CS1100
Introduction to Programming

While; For and Do-While Loops

Repetitive Statements

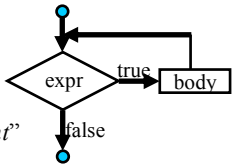
- A very important type of statement
  - iterating or repeating a set of operations
  - a very common requirement in algorithms
- C offers three iterative constructs
  - the ***while*** … construct
  - the ***for*** construct
  - the ***do … while*** construct

The *while* Construct

- General form:

  while (<*expr*>) <*statement*>

- Semantics:
  - repeat: Evaluate the "*expr*"
    If the "*expr*" is true
    execute the "*statement*"
    else
    exit the loop



- "expr" must be modified in the loop or we have an infinite loop!

Computing 2^n, n>=0, using *while* Construct

- Syntax – *while* (condition){ statement}

```
#include<stdio.h>
main( )
{
  int n, counter, value;
  printf ("Enter value for n:");
  scanf ("%d", &n);
  value = 1;
  printf ("current value is %d \n", value);
```

Contd…

```
   counter = 0;  //Initialization
   while (counter <= n) // Termination condition
    {
     value = 2 * value;
     printf ("current value is %d \n", value);
     counter = counter + 1; // Update of variable
    }
}
```

Exercise: try this program and identify problems

Testing the Program

- Choose test cases:
  - A few normal values: n = 2, 5, 8, 11
  - Boundary values: n = 0, 1
  - Invalid values: n = –1
- Hand simulate the execution of the program
  - On paper, draw a box for each variable and fill in the initial values (if any)
  - Simulate exec. of the program one statement at a time
  - For any assignment, write the new value of the variable in the LHS
  - Check if the output is as expected in each test case

Hand Simulation

```
#include<stdio.h>
main( )
{
   int n, counter, value;
   printf ("Enter value for n:");
   scanf ("%d", &n);
   value = 1;
   printf ("current value is %d \n",
   value);
```

n	counter	value
4		1

Current value is 1

Contd…

```
   counter = 0;
   while (counter <= n)
    {
     value = 2 * value;
     printf ("current value is %d \n", value);
     counter = counter + 1;
    }
}
```

condition	n	counter	value
F	4	5	32

Current value is 1

Current value is 2
Current value is 4
Current value is 8
Current value is 16
Current value is 32

Additional Q

Write a while loop condition that will only accept values from 15 to 25 and keeps prompting till the user enters a value in this range.

```
int n;
printf("Enter a number between 15 and 25:");
scanf("%d", &n);
while ( (n < 15) || (n > 25)  ) {
   printf("Enter a number between 15 and 25:");
   scanf("%d", &n);
}
printf("%d", n);
```


do while

```
int n;
do {
   printf("Enter a number between 15 and 25:");
   scanf("%d", &n);
} while ( (n < 15) || (n > 25)  );
printf("%d", n);
```


do while

Write a while loop condition that print the sum of all numbers input, until the number (sentinel) -9999 is input.

```
int n,sum=0;
do {
   printf("Enter a number ");
   scanf("%d", &n);
  if (n != -9999) sum=sum+n;
} while ( n!=-9999  );
printf("%d", sum);
```

```
int n,sum=0;
 printf("Enter a number ");
 scanf("%d", &n);

While ( n!=-9999)
{
sum=sum+n;
   printf("Enter a number ");
   scanf("%d", &n);
}
printf("%d", sum);
```


More on Loops

- Loop execution can be typically seens as being controlled in one of the two ways: counter-controlled and sentinel-controlled.
- ***Counter*** – loop runs till counter reaches its limit.
  - Use it when the number of repetitions is known.
- ***Sentinel*** – loop runs till a certain condition is encountered.
  - For example – a \n (newline) is read from the input.
  - Use it when the number of repetitions is a property of the input and not of the problem being solved.

