CS 214: System Programming (Sec 07)

Shijie Geng
09/12/2017
Email: sg1309@rutgers.edu
Office Hours: Monday 10.30-11.30am at Hill 408

Context of today’s recitation (55min)

- Go over some knowledge points of C language (20min)
- Solving some example programming questions (20min)
- Do the homework for this week (15min)

Go over some knowledge points

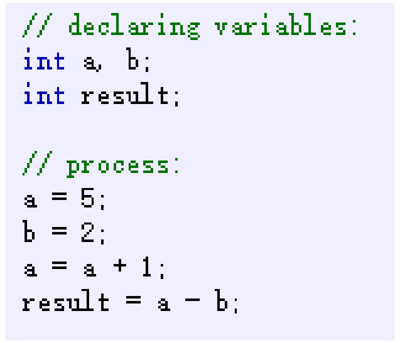
- Basic C syntax
- Header files
- Loops/conditionals
- GCC compilation

Basic C syntax

- Variables
- Constant
- Preprocessor definitions
- Operators
- Basic I/O command

1 Variables

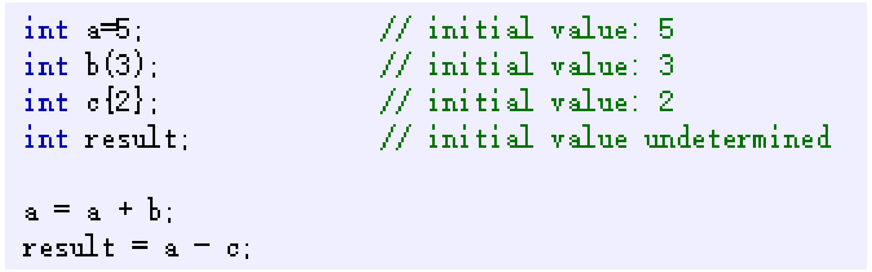
- Declaration of variables



```
// declaring variables:
int a, b;
int result;

// process:
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

- Initialization of variables

  type identifier = initial_value;

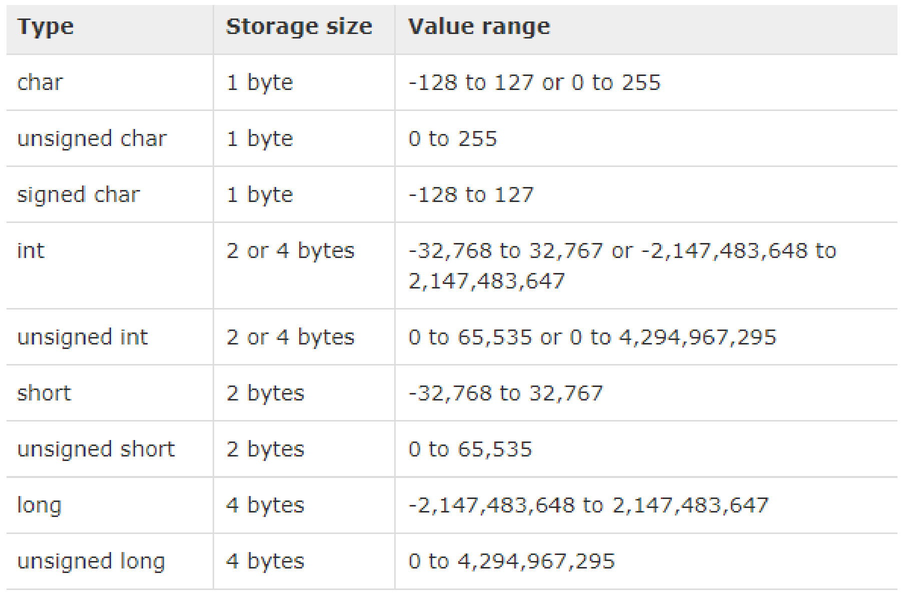


```
int a=5;               // initial value: 5
int b(3);              // initial value: 3
int c{2};              // initial value: 2
int result;            // initial value undetermined

a = a + b;
result = a - c;
```

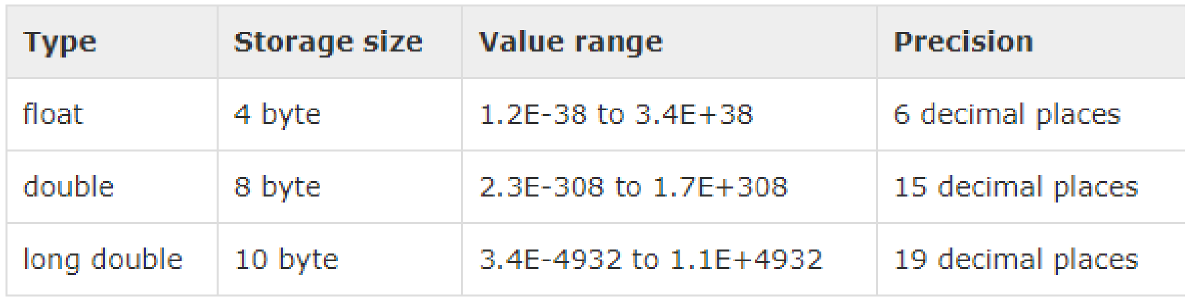
- A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).

1 Variables

- Types: basic type - integer and floating-point



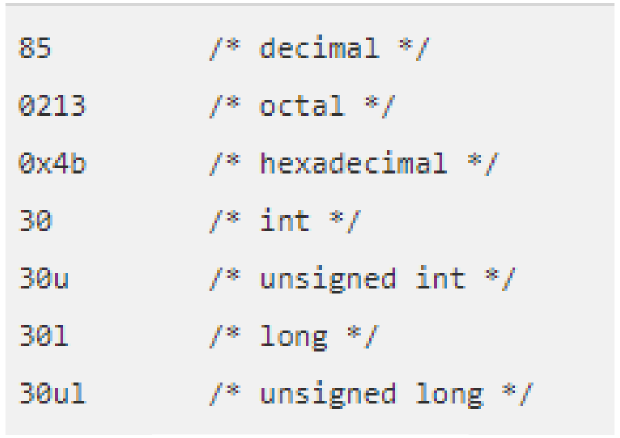
Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295



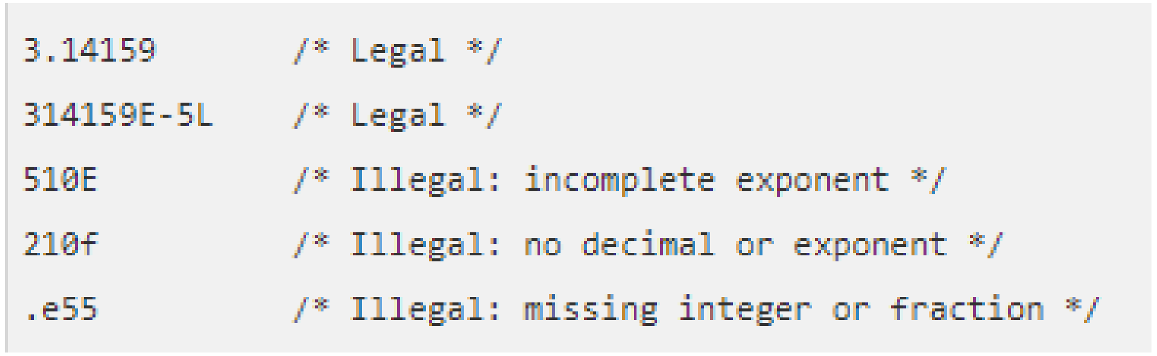
Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

2 Constant

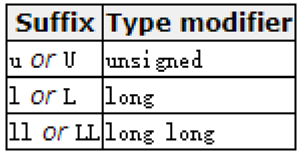
- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.



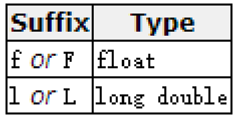
```
85          /* decimal */
0213        /* octal */
0x4b        /* hexadecimal */
30          /* int */
30u         /* unsigned int */
30l         /* long */
30ul        /* unsigned long */
```



