CS214 Recitation (Sec.7)

Sept.19.2017

Topics of this week

Basic LINUX & shell commands

Man pages

C data types (arrays, unions, enums)

Basic LINUX & shell commands

Shell is a program that interprets commands

It can execute commands manually typed by users in a terminal or programs called shell scripts

Shell is a way to interface with operating system and execute commands or scripts

Basic LINUX & shell commands (Cont.)

In LINUX, the shell is called BASH

BASH is short for Bourne Again SHell

Need to notice about BASH:

- Case Sensitivity (commands & filenames) PWD ≠ pwd
- Directory separator is “/” - forward-slash such as /usr/src/linux
- Filename not need to be “filename.txt” style   filename.xxx.xxx.txt is OK

Basic LINUX & shell commands (Cont.)

Special Characters:

- \   back-slash is escape character   rm *   vs   rm *
- /   forward-slash is directory character   /usr/src/linux
- .   period reps current directory
- .. double periods rep parent directory
- ~ your home directory
- *   asterisk reps 0 or more characters in a filename   eg. pic*2017   *

Basic LINUX & shell commands (Cont.)

Special Characters:

- ? question mark reps a single character eg. hello?.txt
- [ ] square brackets rep a range of values eg. hello[3-5].txt reps hello3.txt, hello4.txt, hello5.txt
- | vertical bar reps pipe --- redirect the output of one command into another command eg. ls | more

Basic LINUX & shell commands (Cont.)

Most commands are located in your shell’s path, you can just type the name to execute it   eg. type “ls” to execute command “ls”

To execute programs in your current path   eg. ./program

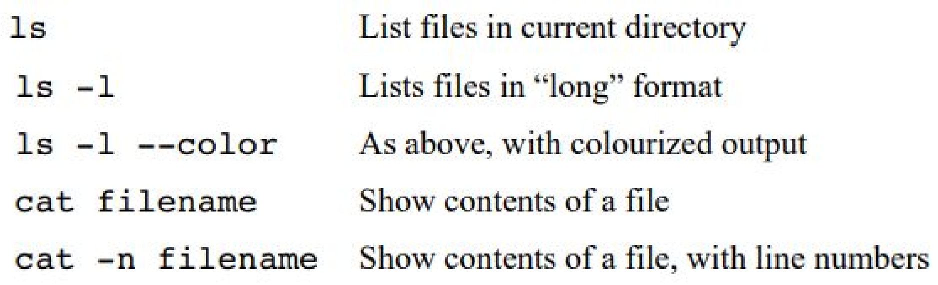
To execute programs or commands not your current path, you need to give the complete location   eg.   /home/shijie/program

Basic LINUX & shell commands (Cont.)

Execute command with arguments

Syntax:

- command [-argument] [-argument] [--argument] [filename]



`ls`	List files in current directory
`ls -l`	Lists files in “long” format
`ls -l --color`	As above, with colourized output
`cat filename`	Show contents of a file
`cat -n filename`	Show contents of a file, with line numbers

Manual Page

Two ways to call out the help information

- man command   eg. man date
- don’t know how to use manual page eg. man man
- command --help eg. date --help
- type q to quit information page

Some useful shell commands

- **pwd** print working directory
- **cd** change directory
- **cd ~** change directory to your home directory
- **cd ..** move up one directory /home/shijie -> /home
- **cd directory** change to specified directory
- **ls** list all files in current directory
- **ls directory** list all files in specified directory
- **ls -l** list files in “long” format, one file per line
- **ls -a** list all files including hidden files
- **ls directory/d*** list all files whose names begin with d in specified directory

Some useful shell commands (Cont.)

