• The break statement: causes an immediate exit from the innermost while, do while, or for loop.

Example 4.1.1

Example 4.1.2

• The continue statement: resumes execution of the loop
  – while loop: immediately jump to the top of the loop and test the expression
  – do while loop: immediately jump to the bottom of the loop and test the expression
  – for loop: immediately execute expression3 and then test expression2

• Examples 4.1.3 and 4.1.4

• Application: generating prime numbers
• The switch statement

```
switch (integer expression) {
  case constant 1:
    statements 1
  case constant 2:
    statements 2
  ...
  case constant n:
    statements n
}
```

Figure 4.1

There may be a default statement at the end.

Figure 4.2

Example 4.3.2

Note: If no break statement, execute the next statement.
• The goto statement and labels
goto statement causes an unconditional branch
to the place specified by a label.
goto l1:
:
l1: action

Example 4.4.1
Note: Don’t abuse goto statement.

• Conditional expressions

The value of
expr 1? expr 2:expr 3
is
expr 2 if expr 1 is true else expr 3.

Example:

\[ x = (i > j)?j : i; \]

\( x \) takes the minimum of \( i \) and \( j \).

• Application: Printing a calendar
• The cast operator:
  convert one data type to another
  e.g.: int x;
  ( float ) x
  The original value of x converted to float.
  The type and value of x are unchanged.
  Example: 4.7.1

• Application: Summing a series

• The sizeof operator
  sizeof ( object)
  the amount of memory in bytes required to
  store object
  Example 4.9.1
• Function getchar:
  Reads one character from the standard input

• Function putchar:
  Writes one character to the standard output
  Example 4.10.1

• Bitwise operators
  - Bitwise complement (one’s complement)
    ~
    flips each bit in its operand.
  - Bitwise logical operators
    and: &
    or: |
    exclusive-or: ^
  Examples 4.11.2 and 4.11.3
– Bitwise shift

moves bits right or left

left shift : <<

right shift: >>

Examples: 4.11.4, 4.11.5 and 4.11.6
FUNCTIONS AND PROGRAM STRUCTURE

- Function terminology

- The return statement:
  return exprn; or return ( exprn )

- Function declarations:
  give data type and name of the functions
  and data types of its parameters.
  A function must be declared before it is called.

- The main function

  main ()
  {
  ...
  }

  Execution starts from main.
  The main function has no declaration.
Function definition CANNOT be nested:

```c
void fun1 ( int size)
{
...
void fun2 (char c )
{
...
}
...
}
```

Example: rewrite the bar-graph program
• Two ways to invoke a function:
  – No return value: use the function name, e.g. print_stars (value).
  – Return a value: function can appear anywhere a simple variable can appear. e.g.
    echo_line() != EOF.

• Arguments: information passed from the invoking to the invoked function.
  Parameters: variables defined in the function header.

• Arguments and parameters may have different names, but they should match in number and data type. If the system cannot convert data type of the argument to the data type of the parameter, an error message is issued.
Example 5.2.1
• Order of evaluation of arguments
  C DOES NOT guarantee the order of evaluation of the arguments.
  Example 5.2.2

• Call by value:
  C passes an argument to an invoked function by making a copy of the expression’s value, storing it in a temporary cell, and making the corresponding parameter this cell’s identifier.
  Example 5.3.1.

• Application: Computing resistance

• The scope of variables in a function (the range it can be referenced).
  Local variables (defined within a function) can only be accessed inside the function.
  Example 5.5.1