```
3.14159       /* Legal */
314159E-5L    /* Legal */
510E          /* Illegal: incomplete exponent */
210f          /* Illegal: no decimal or exponent */
.e55          /* Illegal: missing integer or fraction */
```



Suffix	Type modifier
u *or* U	unsigned
l *or* L	long
ll *or* LL	long long

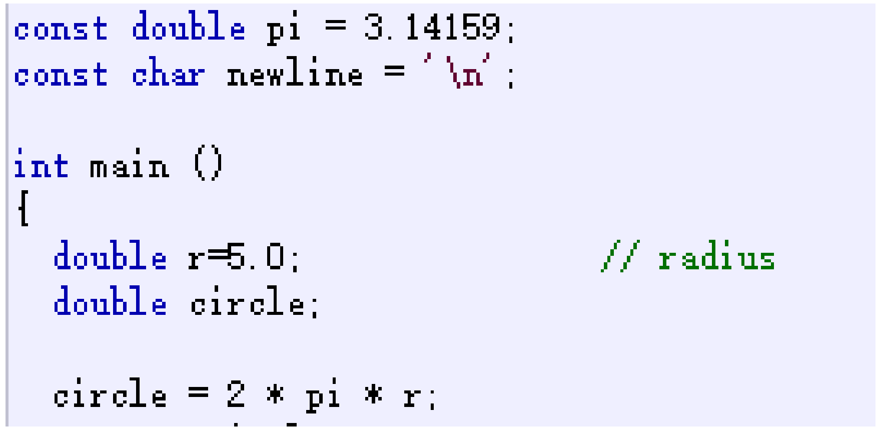


Suffix	Type
f *or* F	float
l *or* L	long double

2 Constant

- Typed constant expressions

  printf("circle is: %c%f",newline,circle);

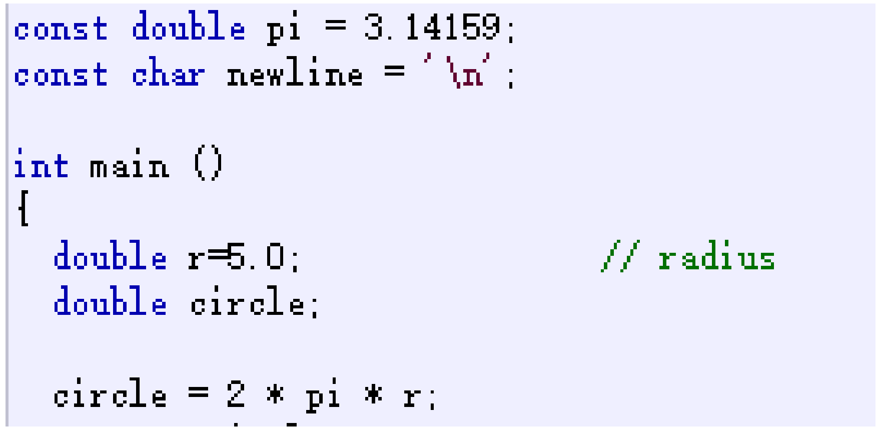


```
const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;                 // radius
  double circle;

  circle = 2 * pi * r;
```

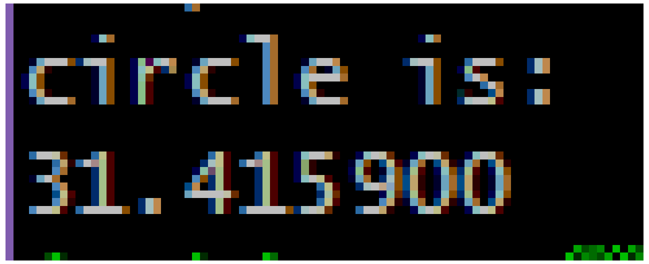
2 Constant

- Typed constant expressions

  printf("circle is: %c%f",newline,circle);



```
const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;                  // radius
  double circle;