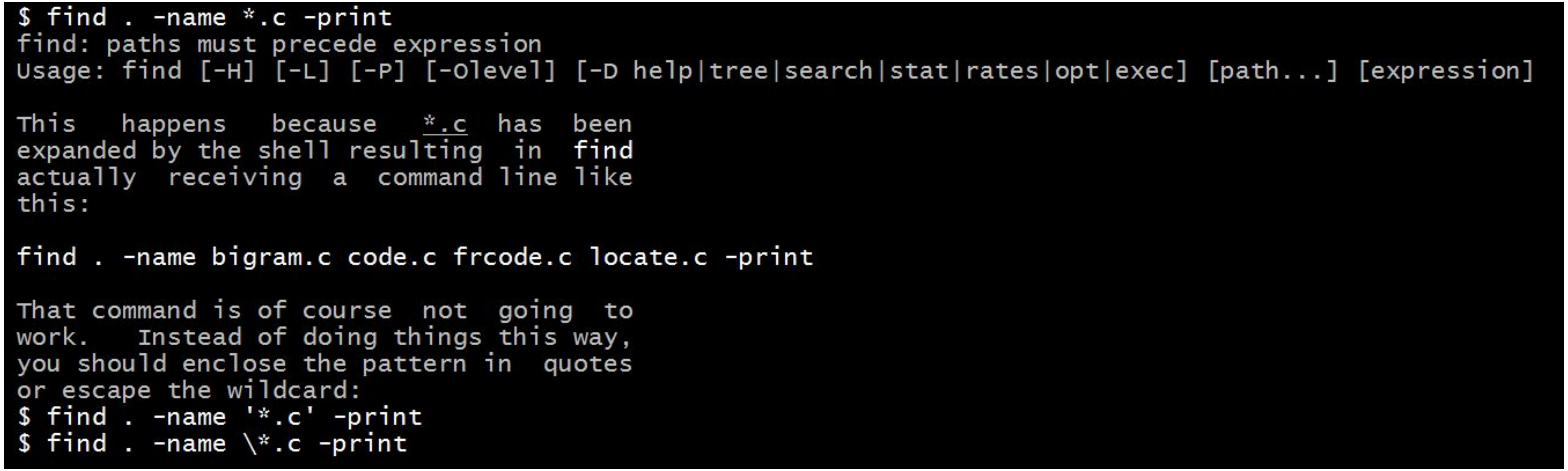
- **ls -la directory | less** (or more)   gives a long listing of all files in specified directory, then pipe the output to program “less” to display output for one screen
- **cat**   displays the contents of a text file   eg. cat test.c
- **cat /proc/cpuinfo**   displays information about your CPU
- **cat /proc/meminfo** displays information about your memory usage
- **file**   tells what kind of file it is   eg. file /bin/ls   file test.c
- **clear** clear the screen
- **echo**   display text on the screen   eg. echo “good”   echo $x
- **grep**   search for a pattern in a file and displays those lines contain the pattern eg. grep “printf” hw0.c

Some useful shell commands (Cont.)

- **cp** copies a file from one location to another eg. cp test.c /tmp
- **mv** moves a file to a new location eg. mv test.c /tmp
- **rm** delete a file eg. rm /tmp/test.c rm *
- **mkdir** make directory
- **rmdir** remove directory
- **df** displays filesystem disk space usage
- **su** switch the root account
- **find** search for files matching certain patterns eg. find . -name *c

Some useful shell commands (Cont.)

- **find** search for files matching certain patterns eg. find . -name *.c

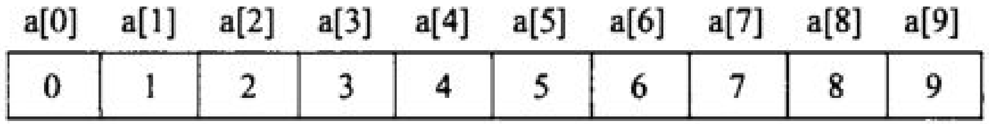


```
$ find . -name *.c -print
find: paths must precede expression
Usage: find [-H] [-L] [-P] [-Olevel] [-D help|tree|search|stat|rates|opt|exec] [path...] [expression]

This   happens   because   *.c  has  been
expanded by the shell resulting  in  find
actually  receiving  a  command line like
this:

find . -name bigram.c code.c frcode.c locate.c -print

That command is of course  not  going  to
work.   Instead of doing things this way,
you should enclose the pattern in  quotes
or escape the wildcard:
$ find . -name '*.c' -print
$ find . -name \*.c -print
```

C data types

arrays

int a[10] = {0,1,2,3,4,5,6,7,8,9} ;



char c[10] = {'0','1','2','3','4'}; what are the values of the other 5 elements?

int a[10] = {0};

int a[] = {1,2,3,4,5}; how many elements are in array a?

char s[5]; char *str = s; is a pointer to the address of first element of array s

str[0]='1', *(s+1)='2' *array with fixed size*

C data types (Cont.)

arrays

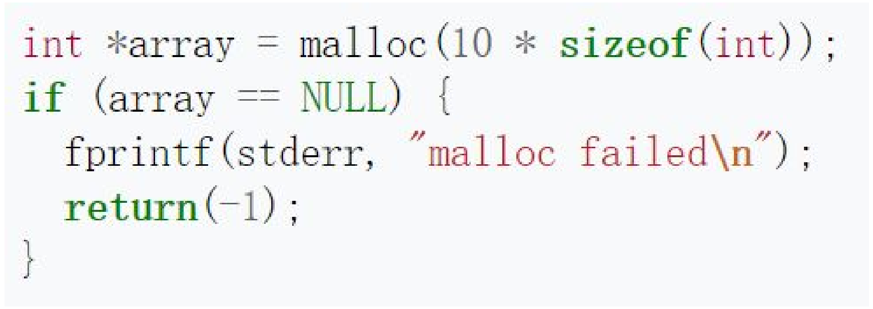
initialize an array with dynamically allocated memory

