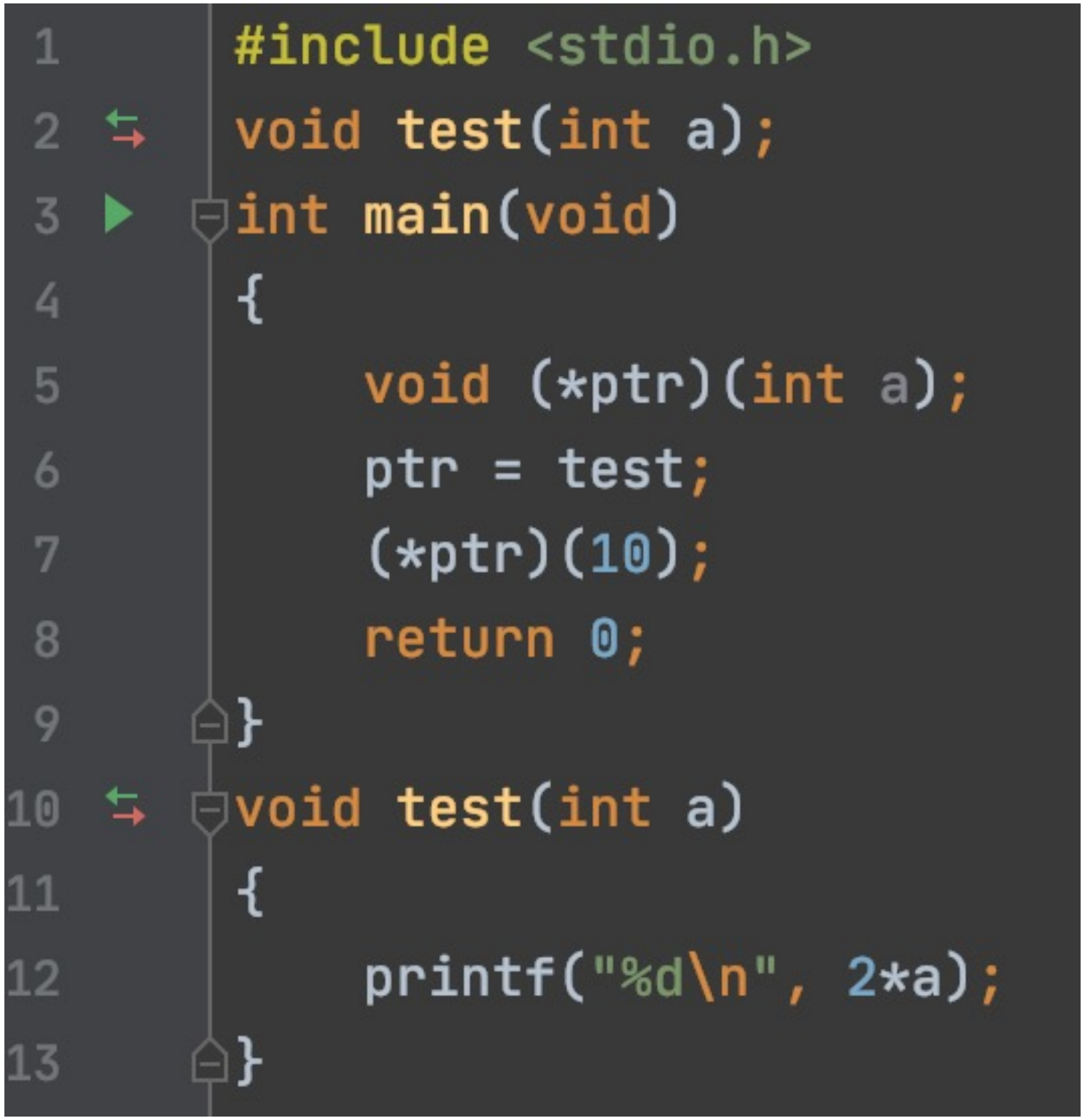
Function Pointers - recall

- In the C environment, like normal data pointers, we can have pointers to functions