Reversing a Number: Methodology

- Print the reverse of a given integer:
- E.g.: 234 → 432
- Method: Till the number becomes zero,
  - extract the last digit
  - number modulo 10
  - make it the next digit of the result
  - multiply the current result by 10 and
  - add the new digit

Reversing a Number: Illustration

- x is the given number
- y is the number being computed
- $x = 5634$2 y = 0
- $x = 5634$ y = 0*10 + 2 = 2
- $x = 563$ y = 2*10 + 4 = 24
- $x = 56$ y = 24*10 + 3 = 243
- $x = 5$ y = 243*10 + 6 = 2436
- $x = 0$ y = 2436*10 + 5 = 24365

Termination condition: Stop when x becomes zero

$x = x/10$

$y = y*10 + (x\%10)$

Reversing a Number: Program

```
main( ){
  int x = 0, y = 0;
  printf ("input an integer :\n");
  scanf ("%d", &x);
  while (x > 0){
    y = y*10 + ( x % 10 );
    x = (x / 10);
  }
  printf ("The reversed number is %d \n", y);
}
```

Remember integer division truncates the quotient

Perfect Number Detection

- Perfect number – sum of proper divisors adds up to the number
- Pseudocode:
  - Read a number, A
  - Set the sum of divisors to 1
  - If A is divisible by 2, Add 2 to the sum of divisors
  - If A is divisible by 3, Add 3 to the sum of divisors

    …
  - If A is divisible by A/2, Add A/2 to the sum of divisors
  - If A is equal to the sum of divisors, A is a perfect number

Refining the Pseudocode

- Read a number, A
- Set the sum of divisors to 1
- Set B to 2
- While B is less than or equal to A/2
  - If A is divisible by B, Add B to the sum of divisors
  - Increment B by 1
- If A is equal to the sum of divisors, A is a perfect number

Perfect Number Detection

```
main ( ){
   int d=2, n, sum=1;
   scanf ("%d", &n);
   while (d <= (n/2)) {
      if (n %d  == 0)
           sum += d;
      d++;
     }
   if (sum == n) printf ("%d is perfect\n", n);
   else printf ("%d is not perfect\n", n);
}
```

d<n will also do, but would do unnecessary work

Exercise: Modify to find the first *n* perfect numbers

for loops

- Counter controlled repetitions needs
  - Initial value for the counter
  - Modification of counter: i = i+1or i= i–1, or any other arithmetic expression based on the problem, and
  - Final value for the counter
- ***for*** repetition structure provides for the programmer to specify all these
- Any statement written using ***for*** can be rewritten using ***while***
- Use of ***for*** helps make the program error free

The *for* construct

- General form:

  for (expr1; expr2; expr3) <statement>
- Semantics:
  - evaluate "expr1" - initialization operation(s)
  - repeat - evaluate expression "expr2" and
  - If "expr2" is true
    - execute "statement" and "expr3"
  - Else stop and exit the loop

Example Code with the *while* Construct

```
scanf("%d", &n);
value = 1;
printf ("current value is %d \n", value);
counter = 0;
while (counter <= n){
    value = 2 * value;
    printf ("current value is %d \n", value);
    counter = counter + 1;
}
```


Example Code with the *for* Construct

```
scanf("%d", &n);
value = 1;
for (count = 0; count <=n; count=count+1){
    if (count == 0) printf("value is %d \n",1);
    else{
        value = 2 * value;
        printf(value is %d \n", value);
    }
}
```

- Observe: a mistake in the earlier program is gone

Computing the Sum of the First 20 Odd Numbers

```
int i, j, sum; sum = 0;
for (j = 1, i = 1; i <= 20; i = i+1){
        sum += j;
        j += 2;
}
```

Set *j* to the first odd number

i : Loop control variable

Termination condition

Increment sum by the i^{th} odd number

Set j to the next odd number

Calculating Compound Interest

$a = p(1 + r)^n$

```
#include<stdio.h>
#include<math.h>
main( ){
int yr;
double amt, principal = 1000.0, rate = .05;
printf("%4s%10s\n", "year", "Amount");
for (yr = 1; yr < = 10; yr++) {
    amt = principal * pow(1.0 + rate, yr);
    printf("%4d%10.2f\n", yr, amt);
}
```

String constants used to align heading and output data in a table

The *do-while* construct

- ***for*** and ***while*** check termination condition before each iteration of the loop body
- Sometimes - execute the statement and check for condition
- General form:

  do {<statement>} ***while*** (expr);
- Semantics:
  - execute the statement and check expr
  - if expr is true, re-execute statement else exit