- #include <stdlib.h>
- use ***malloc*** to allocate a consecutive block of memory of the specified number of bytes ***char *str = (char*)malloc(100*sizeof(char));***
- use ***free*** to release the specified block of memory back to the system ***free(str);***

C data types (Cont.)

arrays

- Since *malloc* might not be able to service your request, it is good to check for this:



```
int *array = malloc(10 * sizeof(int));
if (array == NULL) {
  fprintf(stderr, "malloc failed\n");
  return(-1);
}
```

C data types (Cont.)

unions

- A union is a special data type available in C that allows to store different data types in the same memory location.
- *union Data {*

  ```
  union Data {
     int i;
     float f;
     char str[20];
  } data;
  ```

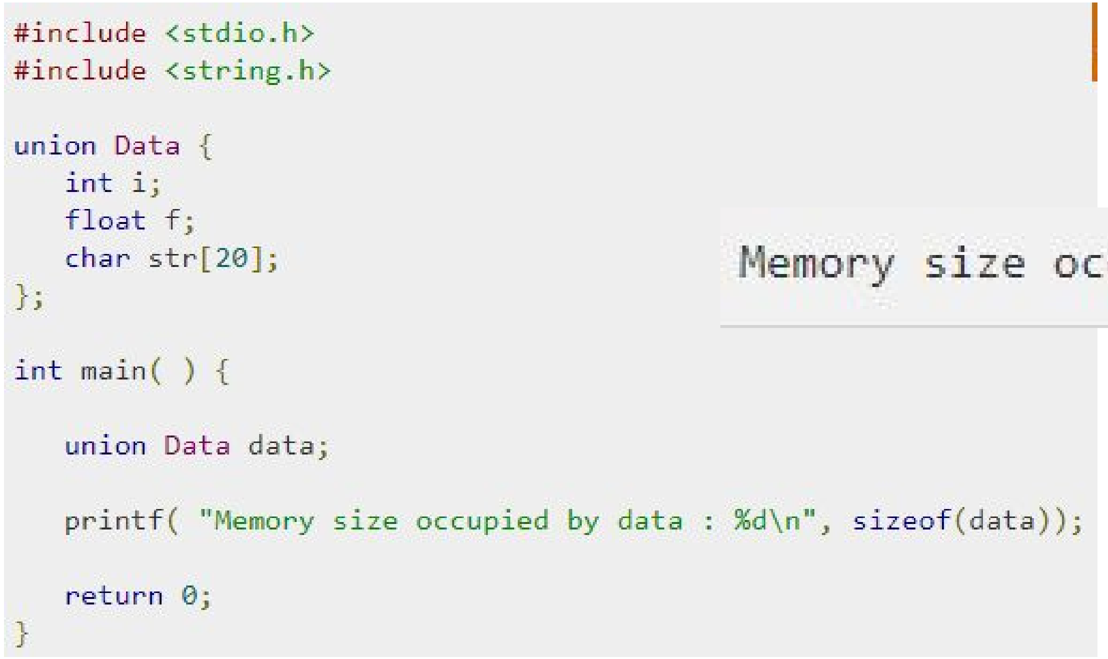
C data types (Cont.)

unions

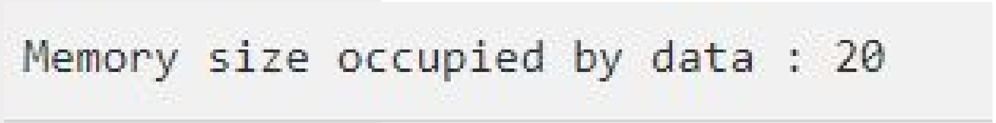
- You can define a union with many members, but only one member can contain a value at any given time.
- The final value assigned to the variable will occupy the memory location
- Unions provide an efficient way of using the same memory location for multiple-purpose.

C data types (Cont.)

