CS1100
Computational Engineering

Control Structures

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

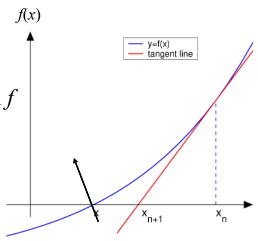
$f(x) = x^2 – N$

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

f' : the derivative of the function f

By simple algebra we can derive

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



$x_{n+1} = x_n – (x_n^{\,2} – N)/2x_n$

$= (x_n^{\,2} + N)/2x_n = (x_n + N/x_n)/2$

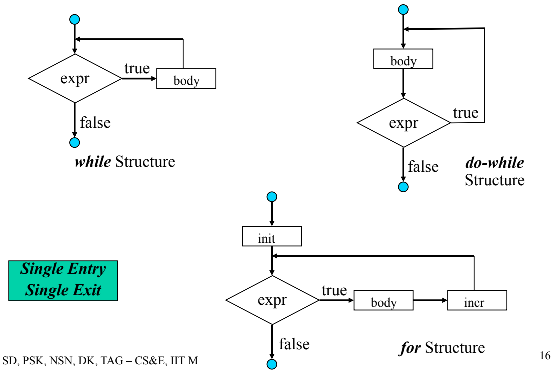
$\sqrt{N}$

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

Square Root of a Number

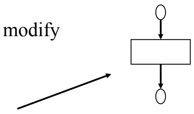
```
int n;
float prevGuess, currGuess, error, sqRoot;
scanf("%d", &n);
currGuess = (float) n/2 ; error = 0.0001;
do{
        prevGuess = currGuess;
        currGuess = (prevGuess + n/prevGuess)/2;
    }while(fabs(prevGuess – currGuess)>error);
sqRoot = currGuess;
printf("%f\n", sqRoot);
```

Repetition Structures



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>
main ( ){
        int i;
        for (i = 1; i < 10; i = i+1){
            if( i == 5)
                break; //continue;
            printf("%4d", i);
        }
}
```


Find a Smallest Positive Number

```
#include<stdio.h>
main ( ){
        int n=0, smallNum =  10000;
        printf("Enter Numbers (in the range 0 to 9999):\n");
        scanf("%d", &n);
        while (n >= 0){
            if(smallNum > n)   smallNum = n;
            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.