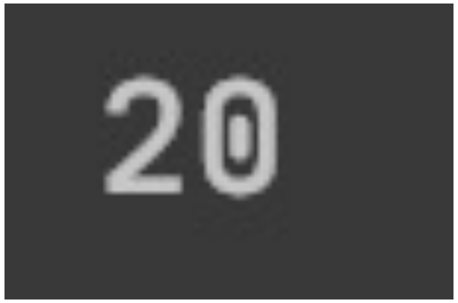


```c
void someFunction(int x)
{
    printf("Square of x is %d\n", (x*x));
}

int main()
{
    void (*functionPointer)(int) = someFunction;
    functionPointer(10);
    return 0;
}
```

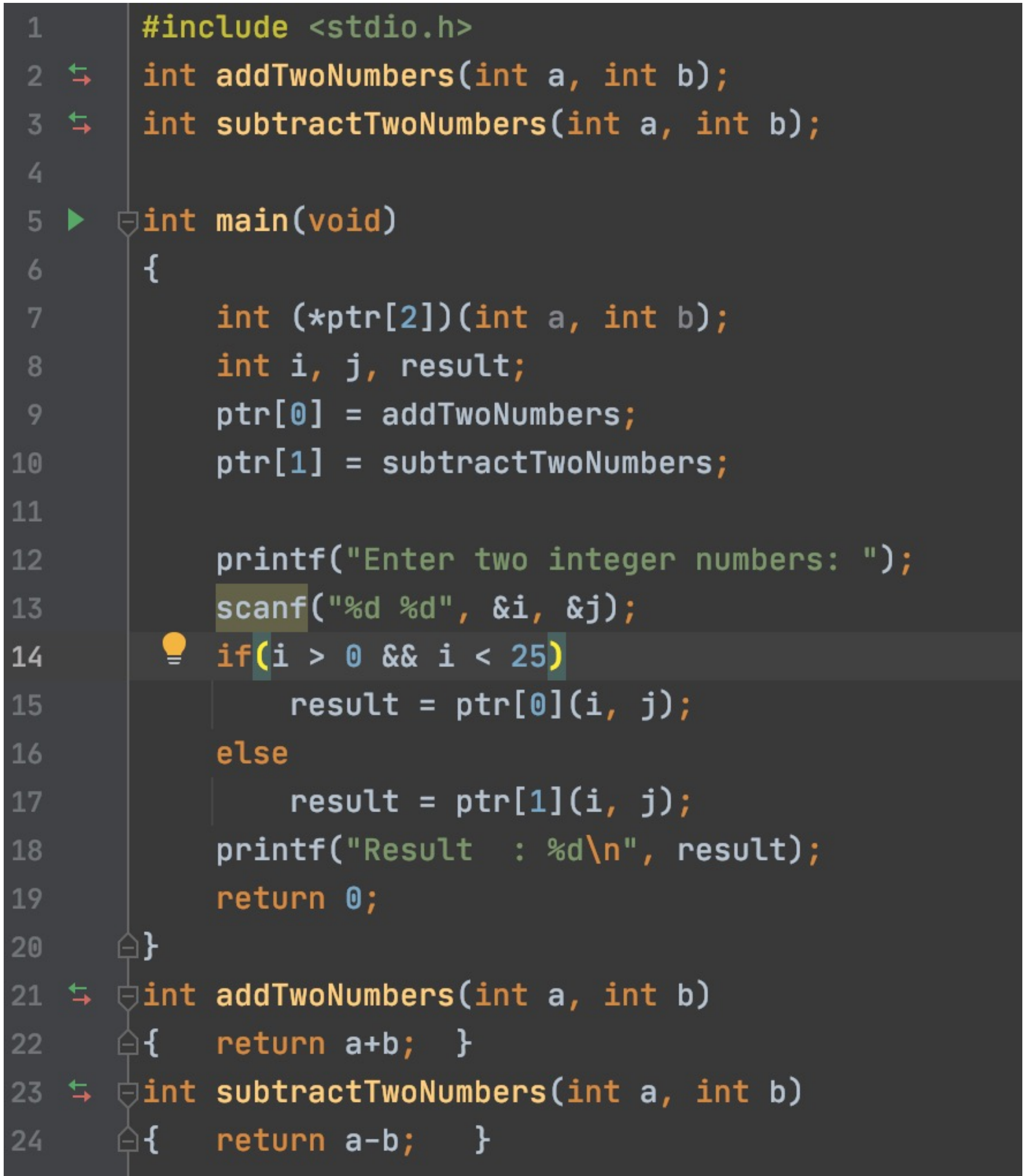
Passing Pointers to functions



```c
#include <stdio.h>
void test(int a);
int main(void)
{
    void (*ptr)(int a);
    ptr = test;
    (*ptr)(10);
    return 0;
}
void test(int a)
{
    printf("%d\n", 2*a);
}
```