unions



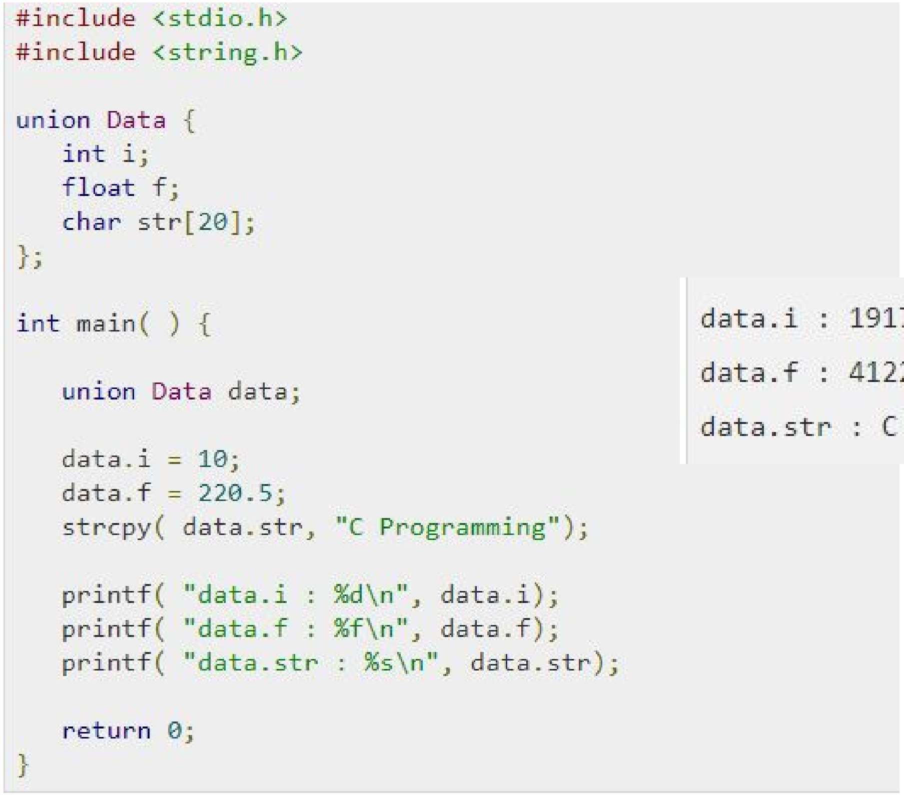
```
#include <stdio.h>
#include <string.h>

union Data {
   int i;
   float f;
   char str[20];
};

int main( ) {

   union Data data;

   printf( "Memory size occupied by data : %d\n", sizeof(data));

   return 0;
}
```



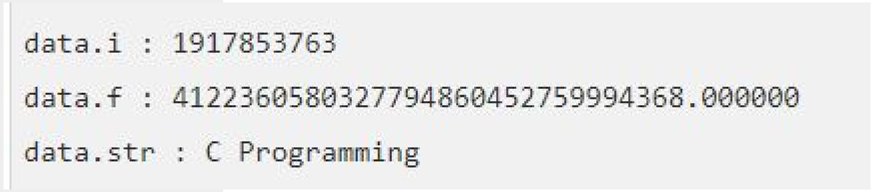
```
Memory size occupied by data : 20
```

C data types (Cont.)

unions



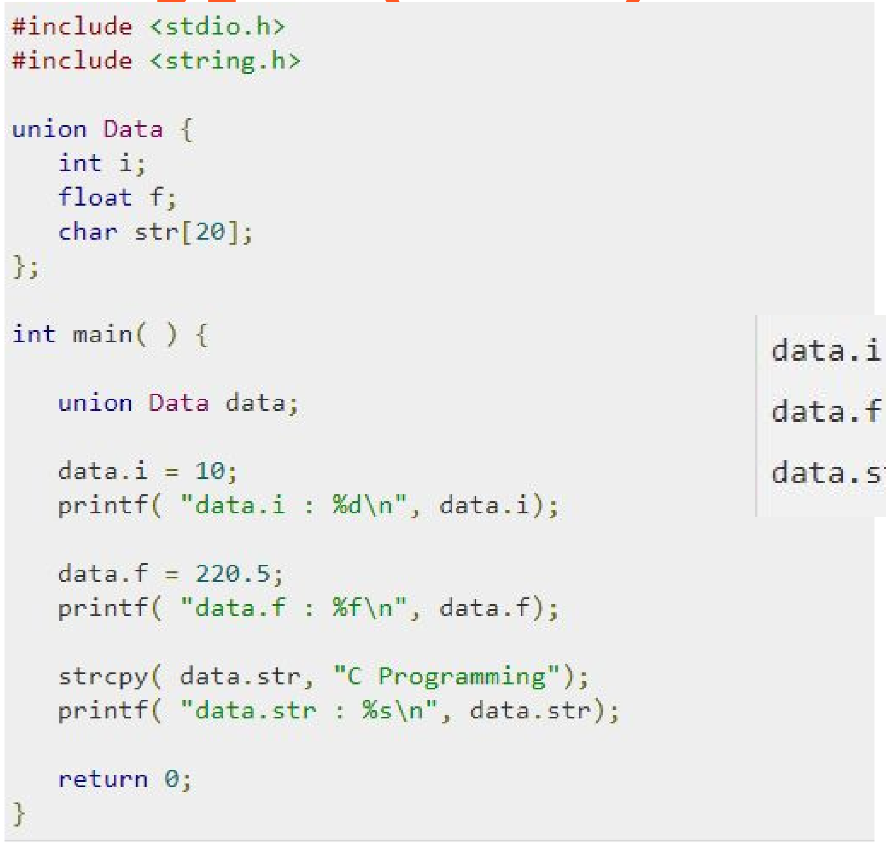
```
#include <stdio.h>
#include <string.h>

union Data {
   int i;
   float f;
   char str[20];
};

int main( ) {

   union Data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```



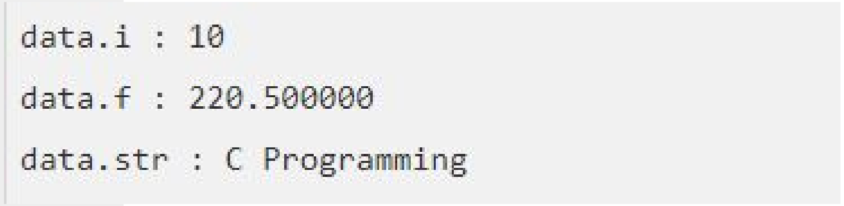
```
data.i : 1917853763