  circle = 2 * pi * r;
```

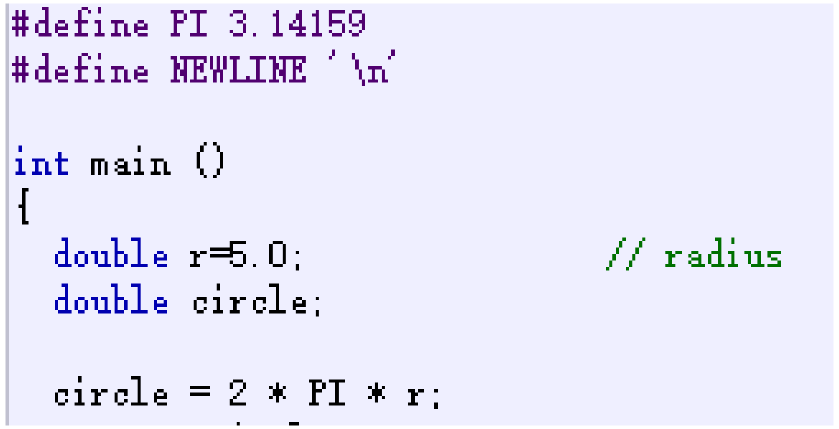


```
circle is:
31.415900
```

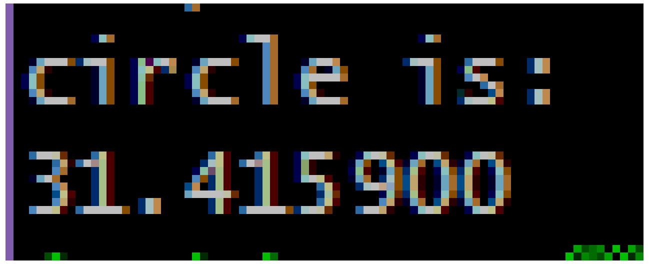
3 Preprocessor definitions

- Preprocessor definitions – don’t need to specify the type

printf("circle is: %c%f",newline,circle);



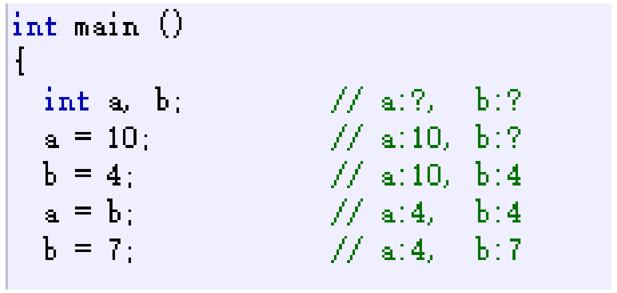
```
#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;               // radius
  double circle;

  circle = 2 * PI * r;
```



```
circle is:
31.415900
```

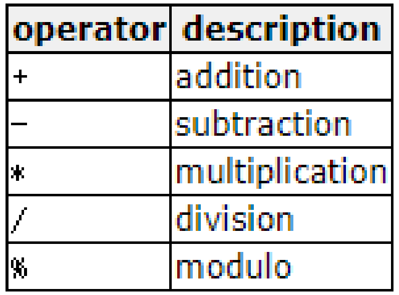
4 Operators

- Assignment Operator



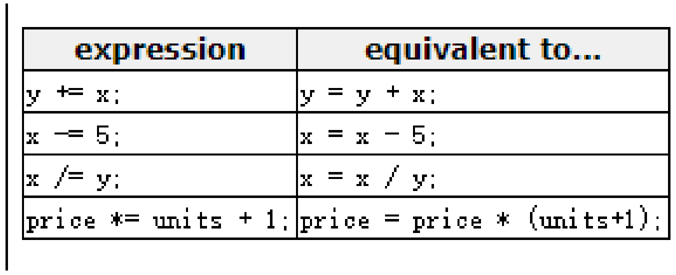
```
int main ()
{
  int a, b;          // a:?,   b:?
  a = 10;            // a:10,  b:?
  b = 4;             // a:10,  b:4
  a = b;             // a:4,   b:4
  b = 7;             // a:4,   b:7
```

- Arithmetic Operator



operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

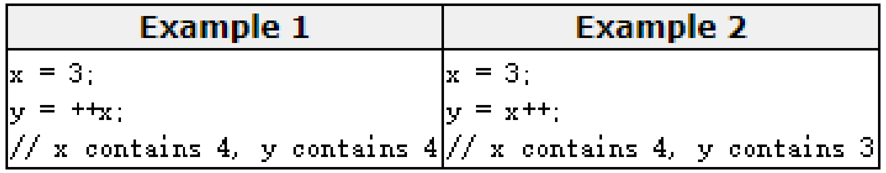
- Compound Assignment



expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

4 Operators

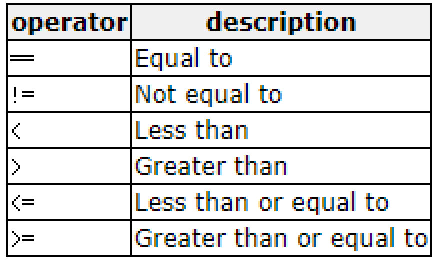
- Increment and Decrement



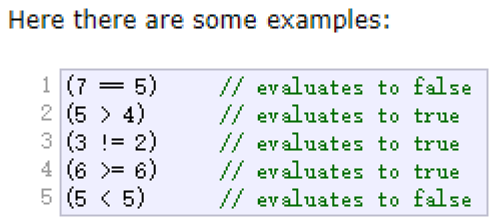
Example 1	Example 2

`x = 3;`
`y = ++x;`
`// x contains 4, y contains 4`
	`x = 3;`
	`y = x++;`
	`// x contains 4, y contains 3`

- Relational and Comparison Operator



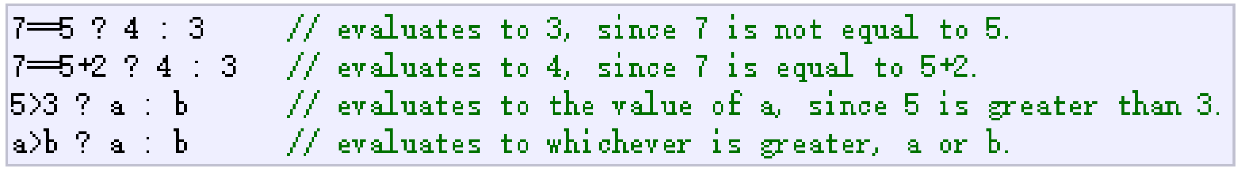
operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to



Here there are some examples:

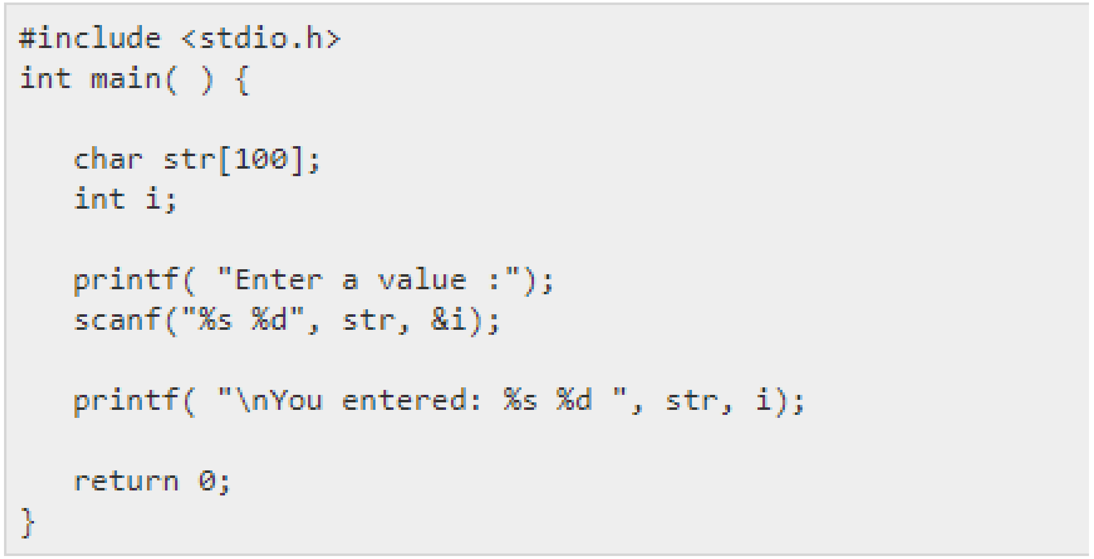
```
(7 == 5)     // evaluates to false
(5 > 4)      // evaluates to true
(3 != 2)     // evaluates to true
(6 >= 6)     // evaluates to true
(5 < 5)      // evaluates to false
```

- Conditional Ternary Operator