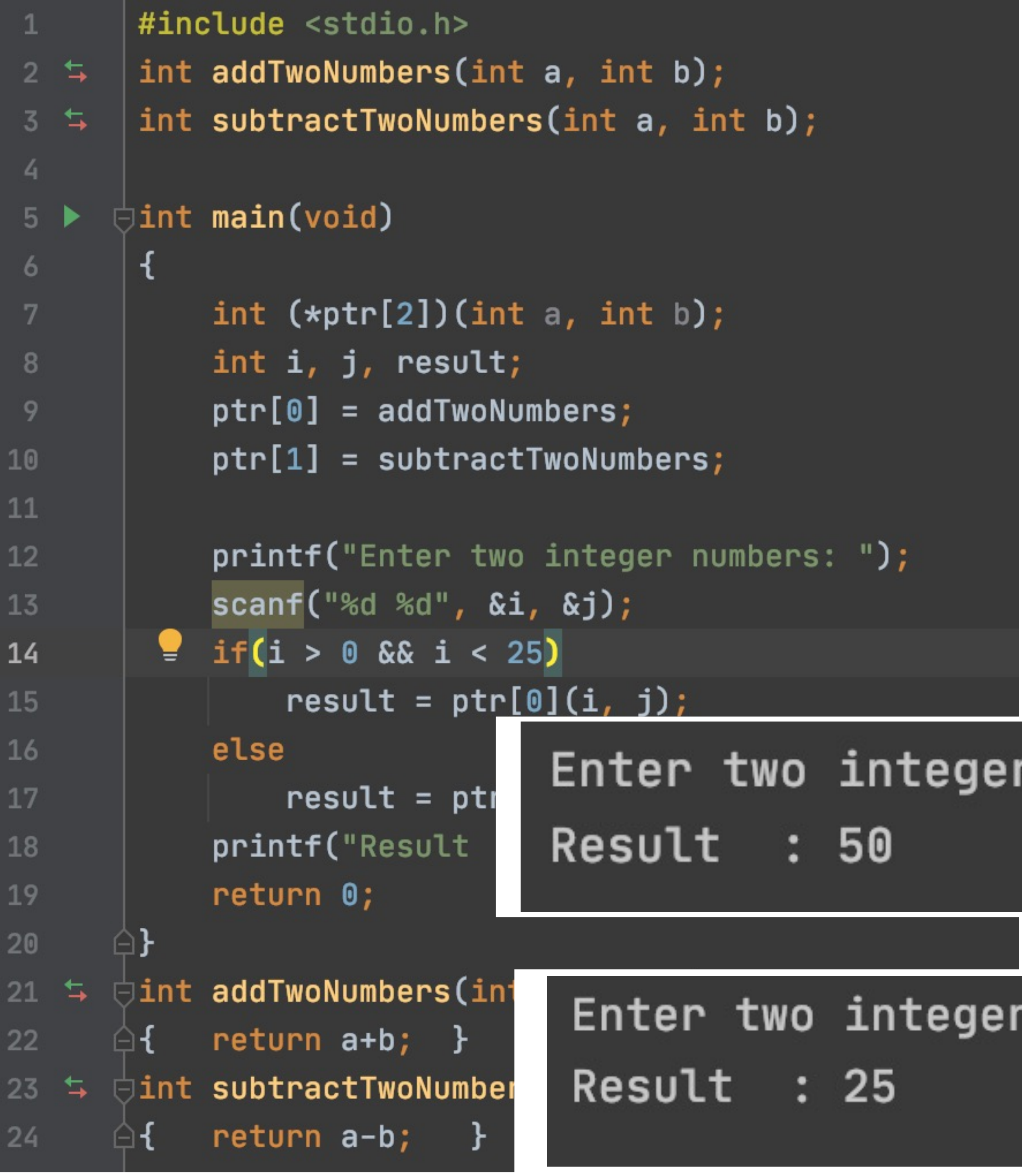
```
20
```



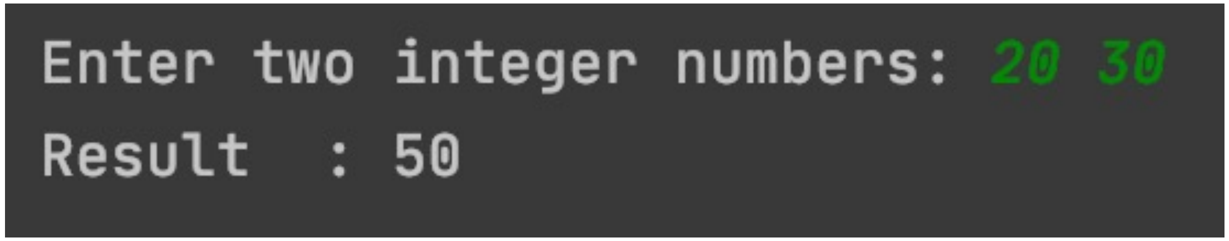
```c
#include <stdio.h>
int addTwoNumbers(int a, int b);
int subtractTwoNumbers(int a, int b);

int main(void)
{
    int (*ptr[2])(int a, int b);
    int i, j, result;
    ptr[0] = addTwoNumbers;
    ptr[1] = subtractTwoNumbers;

    printf("Enter two integer numbers: ");
    scanf("%d %d", &i, &j);
    if(i > 0 && i < 25)
        result = ptr[0](i, j);
    else
        result = ptr[1](i, j);
    printf("Result  : %d\n", result);
    return 0;
}
int addTwoNumbers(int a, int b)
{   return a+b;  }
int subtractTwoNumbers(int a, int b)
