CS101 Lecture 5
Iteration

Timothy A. Gonsalves

Iterative 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 `for` construct

the `while` ... construct

the `do` ... `while` construct

Programming problems

- Write a program to check if a given number is prime. (can this be done without using a logical decision?)
- Write a program to count the number of digits in a given number. Your answer should contain two parts, number of digits before and after the decimal. (can you do this only with assignments to variables, and decisions?)

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!

Repetition Structure - *While*

Syntax – *while (condition){ statement}*

Print powers of 2 up to 2^N

```
#include<stdio.h>
#include<math.h>
main()
{
  int n, count, value;
  printf("Enter power n: ");
  scanf("%d", &n);
```

contd..

Program using *while*

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

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 execution 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

More on Loops

- Two kinds – counter-controlled and sentinel-controlled.
- **Counter** – runs until counter reaches a limit
  Eg: for each of 117 students in a class
  Use when the number of repetitions is known
- **Sentinel** – runs until a certain condition is encountered.
  Eg: '\n' (newline) is encountered in the input.
  Use when the number of repetitions is a property of the input and not known *a priori*

For loops

Counter controlled iteration needs –

- Initial value,
- modification of counter: `+i`, `-i`, any other arithmetic modification depending on the problem
- Final value

`for` iteration structure provides for the programmer to specify all these

Everything that is provided by `for` can be achieved 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

Replace our previous program by the following

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

Observe: a mistake in the earlier program is gone.

CS101 Lecture 6

Timothy A. Gonsalves

Simple example of *for* statement

Compute the sum of the first 20 odd numbers

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

Init the first odd number

i : Loop control variable

Termination condition

Increment sum by the i^{th} odd number

the next odd number

Note: in `for`, $expr_1$, $expr_2$, $expr_3$
-- should involve the loop control variable only

Calculating compound interest

Principal: Rs. 1000/-; rate of interest: 5% pa; period: 10 yrs

```
#include <stdio.h>
main( )
  {
   int yr;
   float amt, principal = 1000.0, rate = .05;
   amt = principal;
   printf("%4s%21s\n", "year","Amount in deposit");
   for (yr = 1; yr < = 10; yr++) {
        amt += amt * rate;
          printf("%4d%21.2f\n", yr,
  }
}
```

Initial value of variable set in its declaration

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

Example for *while* construct

Print the reverse of a given integer:

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

An Example

x is the given number

y is the number being computed

y = 0	x = 56342
y = 0*10 + 2 = 2	x = 5634
y = 2*10 + 4 = 24	x = 563
y = 24*10 + 3 = 243	x = 56
y = 243*10 + 6 = 2436	x = 5
y = 2436*10 + 5 = 24365	x = 0

Termnation condition: Stop when x becomes zero

Reversing a number

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

integer division truncates the quotient

Perfect number detection

Perfect number: sum of proper divisors add up to the number

Consider 4: 1 + 2 = 3 ➔ not perfect

Consider 6: 1 + 2 + 3 = 6 ➔ perfect

Insight: smaller proper divisor is 1
largest proper divisor is $\lfloor n/2 \rfloor$

Method:

For each integer between 1 and $\lfloor n/2 \rfloor$
  If it is a divisor of *n*, add to *sum*

At the end, if *sum* == *n*, *n* is perfect

Perfect number detection

Perfect number: sum of proper divisors add up to the number

```
int main ( )
{
  int d, n, sum;
  scanf("%d", &n);

  for (sum = 1, d = 2; d <= (n/2); d++)
    {
      if (n%d == 0) sum += d;
     }

  if (sum == n)
       printf ("%d is perfect\n", n);
  else printf("%d is not perfect\n", n);
}
```

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

Exercise: Modify to find the first n perfect numbers

The *do while* construct

for and *while* check termination condition before each evaluation 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

CS101 Lecture 7

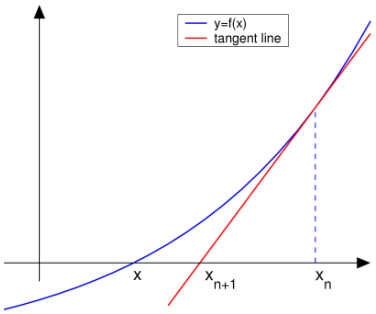
Timothy A. Gonsalves

Square Root: Newton–Raphson method

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

f ' denotes the derivative of the function *f*.
By simple algebra we can derive:

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



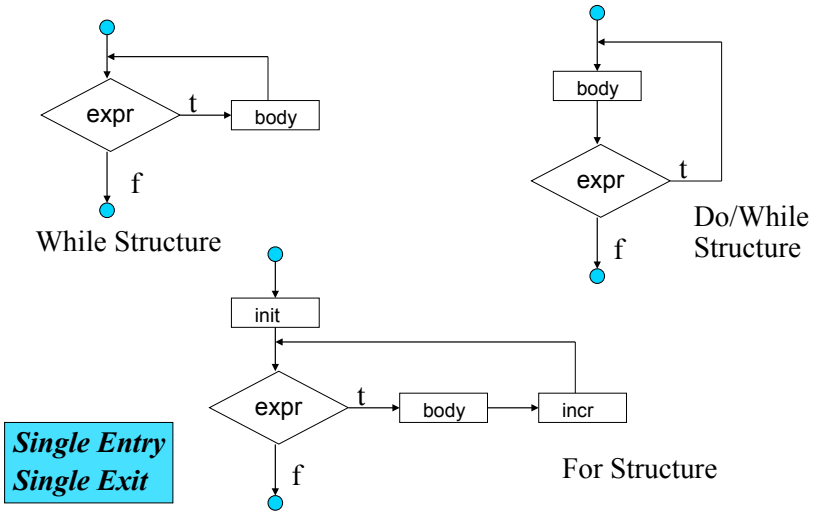
http://en.wikipedia.org/wiki/Newton's_method

Square root of a number

How do you solve the equation "$x^2 = k$" ?

