## Program Control

### Essentials of repetition

- Loop control variable
  - Counts the number of repetitions in the counter-controlled loop
  - Controls the execution of loop
- Sentinel value
  - Indicates the end of data when the number of data is not known in advance
  - Must be distinct from the data but be of the same type

### Counter-controlled repetition

- Requires the following to be known in advance
  1. Name of a control variable, or loop counter
  2. Initial value of the control variable
  3. Increment (or decrement) by which to modify the control variable in each traversal
  4. Condition to test the final value of control variable
- Modifying the control variable in the loop head
  - Makes the loop more efficient
- Must be careful when using floating point variables to control loops
- Do not forget the indentation
- And the white space
- Keep the level of nesting manageable

### The "for" loops

- Powerful mechanism to perform counter-controlled loops
- Power comes from the ability to perform major loop-related functions, such as initialization, testing, and incrementing the loop-control-variable automatically without adding any complications
- Consider the program we wrote with `while` loop to perform the summation $\sum_{i=1}^{n} i$
- The same program with `for` loop is written as

```
/*************************************************************************/
/*                                                                       */
/* summ -- Program to do the summation of integer from 1 to n            */
/* Written by: Sanjiv K. Bhatia                                          */
/* Date      : October 14, 1996                                          */
/* Input     : A number n                                                */
/* Output    : 1 + 2 + 3 + ... + n                                       */
/* Limitation: n must be positive integer                                */
```

```
/*                                                                      */
/************************************************************************/
#include <stdio.h>

int 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 );
    }

    sum = 0;
    for ( i = 0; i <= n; i++ )
        sum += i;

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

    return ( 0 );
}
```

- The loop head has three statements, separated by semicolons
  1. Initialization statement for control variable
  2. Testing of the condition on control variable
  3. Modification of the control variable
- Be careful in using the operator in the testing part; `<` and `<=` are not the same
- The body of the loop can be made of a compound statement
- The general format for a `for` loop and its equivalent `while` loop can be described as

| | |
|---|---|
| `for ( expression1; expression2; expression3 )`<br>`  statement;` | `expression1;`<br>`while ( expression2 )`<br>`{`<br>`  statement;`<br>`  expression3;`<br>`}` |

- Comma operator
  - Can be used to replace the initialization and modification parts in the `for` loop with a list of expressions
  - The list of expressions gets evaluated from left to right
  - As an example, the `for` statement head in the above program could be replaced by

```
for ( i = 0, sum = 0; i <= n; i++ )
```

  - Doing so will remove the initialization statement for `sum`

- The loop body itself could be changed to an empty statement
  - The above loop statement could be changed to

```
for ( i = 0, sum = 0; i <= n; sum += i, i++ );
```

  - Not a good way to write the loop and may lead to errors

- Both comma and semicolon have a distinct role in the loop header, and should not be confused
- Be careful when you place a semicolon just to the right of the semicolon in the `for` loop header
- The modification part in the loop header can also decrement a variable, or perform some other operation on it
- The initialization part can also be any valid statement

```
/*************************************************************************/
/*                                                                       */
/* summ -- Program to add a set of integers given by the user            */
/* Written by: Sanjiv K. Bhatia                                          */
/* Date      : October 15, 1996                                          */
/* Input     : A set of integers terminated by zero                      */
/* Output    : Sum of the integers                                       */
/*                                                                       */
/*************************************************************************/
#include <stdio.h>

int main()
{
    int sum = 0,        /* Current sum                                  */
        num;            /* Number to be read                            */

    printf ( "Please enter the integers to be added (0 to stop):\n" );

    for ( scanf ( "%d", &num ); num; scanf ( "%d", &num ) )
        sum += num;

    printf ( "The sum of the numbers is %d\n", sum );

    return ( 0 );
}
```

- The control variable can also be changed within the body of the loop but should be avoided as it may lead to errors
- `for` and `while` loops can be interchanged
- Any of the expressions from the `for` statement may be omitted, making the following expression valid

```
for ( ; ; );
```

