Structured Program Development

Algorithms

- A precise statement of actions to be executed in a sequential manner to solve a problem
- The order of execution of statements in a computer program is called *program control*

Pseudocode

- A high-level language (like a natural language) to describe in detail the steps to be carried out to solve a problem
- Contains only the statements to be executed
- Useful for developing algorithms
- Cannot be directly executed on a computer
- Can be directly converted to a computer-programming language without much effort

Control structures

- All computer programs can be written using just three control structures
  1. Sequence structure, such as assignment statement
  2. Selection structure, such as `if-then-else`
  3. Repetitive structure, such as `for` or `while` loops
- Flowcharts
  - Graphical representation of an algorithm
  - Control flow illustrated by using *flowlines*
  - Useful for developing and representing but is mostly substituted by pseudocode
  - Action symbol or rectangle
    * Represents any action – calculation or input/output
  - Oval symbol
    * Used to represent beginning and termination of an algorithm
    * Single entry/single exit control structures
  - Connector symbol or circle
    * Connects two portions of an algorithm
    * Useful when the algorithm does not fit into one page
  - Decision symbol or diamond
    * Represents the selection structure
    * `if-then` or single selection structure
    * `if-then-else` or double selection structure
    * `switch` or multiple selection structure
    * Also may be used to represent the repetition structure
    * `while` loop
    * `do-while` loop
    * `for` loop

The if selection structure

- Selects an alternative course of action, if possible
- In pseudocode
  if denominator is not equal to zero
  result is numerator ÷ denominator
- Flowchart symbols for the code
- In C

```
if ( denom != 0 )
    result = num / denom;
```

- A still better way to write the same will be

```
if ( denom )
    result = num / denom;
```

- Must apply consistent and responsible indentation (4 spaces right of the column where `if` begins)

The if/else selection structure

- Allows the selection of one (and only one) of the two alternatives
- In pseudocode
  if denominator is not equal to zero
      result is numerator ÷ denominator
  else
      print "Did not evaluate to avoid division by zero"
- Flowchart symbols for the code
- In C

```
if ( denom )                /* is not equal to zero */
    result = num / denom;
else
    printf ( "Did not evaluate to avoid division by zero\n" );
```

- Both statements of the `if` and `else` part should be consistently indented
- With several *blocks* or *compound statements*, each block should be indented with respect to the containing block in a consistent manner
- The conditional operator (`?:`)
  - The only ternary operator available in C
  - Closely related to the `if/else` structure
  - The above `if/else` statement can be written as

    ```
    denom ? result = num / denom : printf ( "Did not evaluate to avoid division by zero\n" );
    ```

  - You can also assign the results as follows:

```
#define INFINITY 9999999999
result = denom ? num / denom : INFINITY;
```

- Nested `if/else` structure
  - Allows for multiple choices
  - Assigning grade in the class

Raw score	Grade
$S \geq 90$	A
$80 \leq S < 90$	B
$70 \leq S < 80$	C
$60 \leq S < 70$	D
$S < 60$	F

  - The C code can be written as

```
if ( s >= 90 )
    grade = 'A';
else
    if ( s >= 80 )
        grade = 'B';
    else
        if ( s >= 70 )
            grade = 'C';
        else
            if ( s >= 60 )
                grade = 'D';
            else
                grade = 'F';
```

- Never place a semicolon at the end of the condition in the `if` statement
- In `vi` you can use the percent character (%) to match the braces surrounding a compound statement
- In `vi`, you can include `set showmatch` in the `.exrc` file to show how the braces match
- A compound statement can be placed anywhere a single statement can be placed
- An empty statement can be represented by a simple semicolon (`;`)
  - The following is valid

```
if ( denom )
    result = num / denom;
else
    ;
```

The while repetition structure

- Used to continue doing something as long as a condition is met
- Computing the sum of first n integers
- Pseudocode
  Initialize a counter to 1
  while counter $\leq n$
      Add counter to sum
      Increment counter by 1

- The C code

```
/*****************************************************************************/
/*                                                                           */
/* summ -- Program to do the summation of integer from 1 to n                */
/* Written by: Sanjiv K. Bhatia                                              */
/* Date      : October 1, 1996                                               */
/* Input     : A number n                                                    */
/* Output    : 1 + 2 + 3 + ... + n                                           */
/* Limitation: n must be positive integer                                    */
/*                                                                           */
/*****************************************************************************/
#include <stdio.h>

main()
{
    int n,              /* To hold the number of integers, from 0 to n  */
        i,              /* Temporary counter                            */
        sum;            /* Current sum                                  */

    printf ( "Please enter a positive integer: " );
    scanf ( "%d", &n );

    if ( n < 0 )
    {
        printf ( "I cannot work with negative numbers\n" );
        printf ( "Aborting the program\n" );
        exit ( 1 );
    }

    i = sum = 0;
    while ( i <= n )
    {
        sum = sum + i;
        i = i + 1;
    }

    printf ( "The sum of the series 0 + 1 + ... + %d is %d\n", n, sum );
}
```

