IC150 Lecture 3

Program Development:

Pseudo-code and Stepwise Refinement

Basics of C:

Arithmetic and logical operations

Simple input and output

Timothy A. Gonsalves

An Arithmetic Program

- Goal: Write a program that adds two numbers and finds their average
- Method:
  1. Input: Get the numbers from the user
  2. Compute: Do the arithmetic
  3. Output: Give the results to the user

Refine the solution

1. Input:
   1. Read one number
   2. Read second number
2. Compute:
   1. Let sum ← number 1 + number 2
   2. Let avg ← sum / 2
3. Output:
   1. Print sum and average

Refine Pseudo-code Further

1. Input:
   1. Read one number
      1. Prompt the user to enter a number
      2. Let n1 ← number read from user
   2. Read second number
      1. Prompt the user to enter a number
      2. Let n1 ← number read from user
2. Compute:
   1. Let sum ← n1 + n2
   2. Let avg ← sum / 2
3. Output:
   1. Print sum and avg

Convert Pseudo-code to C

```
// Find sum and average
#include <stdio.h>
main()
{
  int n1, n2, sum, avg;
  printf("Enter first number: ");
  scanf("%d", &n1);
  printf("Enter second number: ");
  scanf("%d", &n2);
  sum = n1 + n2;
  avg = sum / 2;
  printf("The sum is %d\n", sum);
  printf("The average is %d\n", avg);
  exit(0);
}
```

Declarations, must precede use

%d - specifies decimal
& - address of variable

assignment

Returning 0 is used to signify normal termination

Program Development Methodology

1. Devise a step-by-step procedure (algorithm) and the necessary data structures (variables)
2. Write the algorithm in pseudo-code
3. Refine each step of the pseudo-code
4. When the pseudo-code is sufficiently simple, convert it to C (or other programming language)
5. Hand-simulate the code with various inputs
6. Type it into a file, compile and test with various inputs

Arithmetic operators in C

Four basic operators

+ , – , * , /

addition, subtraction, multiplication and division

applicable to integers and floating point numbers

integer division - fractional part of result truncated

12/ 5 → 2, 5/9 →0

modulus operator: %

x % y gives the remainder after x is divided by y

applicable only for integers, not for float or double

Order of evaluation (operator precedence)

first parenthesized subexpessions
- innermost first

second * , / and % - left to right

third + and – - left to right

a + b * c * d % e – f / g
4 1 2 3 6 5

a + (((b * c) * d) % e) – (f / g)

good practice -- use parentheses rather than rely on precedence rules -- better readability

Precedence – another example

Value = a * (b+c) % 5 + x / (3 + p) – r - j

Evaluation order –

1. (b+c) and (3+p) : due to brackets
2. * and % and / have same precedence: a(b+c) is evaluated first, then *mod 5*. Also, x/(3+p).
3. Next, the additions and subtractions are done from left to right.
4. Finally, the assignment of the RHS to LHS is done.
   = is done right to left, e.g. a = b = 5;

Relational and logical operators

A logical variable can have two values
{true, false} or {t,f} or {1, 0}

In C: int flag // 0 is false, any non-zero value is true

!	unary logical negation operator
< , <= , > , >=	comparison operators
== , !=	equality and inequality
&&	logical AND operator
||	logical OR operator
	logical operators return true/false

order of evaluation - as given above

note: assignment (=) vs equality (==)

Increment and decrement operators

unusual operators - prefix or postfix
only to variables

++	adds 1 to its operand
--	subtracts 1 from its operand
n++	increments n after its use
++n	increments n before its use

n = 4 ; x = n++; y = ++n;

x is 4 , y is 6 and n is 6 after the execution

Assignment statement/expression

- Form: *variable-name = expression*

  total = test1Marks + test2Marks + endSemMarks;

  int i; float x;

  i = x; fractional part of x is dropped

  x = i; i is converted into a float

- Multiple assignment:

  x = y = z = a + b

  x = (y = (z = a + b))

Assignment operators

expression n = n + 10;

abbreviated form: n += 10; (assignment operator)

most binary operators: corr. assignment operator

X *op*= expr

is equivalent to

X = X *op* (expr)

op : +, – , *, / , %

Advantage: Conciseness

Output Statement

printf(format-string, var_1, var_2, …, var_n);

format-string indicates:

how many variables to expect

type of the variables

how many columns to use for printing them

any character string to be printed

– sometimes this would be the only output

enclosed in double quotes

Examples - output

int x; float y;

x = 20; y = – 16.7889;

printf("Value x = %d and value y = %9.3f\n", x, y);

'%d' , '%9.3f' : conversion specifiers

'd' , 'f' :conversion characters

The output:

Value x = 20 and value y = –16.789

- blank space (9 spaces)

General form

General conversion specifier: %*w.p c*

w : total width of the field, (optional)

p : precision (digits after decimal point)

c : conversion character

Conversion Characters:

d - signed decimal integer

u - unsigned decimal integer

o - unsigned octal value

x - unsigned hexadecimal value

f - real decimal in fractional notation

e - real decimal in exponent form

Input Statement

scanf(format-string, &var$_1$, &var$_2$, …, &var$_n$);

Format String:

types of the data items to be stored in var$_1$ etc enclosed in double quotes

Example: scanf("%d%f ", &marks, &aveMarks);

data line : 16 14.75

scanf skips spaces and scans more than one line to read the specified number of values

Conversion Specifiers for "scanf"

- d - read a signed decimal integer
- u - read an unsigned decimal integer
- o - read an unsigned octal value
- x - read an unsigned hexadecimal value
- f - read a real decimal in fractional notation
- e - read a real decimal in exponent form
- c - read a single character
- s - read a string of characters

Solving a quadratic equation (*rm –i* is safe)

Procedure or recipe to solve $ax^2 + bx + c = 0$

1. Inpute: Get the numerical values of a, b and c
2. Compute: Evaluate $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
3. Output: Print out the roots, x_1 and x_2

A step-by-step procedure that terminates is an **algorithm**

Solving a quadratic equation (*rm –i* is safe)

```
// Solve a*x^2 + b*x + c = 0
#include<stdio.h>
#include<math.h>
main()
{  float a, b, c;
   float discrim;    // discriminant b^2-4ac
   float x1, x2;     // the two roots
   float denom;
  // prompt for each coeff, read and store it
   printf("Enter the 1st coefficient:");
   scanf("%f",&a);
```

Quadratic (continued)

```
  printf("Enter the 2nd coefficient:  ");
  scanf("%f", &b);
  printf("Enter the 3rd coefficient:  ");
  scanf("%f", &c);
          /* Now compute the roots*/
  discrim = pow(b, 2) – 4* a* c;
  denom = 2*a;
  x1 = (-b + sqrt(discrim))/denom;
  x2 = (-b – sqrt(discrim))/denom;
  printf("the roots were %f,  %f \n", x1,x);
}
```

$b^2 - 4ac$

Exercise (see *http://www.gnu.org*)

Modify the program so that the quadratic is also output.

Summary: Variables are modified as the program runs.

Problem Solving with Variables

- Write a program that will take two degree 5 polynomials as input and print out their product.
- What are the inputs?
  - Coefficients from each polynomial. Six from each.
  - We need 12 *Input variables.*
- How many outputs are there?
  - We need 12 *Output variables*

Another exercise (*www.howstuffworks.com*)

- Write a program that takes as input 5 digit numbers and prints them out in English.
- Example: 512 – Five Hundred and Twelve

Solve the problem first, identify input variables, Output variables, intermediate variables.

What values are taken by the intermediate variables, how they are calculated from input values, and output variables.