{   return a-b;   }
```

Lecture06

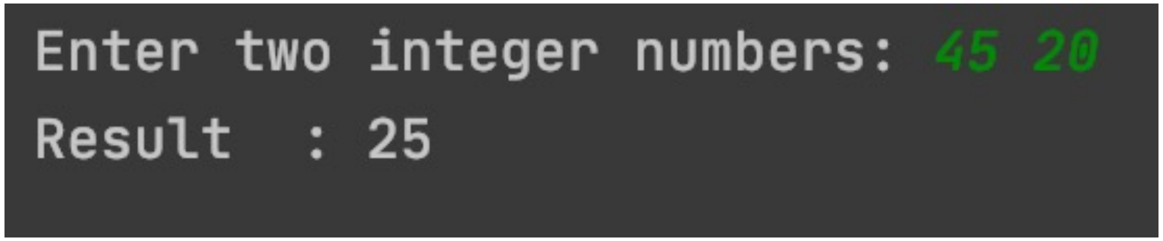
Lecture06



```
#include <stdio.h>
int addTwoNumbers(int a, int b);
int subtractTwoNumbers(int a, int b);

int main(void)
{
    int (*ptr[2])(int a, int b);
    int i, j, result;
    ptr[0] = addTwoNumbers;
    ptr[1] = subtractTwoNumbers;

    printf("Enter two integer numbers: ");
    scanf("%d %d", &i, &j);
    if(i > 0 && i < 25)
        result = ptr[0](i, j);
    else
        result = ptr
    printf("Result
    return 0;
}
int addTwoNumbers(int
{   return a+b;  }
int subtractTwoNumber
{   return a-b;   }
```