- Infinite loops
  - Set up such that the condition being checked in the loop never becomes false
  - Avoid such loops, if possible
- Flowchart symbols

Counter-controlled loops

- Just like the loop above
- You know the number of times the loop should be traversed

- Also known as *definite repetition*
- The number must be known before the start of loop

Sentinel-controlled loops

- Used when you do not know the number of times the loop must be traversed
- Also known as *indefinite repetition*
- Terminated by a *dummy value*, a *signal value*, or a *flag value*
- Write a program to compute the sum of a number of positive integers
- The C code

```
/******************************************************************************/
/*                                                                            */
/* summ -- Program to add a set of positive integers                          */
/* Written by: Sanjiv K. Bhatia                                               */
/* Date      : October 8, 1996                                                */
/* Input     : A set of positive integers, terminated by zero                 */
/* Output    : Sum of the integers                                            */
/* Limitation: Program does not check for negative integers                   */
/*                                                                            */
/******************************************************************************/
#include <stdio.h>

main()
{
    int n,              /* To hold the integer to be input by the user  */
        sum;            /* Current sum                                  */

    printf ( "Please enter a series of integers, terminated by a zero:\n" );
    scanf ( "%d", &n );

    sum = 0;
    while ( n )
    {
        sum = sum + n;
        scanf ( "%d", &n );
    }

    printf ( "The sum is %d\n", sum );
}
```

- The sentinel value must be a legitimate data value
- The prompt for inputting data values must explicitly remind the user about the legitimate value to terminate the loop (or data entry phase)
- Three logical phases of any program
  1. Initialization phase

2. Processing phase
3. Output phase

- Top-down stepwise refinement process for developing algorithms
- Promotion or implicit conversion of data types
  - Ensures that the data items used in an operation are of the same type
  - For example, `int` is promoted to `float`
- Explicit conversion
  - Performed by using the cast operators
  - If `n` is an `int`, it can be converted to `float` by typecasting as `(float)n`
  - All the typecasting operators are unary operators
  - Typecasting operators are of higher priority compared to *, /, and %, but of lower priority than the parentheses
- Specifying precision for printing
  - Achieved by specifying the number of digits after the decimal point
  - The numbers are rounded to the required precision
  - The print statement can be specified as

    ```
    printf ( "%.2f\n", 3.14159 );
    ```

  - The precision digits should not be used in the `scanf` statement

Nested control structures

- Assigning grades to the students in a class
  - Get a student's name
  - For each student, get individual test and assignment grades
  - Compute the GPA for each student and print it against the student's name
- Must write a pseudocode and test it by yourself in great detail
- Correct pseudocode implies that most of the problem has been solved

Assignment operators

- Used to abbreviate assignment expressions
- `sum = sum + n` can be abbreviated to `sum += n`
- No space between arithmetic operator and = sign
- Abbreviation works with all the arithmetic operators

Increment and decrement operators

- Unary operators

- `i = i + 1` can be expressed as `i++`
- Difference between `i++` (postincrement) and `++i` (preincrement)
- Cannot be applied to an arithmetic expression with more than one operand
- Must be careful with the use, especially in things like `printf`
- Advantage of the operators
  - Consider the code

    ```
    while ( b )              /* is not equal to zero */
    {
        p += a;
        b--;
    }
    ```

  - It can be more efficiently written as

    ```
    while ( b-- )            /* is not equal to zero */
        p += a;
    ```

More input/output functions

- `getchar` – Get a character from the user
  - To read a character in a variable, use the following code:

    ```
    char c;
    c = getchar();
    ```

- `putchar` – Put a character on the screen
  - A character can be displayed on the screen by using the following code

    ```
    char c;
    putchar ( c );
    ```

- Problems with `scanf` and solutions
  - `scanf` creates lots of trouble when reading numbers and characters together, especially in the handling of end of line character
  - From this point onwards, we will not even use it for that reason
  - We'll use a combination of other routines to get around the problems

    ```
    char line[100];     /* Line of keyboard input */

    fgets ( line, sizeof ( line ), stdin );
    sscanf ( line, format, &var1, &var2, ... );
    ```

Infinite while loop

- An infinite `while` loop can be written as

```
while ( 1 )
```