data.f : 4122360580327794860452759994368.000000

data.str : C Programming
```

C data types (Cont.)

unions



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

union Data {
   int i;
   float f;
   char str[20];
};

int main( ) {

   union Data data;

   data.i = 10;
   printf( "data.i : %d\n", data.i);

   data.f = 220.5;
   printf( "data.f : %f\n", data.f);

   strcpy( data.str, "C Programming");
   printf( "data.str : %s\n", data.str);

   return 0;
}
```



```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

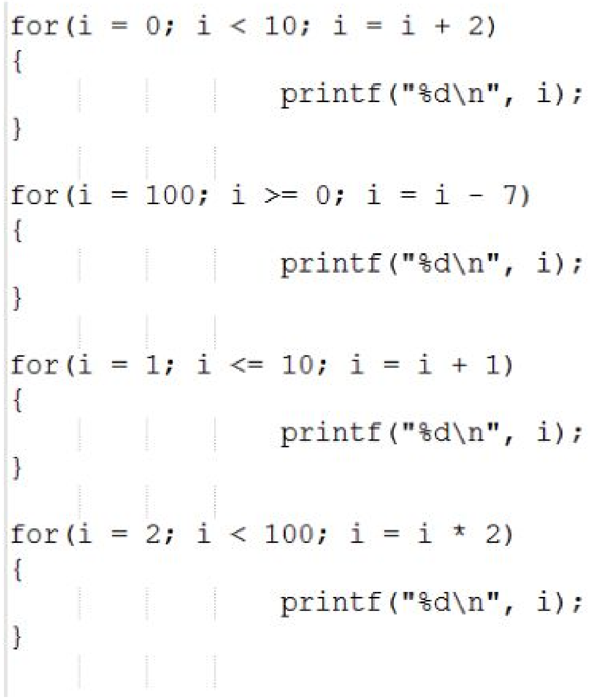
C data types (Cont.)

enums - enumerated type

- ***enum Weekday {sun,mon,tue,wed,thu,fri,sat}; enum Weekday workday, weekend;***
- the default value of sun=0, mon=1, ..., sat=6
- printf(“%d”,sun);   // output is 0
- we can manually specify the value of enum elements: ***enum Weekday {sun=7, mon=1, tue, wed, thu, fri, sat};*** what are the value of last 5 elements?
- since each element has a integer value, we can use it to compare with other values eg. ***if (workday==mon) or if(workday>tue)***

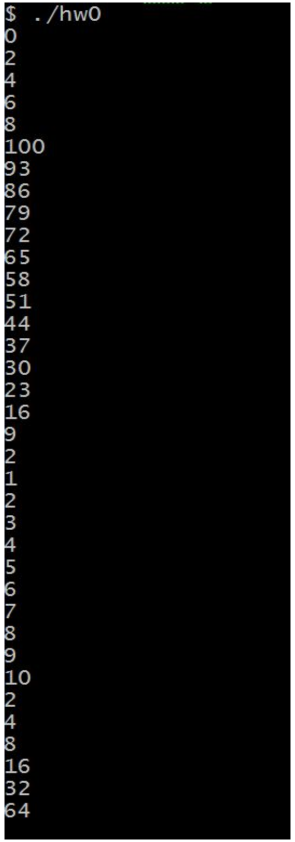
Homework 1 - Q1

What do these loops print? Determine what they print first, then run them

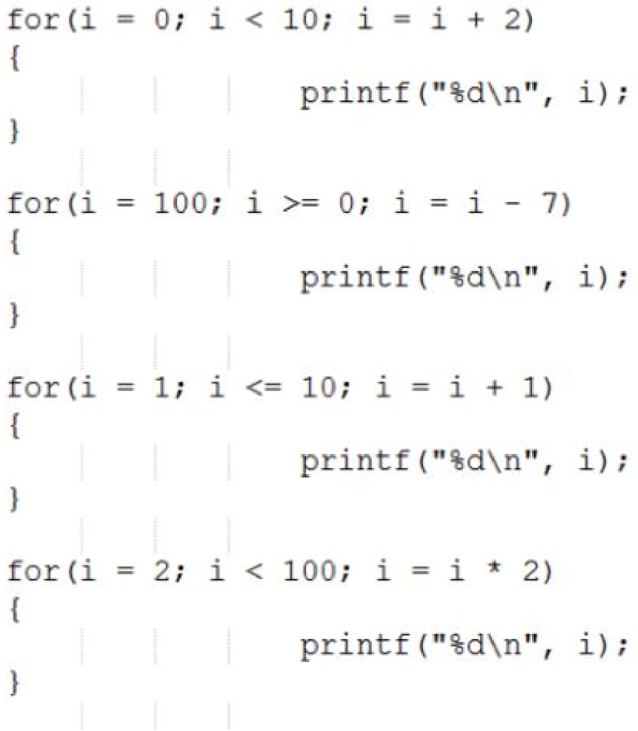