```
7==5 ? 4 : 3     // evaluates to 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3   // evaluates to 4, since 7 is equal to 5+2.
5>3 ? a : b      // evaluates to the value of a, since 5 is greater than 3.
a>b ? a : b      // evaluates to whichever is greater, a or b.
```

5 Basic I/O command



```
#include <stdio.h>
int main( ) {

   char str[100];
   int i;

   printf( "Enter a value :");
   scanf("%s %d", str, &i);

   printf( "\nYou entered: %s %d ", str, i);

   return 0;
}
```

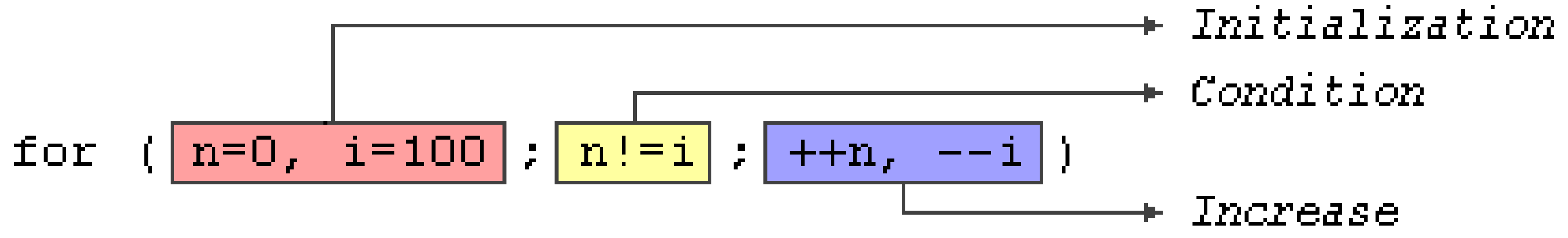
The **format** can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively

Header files

- #include <stdio.h>
- #include <xx.h> & #include “xx.h”

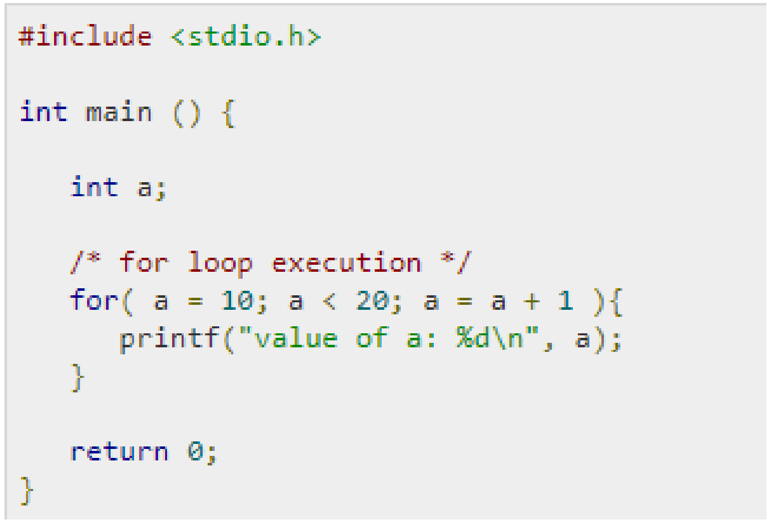
Loops/conditionals

- for loop

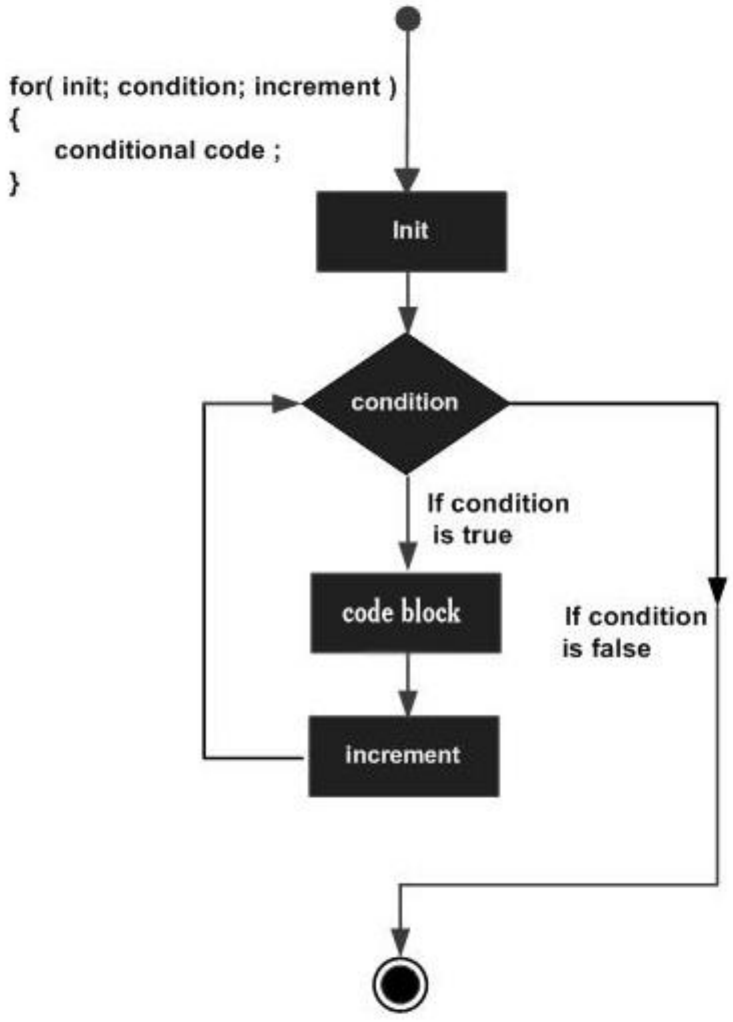

