for Statement

- The “for” statement is the most commonly used looping structure in C.
- General syntax:

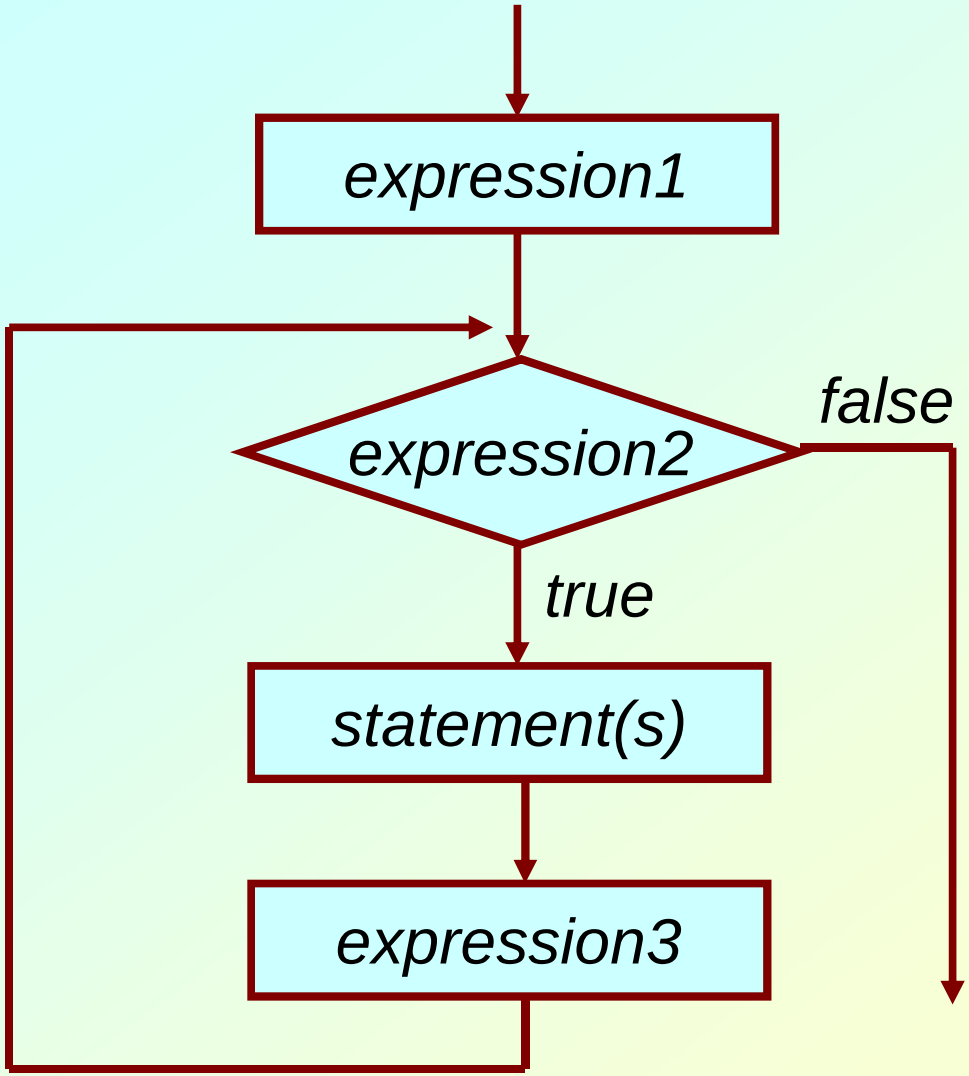
```
for (expression1; expression2; expression3)
  statement-to-repeat;
```

```
for (expression1; expression2; expression3)
{
  statement_1;
    :
  statement_N;
}
```

- **How it works?**
  - "expression1" is used to ***initialize*** some variable (called ***index***) that controls the looping action.
  - "expression2" represents a ***condition*** that must be true for the loop to continue.
  - "expression3" is used to ***alter*** the value of the ***index*** initially assigned by "expression1".

```
int  digit;
for (digit=0; digit<=9;digit++)
  printf ("%d \n", digit);
```

```
int  digit;
for (digit=9;digit>=0;digit--)
   printf ("%d \n", digit);
```



Single-entry / single-exit structure

for :: Examples

```
int  fact = 1, i;

for  (i=1; i<=10; i++)
   fact = fact * i;
printf (“%d \n”, fact);
```

```
int  sum = 0, N, count;

scanf (“%d”, &N);

for (i=1; i<=N, i++)
   sum = sum + i * i;

printf (“%d \n”, sum);
```

- **The comma operator**
  - **We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”.**

```
for  (fact=1, i=1; i<=10; i++)
  fact = fact * i;
```

```
for (sum=0, i=1; i<=N, i++)
  sum = sum + i * i;
```

for :: Some Observations

- **Arithmetic expressions**
  - **Initialization, loop-continuation, and increment can contain arithmetic expressions.**

```
for (k = x;  k <= 4 * x * y;  k += y / x)
```