```
for(i = 0; i < 10; i = i + 2)
{
            printf("%d\n", i);
}

for(i = 100; i >= 0; i = i - 7)
{
            printf("%d\n", i);
}

for(i = 1; i <= 10; i = i + 1)
{
            printf("%d\n", i);
}

for(i = 2; i < 100; i = i * 2)
{
            printf("%d\n", i);
}
```

Homework 1 - Q1 Answer



```
$ ./hw0
0
2
4
6
8
100
93
86
79
72
65
58
51
44
37
30
23
16
9
2
1
2
3
4
5
6
7
8
9
10
2
4
8
16
32
64
```



```
for(i = 0; i < 10; i = i + 2)
{
            printf("%d\n", i);
}

for(i = 100; i >= 0; i = i - 7)
{
            printf("%d\n", i);
}

for(i = 1; i <= 10; i = i + 1)
{
            printf("%d\n", i);
}

for(i = 2; i < 100; i = i * 2)
{
            printf("%d\n", i);
}
```

-> 0, 2, ..., 6, 8

-> 100, 93, ... , 9, 2

-> 1, 2, ..., 10

-> 2, 4, 8, ... , 64

Homework 1 - Q2

Write a program to print this triangle:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

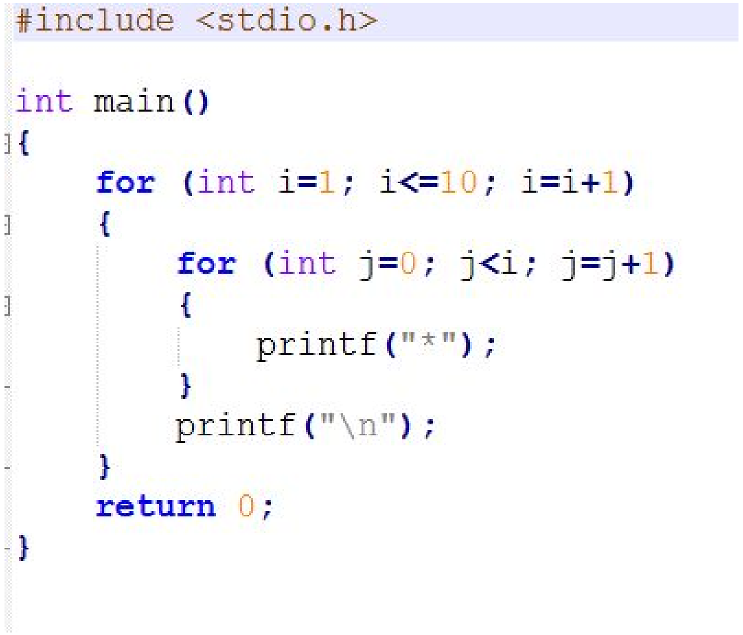
Don't use ten printf statements; use two nested loops instead. You'll have to use braces around the body of the outer loop if it contains multiple statements:

```
for(i = 1; i <= 10; i = i + 1)
 {
        /* multiple statements */
        /* can go in here */
  }
```

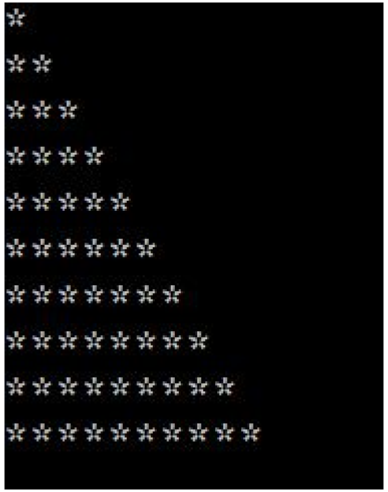
Change your loops to be while loops

Homework 1 - Q2 Answer

Using two for loop



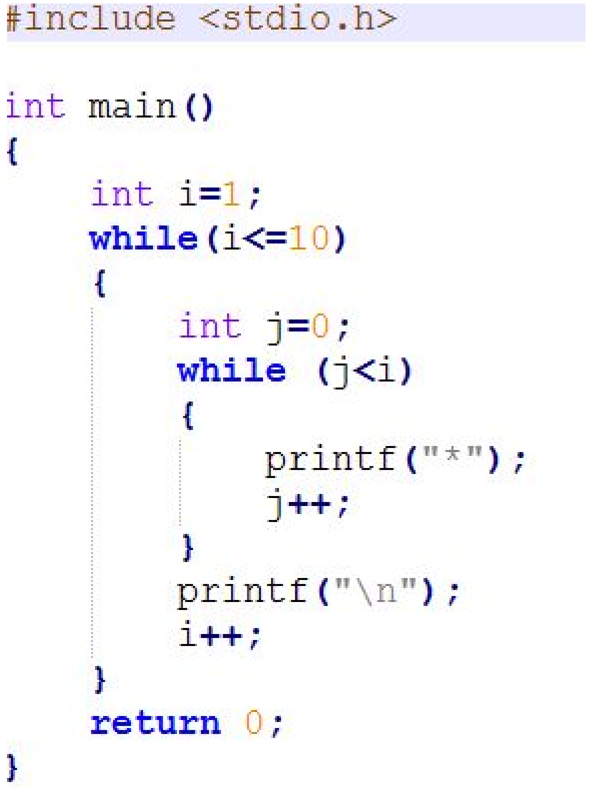
```c
#include <stdio.h>

int main()
{
    for (int i=1; i<=10; i=i+1)
    {
        for (int j=0; j<i; j=j+1)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```



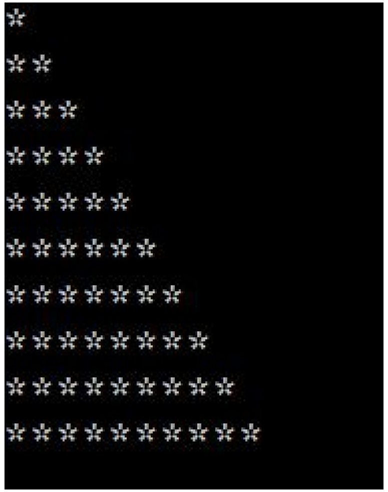
```
*
**
***
****
*****
******
*******
********
*********
**********
```

Homework 1 - Q2 Answer

Using two while loop



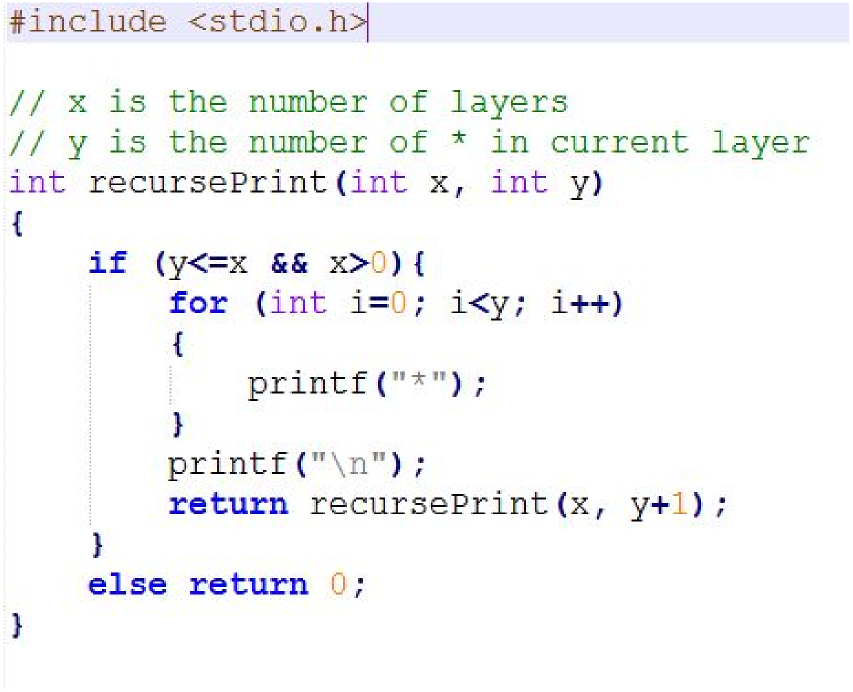
```
#include <stdio.h>

int main()
{
    int i=1;
    while(i<=10)
    {
        int j=0;
        while (j<i)
        {
            printf("*");
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```



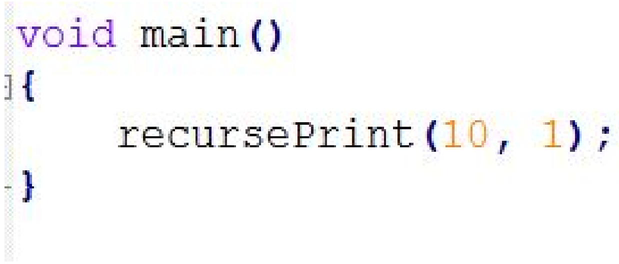
```
*
**
***
****
*****
******
*******
********
*********
**********
```

Homework 1 - Q2 Answer

Using one loop with recursion



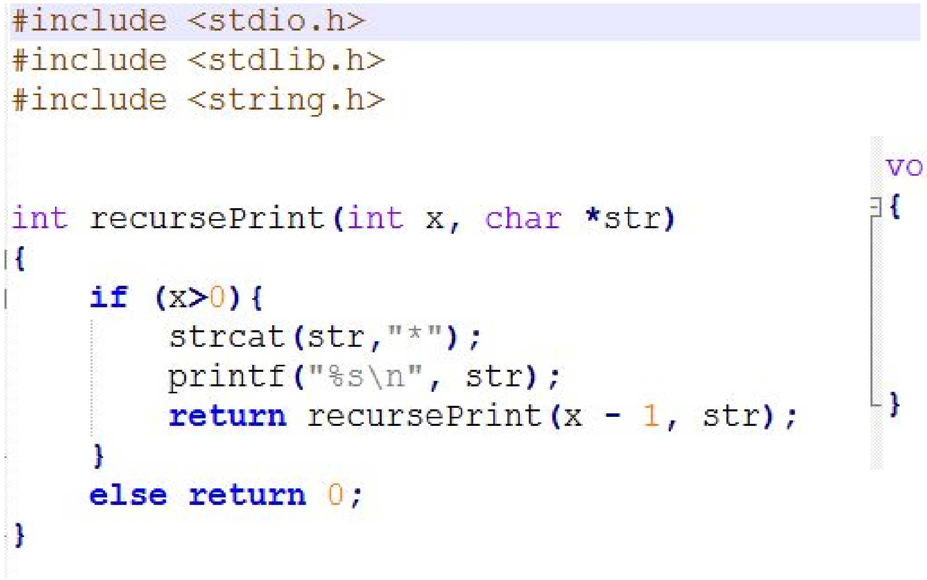
```
#include <stdio.h>