```
for ( n=0, i=100 ; n!=i ; ++n, --i )
```

Loops/conditionals

- for loop

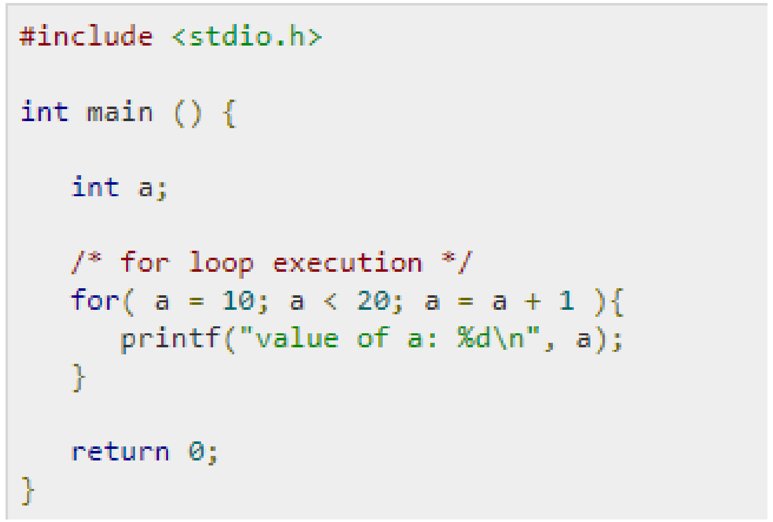


```c
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```

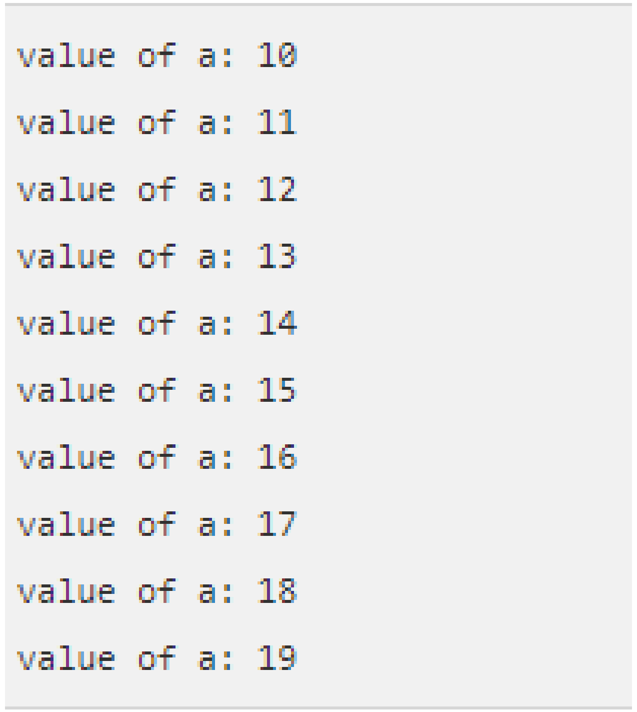


Loops/conditionals

- for loop



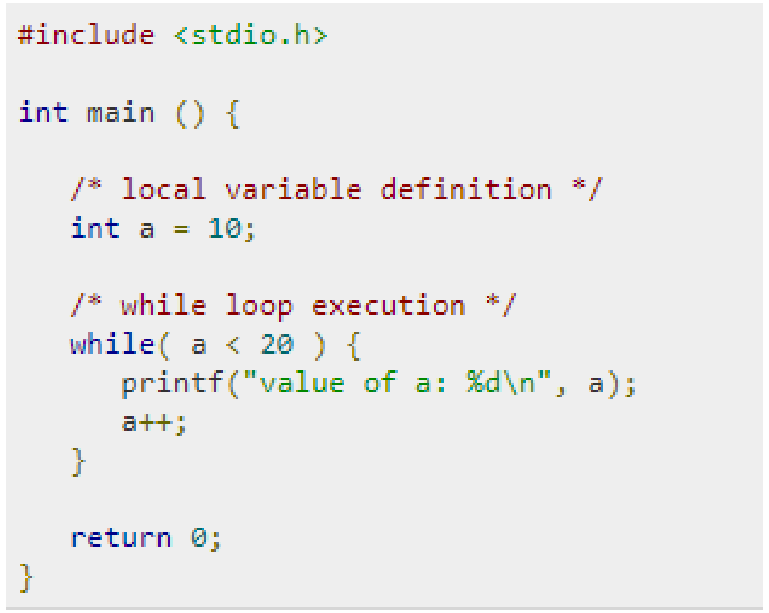
```
#include <stdio.h>

int main () {

   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }

   return 0;
}
```



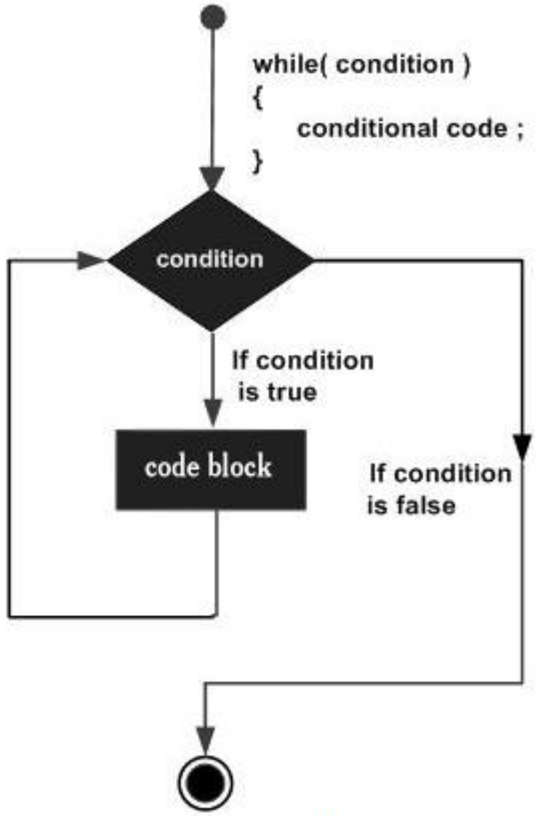
```
value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19
```