- **"Increment" may be negative (decrement)**

```
for  (digit=9; digit>=0; digit--)
```

- **If loop continuation condition initially *false*:**
  - **Body of *for* structure not performed.**
  - **Control proceeds with statement after *for* structure.**

A common mistake

```
int  fact = 1, i;

for  (i=1; i<=10; i++)
   fact = fact * i;
printf ("%d \n", fact);
```

```
int  fact = 1, i;

for  (i=1; i<=10; i++);
   fact = fact * i;
printf ("%d \n", fact);
```

Loop body will execute only once!

Specifying “Infinite Loop”

```
while  (1)  {
   statements
}
```

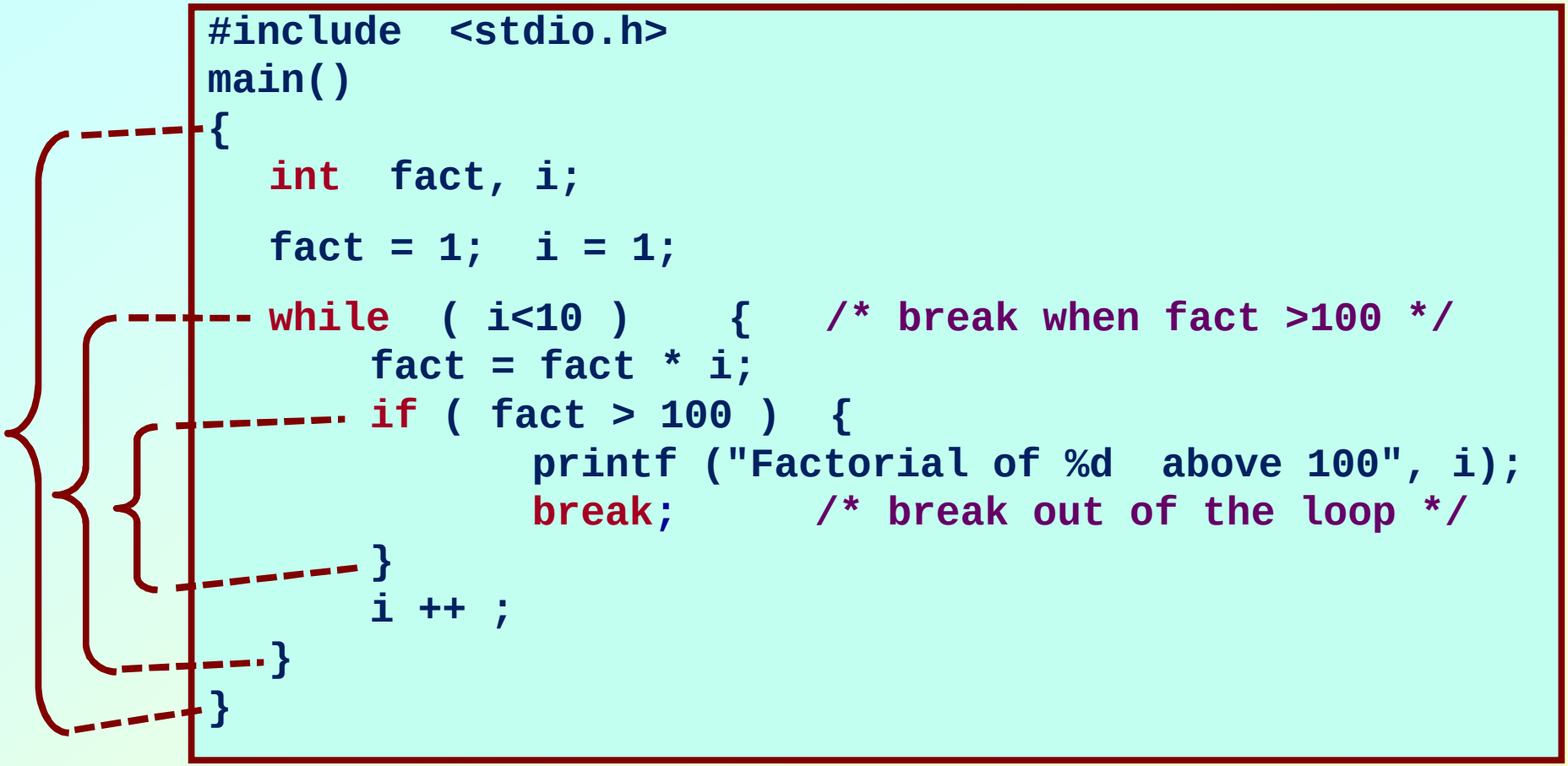
```
for  (; ;)
{
   statements
}
```

```
do  {
   statements
}  while (1);
```

The "break" Statement Revisited

- Break out of the loop { }
  - can use with
    - while
    - do while
    - for
    - switch
  - does not work with
    - if
    - else
- Causes immediate exit from a *while*, *do/while*, *for* or *switch* structure.
- Program execution continues with the first statement after the structure.