An Example

```
#include<stdio.h>
main( )
{
  int count = 1;
  do{
     printf("%d\n", count);
  } while(++count <= 10);
return 0;
}
```


Find the Square Root of a Number

- How do we find the square root of a given number N?
- We need to find the positive root of the polynomial $x^2 - N$
- Solve: $x^2 - N = 0$

Newton–Raphson Method

$f(x) = x^2 - N$

$$f'(x_n) = \frac{0 - f(x_n)}{(x_{n+1} - x_n)}$$

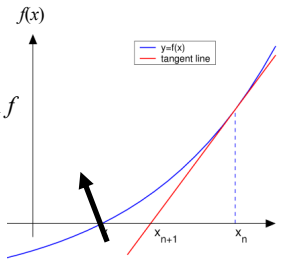
f' : the derivative of the function f

By simple algebra we can derive

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$x_{n+1} = x_n - (x_n^2 - N)/2x_n$

$= (x_n^2 + N)/2x_n = (x_n + N/x_n)/2$



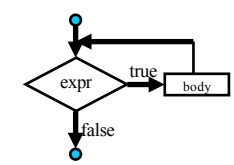
$\sqrt{N}$

http://en.wikipedia.org/wiki/Newton's_method

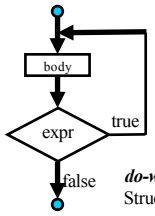
Square Root of a Number

```
int N;
double prevGuess, currGuess, error, sqRoot;
scanf("%d", &N);
currGuess = (float) N/2 ; error = 0.0001;
do{
        prevGuess = currGuess; // prevG = x_n
        currGuess = (prevGuess + N/prevGuess)/2;
    } while (fabs(prevGuess – currGuess) > error);
sqRoot = currGuess;
printf("%lf\n", sqRoot);
```


Repetition Structures

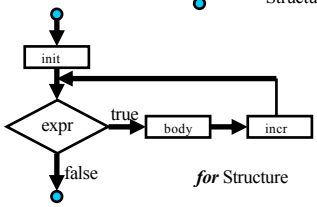


while Structure



do-while Structure

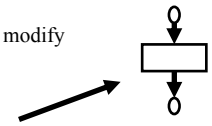
Single Entry Single Exit



for Structure

Structured Programming

- To produce programs that are
  - easier to develop, understand, test, modify
  - easier to get correctness proof
- Rules
  - Begin with the "simplest flowchart"
  - Any action box can be replaced by two action boxes in sequence
  - Any action box can be replaced by any elementary structures (sequence, ***if***, ***if/else***, ***switch***, ***while***, ***do-while*** or ***for***)
  - Rules 2 and 3 can be applied as many times as required and in any order



Break and Continue

- ***break*** – breaks out of the innermost loop or switch statement in which it occurs
- ***continue*** – starts the next iteration of the loop in which it occurs

An Example

```
#include<stdio.h>
// Prints 1 2 3 4 and exits for loop
main ( ){
        int i;
        for (i = 1; i < 10; i = i+1){
            if( i == 5)
                break;
            printf("%4d", i);
        }
}
```


An Example

```
#include<stdio.h>
// Prints 1 2 3 4 6 7 8 9
main ( ){
        int i;
        for (i = 1; i < 10; i = i+1){
            if ( i == 5)
                continue;
            printf("%4d", i);
        }
}
```


Find the Smallest Positive Number

```
#include<stdio.h>
int main ( ){
    int n=0, smallNum =  10000;
    printf("Enter a non-negative number (0 to 9999): ");
    scanf("%d", &n);
    while (n >= 0){
        if (n < smallNum) smallNum = n;
        printf("Enter a non-negative number (0 to 9999): ");
        scanf("%d",&n);
    }
    printf("Smallest number is %d\n",smallNum);
}
```


Exercises

- Write a program that reads in the entries of a 3x3 matrix, and prints it out in the form of a matrix. The entries could be floating point too.
- Write a program that reads in orders of two matrices and decides whether two such matrices can be multiplied. Print out the decision.
- Write a program that reads in two matrices, and multiplies them. Your output should be the two matrices and the resulting product matrix.
- Compute sin (x), using Taylors expansion. Your answer should be correct up to 'k' places of decimal. Where 'k' is an input value.