- Consider the following loop

```
for ( ; i > 0 ; )
{
    ...
}
```

- `i` is not reinitialized and uses the value previously assigned
- The value of `i` is modified within the loop body
- The loop continues as long as `i` is positive

**The switch statement**

- Multiple selection structure to decide between a number of available choices
- Uses a series of `case` labels, and an optional `default` case
- The general syntax is:

```
switch ( expression )
{
  case constant1: statement ... statement
  case constant2: statement ... statement
  ...
  default       : statement ... statement
}
```

- The `expression` in the `switch` header is known as the *controlling expression*
- The constants to the right of `case` are known as `case` labels
- No variable is allowed in the constant expression to the left of colon
- The constant prefix may occur more than once before a sequence of statements
- The `break` statement is used to jump out of the `switch` statement
- Each `case` can have one or more actions

```
switch ( telephone_number )
{
  case 398474 :
  case 987619 :
    telephone_number = 844564; break;
  case 730488 :
    telephone_number = 844565; break;
  default :
    printf ( "The telephone number %d was not found\n", telephone_number );
}
```

- Program to guess a city starting with a letter

```
/* Program to demonstrate the use of switch statement and character input */

#include        <stdio.h>
```

```
int main()
{
    char ch;                        /* To read in a value     */

    printf ( "Please type a character followed by enter " );
    printf ( "(or just enter to stop program): ");

    while ( ( ch = getchar() ) != '\n' )
    {
        switch ( ch )
        {
        case 'a' :
        case 'A' : printf ( "Amsterdam\n" ); break;

        case 'b' :
        case 'B' : printf ( "Bombay\n" ); break;

        case 'c' :
        case 'C' : printf ( "Cairo\n" ); break;

        default  :
            printf ( "Sorry, I do not know a city starting with " );
            printf ( "that character\n" );
        }

        /* Flush out the input buffer */

        while ( ( ch = getchar() ) != '\n' );

        printf ( "Please type a character followed by enter " );
        printf ( "(or just enter to stop program): ");
    }

    printf ( "Thanks for using the city guesser\n" );

    return ( 0 );
}
```

- The `EOF` constant and its use as a sentinel value
  - In Unix, `EOF` is `^D`
  - In MS-DOS, `EOF` is `^Z`
  - Using `EOF` makes the code more portable
- `break` statement
  - The statement
    `break;`
    terminates the innermost loop containing the statement
  - The following segments are equivalent

```
while ( 1 )
{
  ch = getchar();
  if (ch == '+')
    break;
  ...
}
```

```
while (ch = getchar(), ch != '+')
   ...
```

```
while ( (ch = getchar()) != '+
   ...
```

**The do/while repetition structure**

- Similar to `while` loops with condition testing at the end of the loop
- Major implication – The body of the loop is traversed at least once
- The syntax is:

```
do
    statement
while ( condition );
```

- Example

```
do
{
    s += i;
    i++;
} while ( i <= n );
```

- The following is valid

```
do ; while ( 1 );
```

- Always include braces to include the body of the `do/while` loop even though they are not required

**The continue statement**

- The statement
  `continue;`
  causes a jump to the test for loop termination
- The remaining part of the loop is skipped for that iteration only
- The following codes are equivalent

```
while ( i++ < n )
{
  if ( i * i < s )
  {
    ...
  }
}
```

```
while ( i++ < n )
{
  if ( i * i >= s )
    continue;
  ...
}
```

**Logical operators**

- Already covered with truth tables

**Equality and assignment operators**

- *Never swap the equality operator (==) and the assignment operator (=)*
- `if ( grade = 100 ) letter_grade = 'A';`

**Rules for precedence and associativity**

- The rules of precedence for all the operators in C is given as follows (top (high) to bottom (low))

```
()  []  ->  .
!   ~   ++  --  -   (type) *   &   sizeof
*   /   %
+   -   (binary)
<<  >>
<   >   <=  >=
==  !=
&
^
|
&&
||
?:
=   +=  -=  *=  /=  %=  <<=  >>=  &=  ^=  |=
,
```