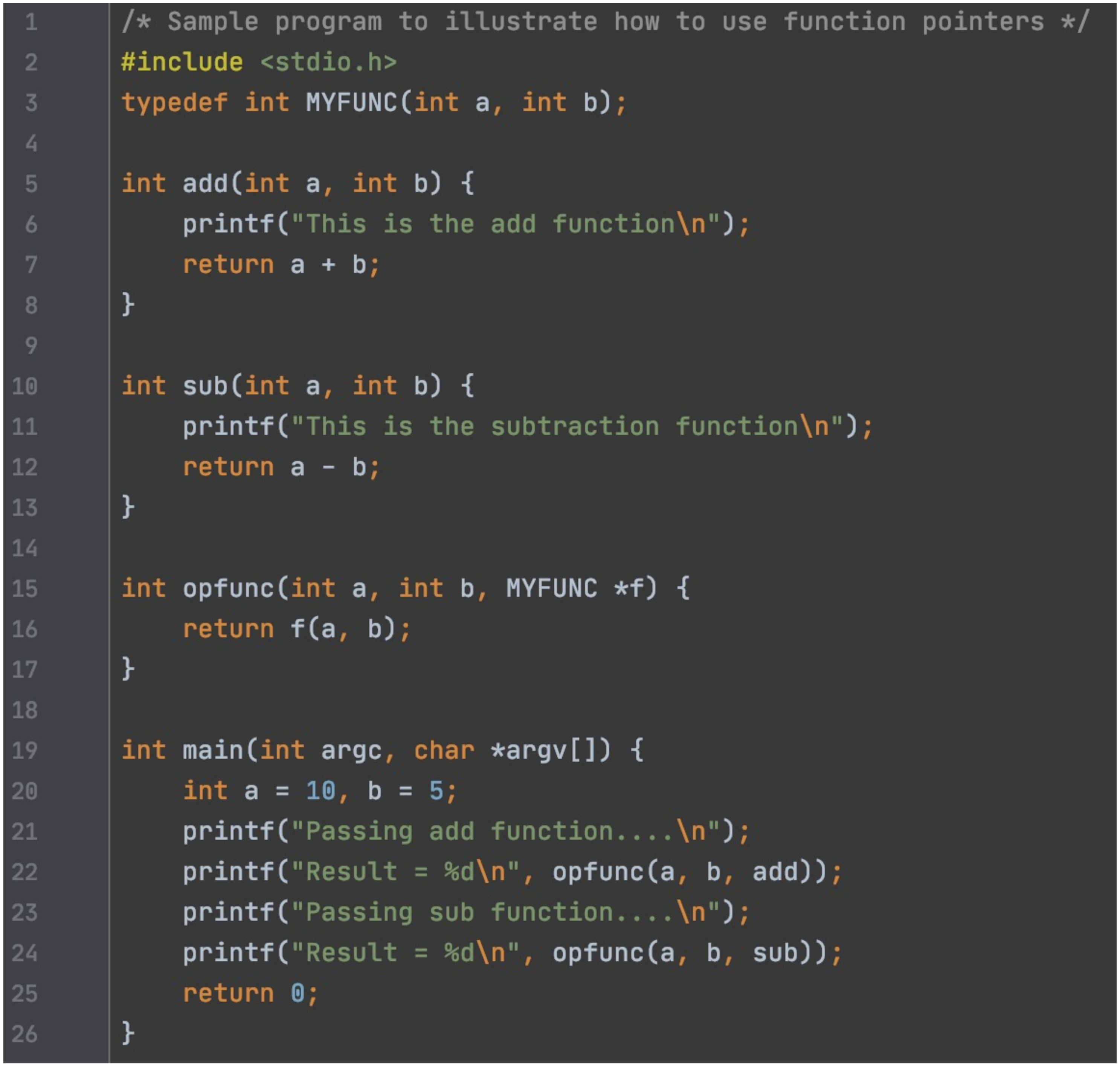


```
Enter two integer numbers: 20 30
Result  : 50
```



```
Enter two integer numbers: 45 20
Result  : 25
```

Exercise 5



```c
/* Sample program to illustrate how to use function pointers */
#include <stdio.h>
typedef int MYFUNC(int a, int b);

int add(int a, int b) {
    printf("This is the add function\n");
    return a + b;
}

int sub(int a, int b) {
    printf("This is the subtraction function\n");
    return a - b;
}

int opfunc(int a, int b, MYFUNC *f) {
    return f(a, b);
}

int main(int argc, char *argv[]) {
    int a = 10, b = 5;
    printf("Passing add function....\n");
    printf("Result = %d\n", opfunc(a, b, add));
    printf("Passing sub function....\n");
    printf("Result = %d\n", opfunc(a, b, sub));
    return 0;
}
```

- In this example we define a function *opfunc* that takes as input a pointer to a function that takes two integer arguments and returns an integer value.
- We use *typedef* to define the function signature so that we can use this as a type in the function definition.
- Then we can have different functions with the given type signature and these functions can perform different operations. In this example, we define two functions to perform addition and subtraction on the two arguments passed to the function.
- Now we can invoke the *opfunc* by providing two integer values and the corresponding function to perform the required operation.

- Example 4.22 uses this mechanism through which we can define different functions to perform different operations on a given file as we traverse the file hierarchy.
- In this example, we just count the different files types, however, we could perform other operations such as check the file size or file permission or any other user-defined operation.