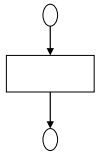
```
int main()
  {
      int k;
      float prevGuess, currGuess, error, sqRoot;
      scanf("%d", &k);
      currGuess = (float) k/2; error = 0.0001;

      do
       {
          prevGuess = currGuess;
          currGuess = (prevGuess + k/prevGuess)/2;
       } while (fabs(prevGuess-currGuess) >error);

   sqRoot = currGuess;
   printf("%f\n", sqRoot);
  }
```


Repetition Structures



Structured Programming

To produce program that are
- easier to develop, understand, test, modify
- easier to get correctness proof



Rules

1 Begin with the "simplest flowchart".

2 Any action box can be replaced by two action boxes in sequence.

3 Any action box can be replaced by any elementary structures (sequence, if, if/else, switch, while, do/ while or for).

4 Rules 2 and 3 can be applied as many times as required and in any order. Stepwise refinement

Spaghetti programming

1. First line
2. IF (condition1) Goto 13
3. Third line
4. If (condition 2) go to 7
5. Fourth line
6. Fifth line
7. Go to 3
8. Sixth line
9. If (condition 3) go to 12
10. Go to 8
11. Line number 11
12. Line 12
13. If (condition 4) go to 1
14. Print ("I' m done finally !")
15. End

Exercises

Write a program that reads in the entries of a 3 by 3 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.

An Array

- A data structure containing items of *same* type
- int marks[7] = {22,15,75,56,10,33,45};
  - a contiguous group of memory locations named "marks" for holding 7 integer items
  – One name for many memory locations
  - elements/components - variables
    marks[0], marks[1], … , marks[i]
    i - index or subscript ($0 \le i \le 6$)
  - the value of marks[2] is 75
  - new values can be assigned to elements
    marks[3] = 36;

marks	
22	0
15	1
75	2
56	3
10	4
33	5
45	6

Example using arrays

Read ten numbers *into an array* and compute their average

```
#include <stdio.h>
#define SIZE 10
main(){
  int numbers[SIZE], sum = 0, i;
  float avg;
  for (i = 0; i < SIZE; i++)
    scanf("%d", &numbers[i]);
  for (i = 0; i < SIZE; i++)
    sum += numbers[i];
  avg = (float) sum/SIZE;
  printf("The average is: %f\n",avg);
  return 0;
}
```

Typecast

Good practice to always use *return*

Switch Selection Structure

In place of the *else if* for a multiway selection

Syntax – *if (condition_1){execute these}*

else if (condition_2) {execute these}

else if (condition_3) {execute these}

and so on…..

Switch replaces *else if* for a very special case

Syntax – *switch(expression){*

case const-expr: statements

case const-expr: statements

Counting digits in text (Kernighan & Ritchie, p.59)

```
#include<stdio.h>
main()
{
  int c, i, nWhite, nOther, nDigit[10];
    nWhite = nOther = 0;
    for(i=0;i<10;i++) nDigit[i]=0;

    while((c = getchar()) ! = EOF){
      switch(c){

  case'0':case'1':case'2':case'3':case'4':case'5':
            case'6':case'7':case'8':case'9':
                                  nDigit[c-'0']++;
                                  break;
```

An array of ten integers

… Counting digits

```
      case' ':
      case'\n':
      case'\t': nWhite++; break;
      default:  nOther++; break;
       }
   }
  printf("Digits: ")
  for(i=0;i<10;i++)
     printf("%d occurred %d times \n", i,
                                  nDigit[i]);
  printf("White spaces: %d, other: %d\n",
                           nWhite, nOther);
}
```

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.

More on this later.

Polynomial Evaluation

Evaluate

$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots a_1 x + a_0$

for a given x value.

Computing each term and summing up

n + (n-1) + (n-2) + … + 1 + 0 = n(n+1)/2 multiplications and n additions

Improved Method:

$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \ldots + x(a_{n-1} + xa_n))))$

for instance, $p(x) = 10x^3 + 4x^2 + 5x + 2$

$= 2 + x(5 + x(4 + 10x))$

n multiplications and n additions – will run faster!

… Polynomial Evaluation

```
#include <stdio.h>
#define MAX_COEFF 20
main(){
  int coeff[MAX_COEFF], n, x, value, i;
  scanf("%d%d", &n, &x); //read degree and
                         // evaluation point
  for(i = 0; i <= n; i++)
    scanf("%d", &coeff[i]);  // read coeffs
  value = coeff[n];          // a0
  for(i = n-1; i >= 0; i--) // evaluate p(x)
    value = x*value + coeff[i];
  printf("Value of p(x) at x = %d is %d\n",
                                  x, value);
}
```


More Exercises

1. Sort an array of numbers into ascending order.
   a) Write the output into another array
   b) Assuming that arrays are expensive, use only one array: read in the values into an array, sort *in place,* and print out the array.
2. *Matrix Sorting* – The input is a matrix. Identify a sequence of column interchanges such that in the resulting matrix the rows are all sorted in ascending order. Can every matrix be sorted?