Loops/conditionals

- while loop

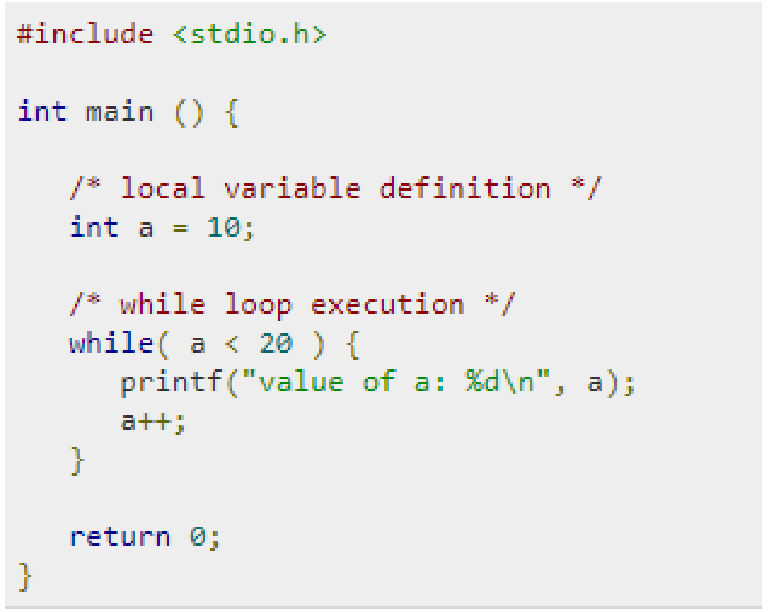


```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```

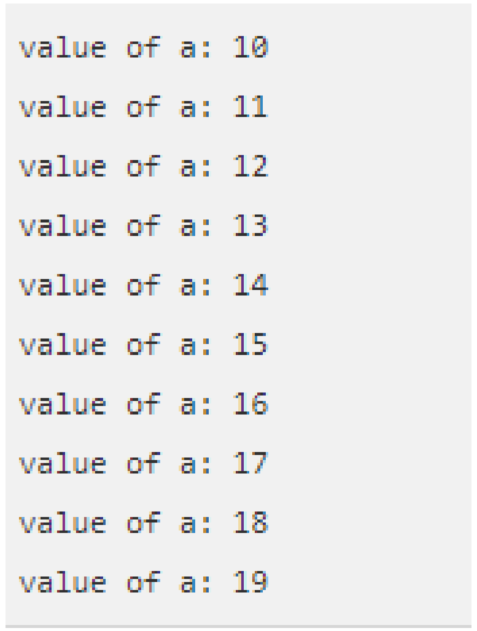


Loops/conditionals

- while loop



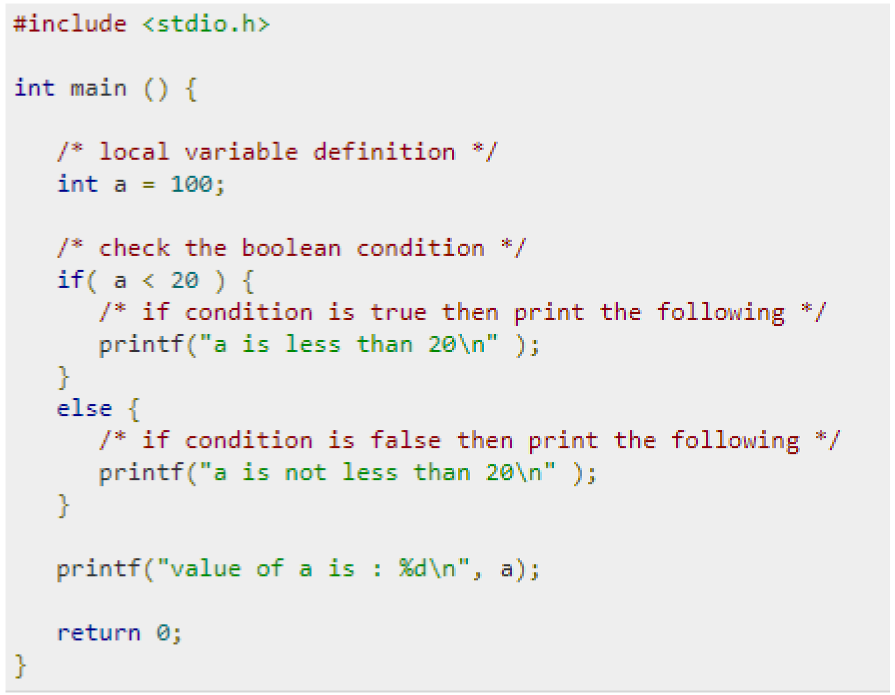
```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {
      printf("value of a: %d\n", a);
      a++;
   }

   return 0;
}
```



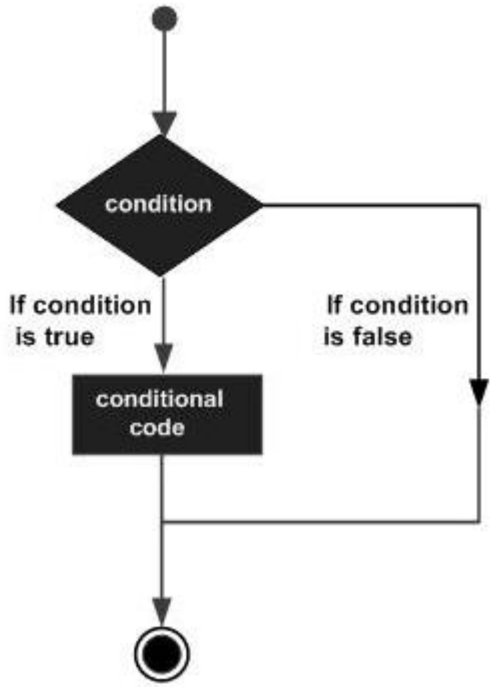
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Loops/conditionals

- if and else condition

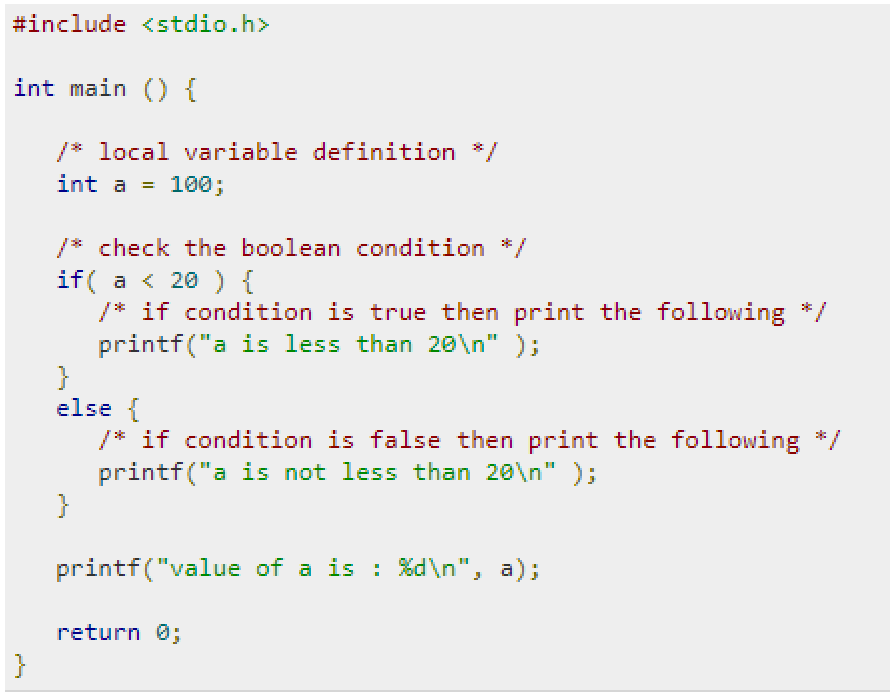