// x is the number of layers
// y is the number of * in current layer
int recursePrint(int x, int y)
{
    if (y<=x && x>0){
        for (int i=0; i<y; i++)
        {
            printf("*");
        }
        printf("\n");
        return recursePrint(x, y+1);
    }
    else return 0;
}
```



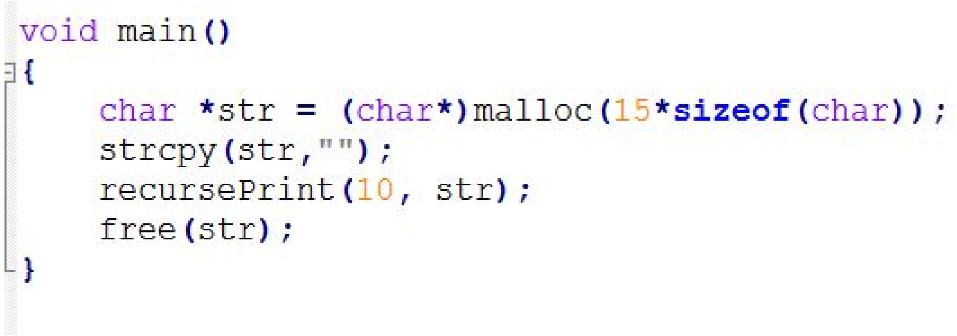
```
void main()
{
    recursePrint(10, 1);
}
```

Homework 1 - Q2 Answer

Not using loop with recursion



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


int recursePrint(int x, char *str)
{
    if (x>0){
        strcat(str,"*");
        printf("%s\n", str);
        return recursePrint(x - 1, str);
    }
    else return 0;
}
```



```c
void main()
{
    char *str = (char*)malloc(15*sizeof(char));
    strcpy(str,"");
    recursePrint(10, str);
    free(str);
}
```

Homework 2

HW2 assignment:

0. What's wrong with this #define line?

```
#define N 10;
```

1. Suppose you defined the macro

```
#define SIX 2*3
```

Then, suppose you used it in another expression:

```
int x = 12 / SIX;
```

What value would x be set to?

Homework 2 (Cont.)

2. Write your own version of atoi
   Take a char, inspect its int value and return its corresponding int value

e.g.

```
int test = my_atoi('5');
if( test == 5 )
{
        return 0;
}
else
{
        return -1;
}
```

Next, take a string of any length, scan its chars until you hit the '\0' and return the entire string's int value

e.g.

```
int test = my_atoi("512");
if( test == 512 )
{
        return 0;
}
else
{
        return -1;
}
```