An example with "*break*"



```
#include  <stdio.h>
main()
{
   int  fact, i;

   fact = 1;  i = 1;

   while  ( i<10 )    {   /* break when fact >100 */
        fact = fact * i;
        if ( fact > 100 )  {
              printf ("Factorial of %d  above 100", i);
              break;      /* break out of the loop */
        }
        i ++ ;
   }
}
```

The "continue" Statement

- **Skips the remaining statements in the body of a *while*, *for* or *do/while* structure.**
  - **Proceeds with the next iteration of the loop.**
- **while and do/while**
  - **Loop-continuation test is evaluated immediately after the continue statement is executed.**
- **for structure**
  - ***expression3* is evaluated, then *expression2* is evaluated.**

An example with "*break*" and "*continue*"

```
fact = 1; i = 1;     /* a program to calculate 10! */
while  (1)  {
   fact = fact * i;
   i ++;
   if (i<10)
        continue;     /* not done yet ! Go to loop and
                         perform next iteration*/
   break;
}
```

Some Examples

Example 1: Test if a number is prime or not

```
#include <stdio.h>
main()
{
   int  n, i=2;
   scanf (”%d”, &n);
   while (i < n)  {
       if (n % i == 0)  {
              printf (”%d is not a prime \n”, n);
              exit;
       }
       i++;
   }
   printf (”%d is a prime \n”, n);
}
```

More efficient??

```
#include <stdio.h>
main()
{
   int  n, i=3;
   scanf (”%d”, &n);
   while (i < sqrt(n))  {
       if (n % i == 0)  {
              printf (”%d is not a prime \n”, n);
              exit;
       }
       i = i + 2;
   }
   printf (”%d is a prime \n”, n);
}
```

Example 2: Find the sum of digits of a number

```
#include  <stdio.h>
main()
{
   int n, sum=0;
   scanf (”%d”, &n);
   while (n != 0)  {
        sum = sum + (n % 10);
        n = n / 10;
   }
   printf (”The sum of digits of the number is %d \n”, sum);
}
```

Example 3: Decimal to binary conversion

```
#include  <stdio.h>
main()
{
   int  dec;
   scanf (”%d”, &dec);
   do
   {
       printf (”%2d”, (dec % 2));
       dec = dec / 2;
   }  while (dec != 0);
   printf (”\n”);
}
```

Example 4: Compute GCD of two numbers

```
#include  <stdio.h>
main()
{
   int  A, B, temp;
   scanf (”%d %d”, &A, &B);
   if  (A > B)
     {temp = A;  A = B;  B = temp;}
   while ((B % A) != 0)  {
        temp = B % A;
        B = A;
        A = temp;
   }
   printf (”The GCD is %d”, A);
}
```

```
12 ) 45 ( 3
     36
     --
      9 ) 12 ( 1
           9
           -
           3 ) 9 ( 3
                9
                -
                0
```

Initial: A=12, B=45
Iteration 1: temp=9, B=12,A=9
Iteration 2: temp=3, B=9, A=3

B % A = 0 ✍ GCD is 3

Shortcuts in Assignments

- **Additional assignment operators:**

  + =, – =, * =, / =, % =

a += b is equivalent to **a = a + b**

a *= (b+10) is equivalent to **a = a * (b + 10)**

and so on.

More about scanf and printf

Entering input data :: scanf function

- **General syntax:**

  scanf (control string, arg1, arg2, …, argn);

  – **"control string refers to a string typically containing data types of the arguments to be read in;**

  – **the arguments arg1, arg2, … represent pointers to data items in memory.**

  Example: scanf (%d %f %c", &a, &average, &type);

- **The control string consists of individual groups of characters, with one character group for each input data item.**

  – **'%' sign, followed by a conversion character.**

- **Commonly used conversion characters:**

c	single character
d	decimal integer
f	floating-point number
s	string terminated by null character
X	hexadecimal integer

- **We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.**

  Example: scanf ("%3d %5d", &a, &b);

Writing output data :: printf function

- **General syntax:**

  printf (control string, arg1, arg2, …, argn);

  – **"control string refers to a string containing formatting information and data types of the arguments to be output;**

  – **the arguments arg1, arg2, … represent the individual output data items.**

- **The conversion characters are the same as in scanf.**

- **Examples:**

```
printf  (“The average of %d and %d is %f”, a, b, avg);
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- **Many more options are available:**
  - Read from the book.
  - Practice them in the lab.
- **String I/O:**
  - Will be covered later in the class.