```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }
   else {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```

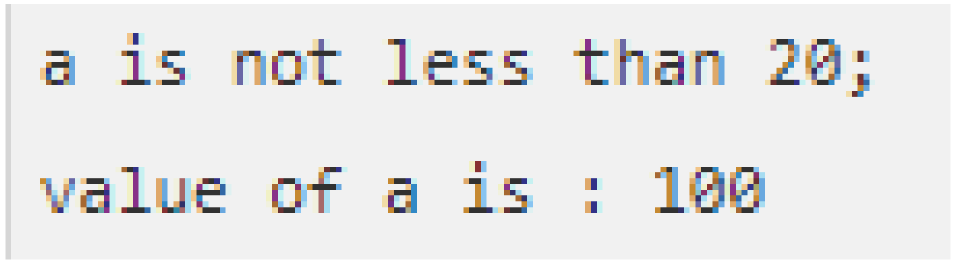


Loops/conditionals

- if and else condition



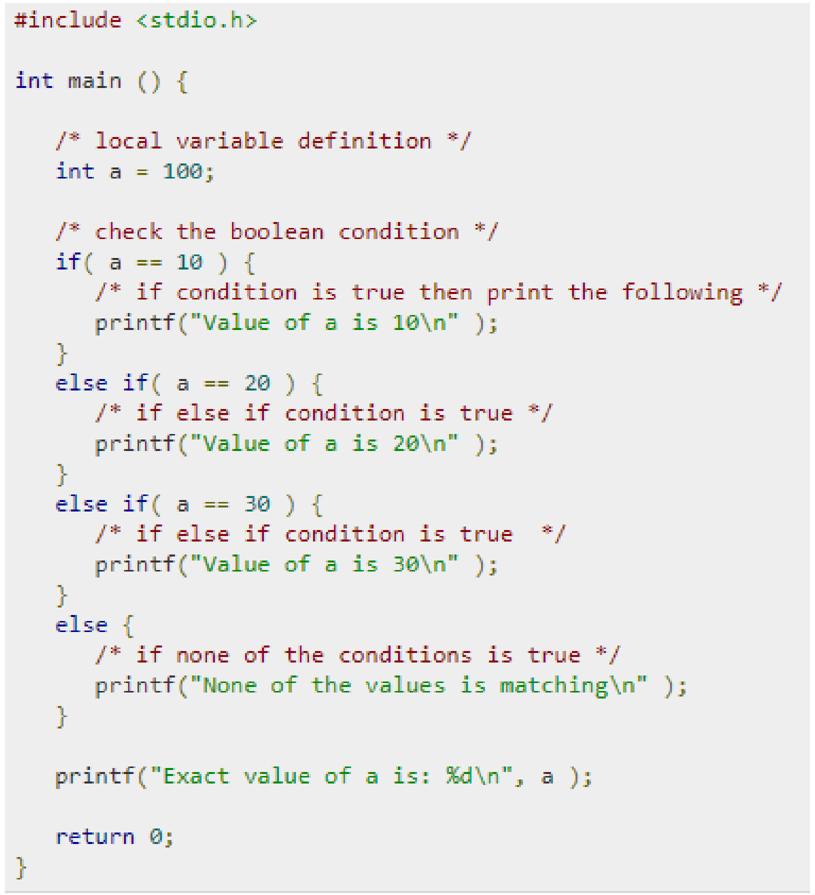
```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }
   else {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```



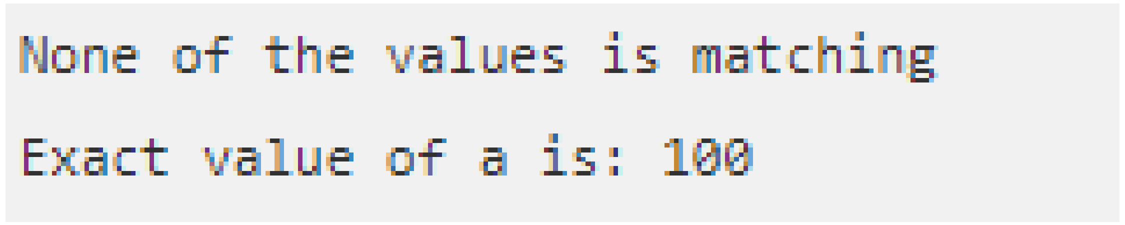
```
a is not less than 20;
value of a is : 100
```

Loops/conditionals

- if and else condition – more elses



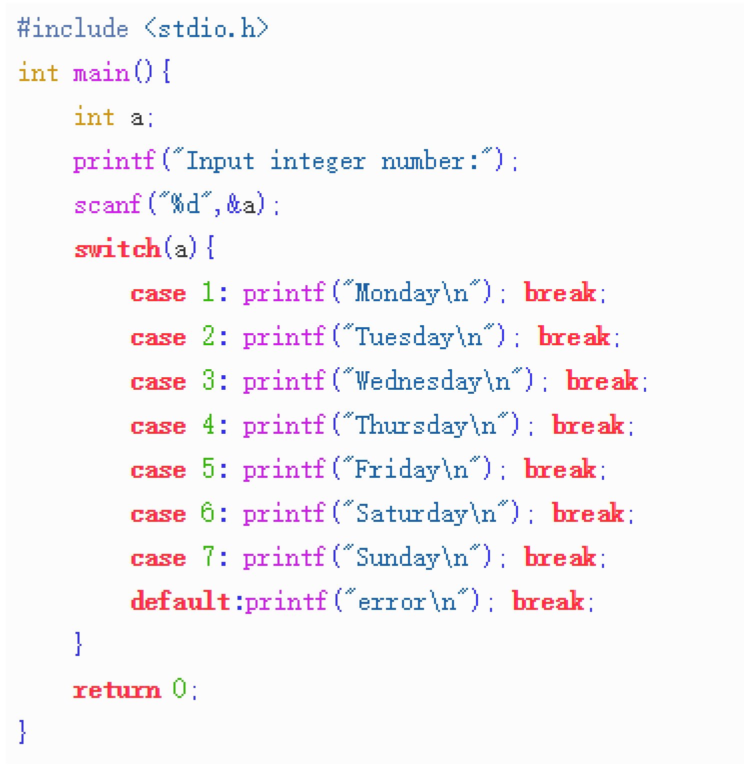
```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a == 10 ) {
      /* if condition is true then print the following */
      printf("Value of a is 10\n" );
   }
   else if( a == 20 ) {
      /* if else if condition is true */
      printf("Value of a is 20\n" );
   }
   else if( a == 30 ) {
      /* if else if condition is true  */
      printf("Value of a is 30\n" );
   }
   else {
      /* if none of the conditions is true */
      printf("None of the values is matching\n" );
   }

   printf("Exact value of a is: %d\n", a );

   return 0;
}
```



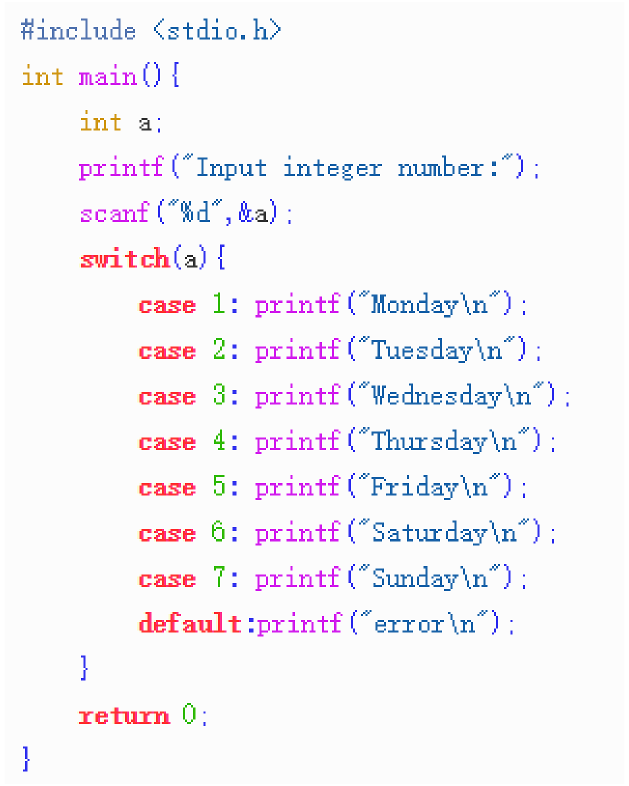
```
None of the values is matching

Exact value of a is: 100
```

Loops/conditionals

- switch condition



```c
#include <stdio.h>
int main(){
    int a;
    printf("Input integer number:");
    scanf("%d",&a);
    switch(a){
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3: printf("Wednesday\n"); break;
        case 4: printf("Thursday\n"); break;
        case 5: printf("Friday\n"); break;
        case 6: printf("Saturday\n"); break;
        case 7: printf("Sunday\n"); break;
        default:printf("error\n"); break;
    }
    return 0;
}
```



```c
#include <stdio.h>
int main(){
    int a;
    printf("Input integer number:");
    scanf("%d",&a);
    switch(a){
        case 1: printf("Monday\n");
        case 2: printf("Tuesday\n");
        case 3: printf("Wednesday\n");
        case 4: printf("Thursday\n");
        case 5: printf("Friday\n");
        case 6: printf("Saturday\n");
        case 7: printf("Sunday\n");
        default:printf("error\n");
    }
    return 0;
}
```

GCC compilation

- 4 steps: preprocessing – compilation – assembly – linking

preprocessing : from xx.c to xx.i -> gcc –E xx.c –o xx.i

gcc –E xx.c (on screen)

compilation : from xx.i to xx.s -> gcc –S xx.i –o xx.s

gcc –S xx.i (the same)

GCC compilation

- 4 steps: preprocessing – compilation – assembly – linking

```
assembly : from xx.s to xx.o  ->  gcc –c xx.s –o xx.o
                                  gcc –c xx.s (the same)
linking : from xx.o to xx.exe or xx  ->  gcc xx.o –o xx (linux)
                                         gcc xx.o –o  xx.exe  (windows)
```

GCC compilation

- 4 steps in one command: gcc –o xx.exe xx.c (in windows)
  gcc –o xx xx.c (in linux)

- Execute the binary file : use *./xx.exe* or *./xx*

Example Question 1

- Print all the narcissistic number
- A *narcissistic number* is 3-digit number which equals to the sum of the cube of its each digit.
- For example, 153 is a narcissistic number, because 153=1^3+5^3+3^3
- Hint: Use loops/conditionals to solve this question

Example Question 2

- Calculate the value of 1!+2!+3!+…+20! (! represents factorial)
- Hint: Use double loops to solve this question

Example Question 3

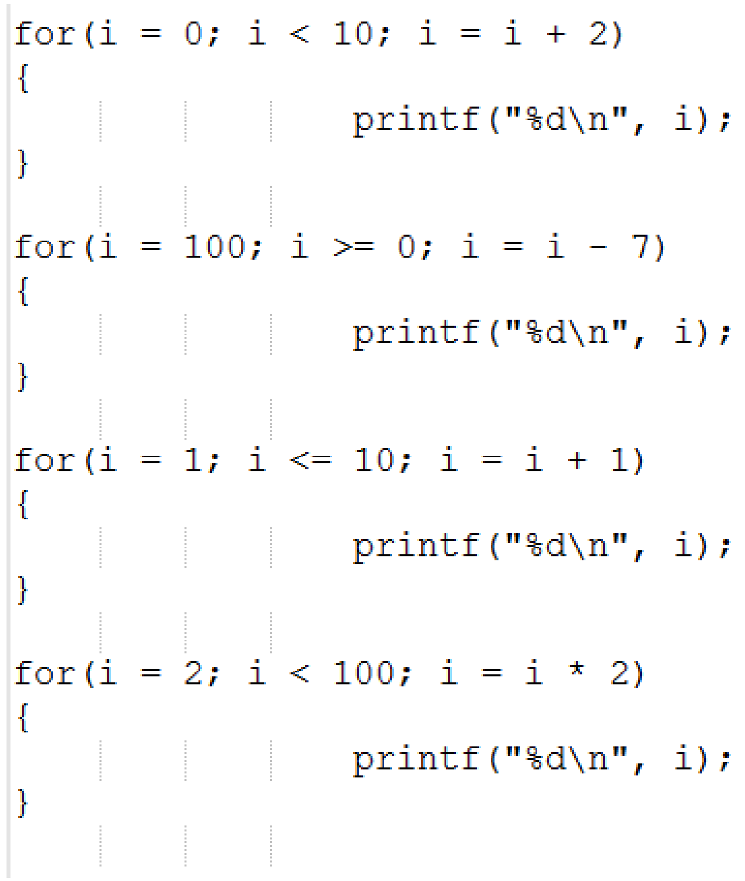
- Use gcc to compile the source codes to the two questions
- Execute the binary code and show the answers

Homework Instructions

- There are two questions for this week
- The homework will not be collected, but you can solve it and practice yourself

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 - 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