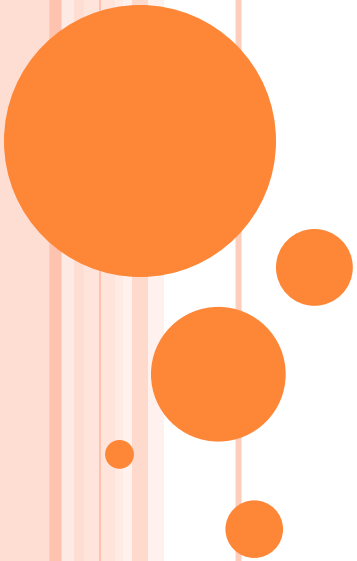


STRUCTURED PROGRAMMING

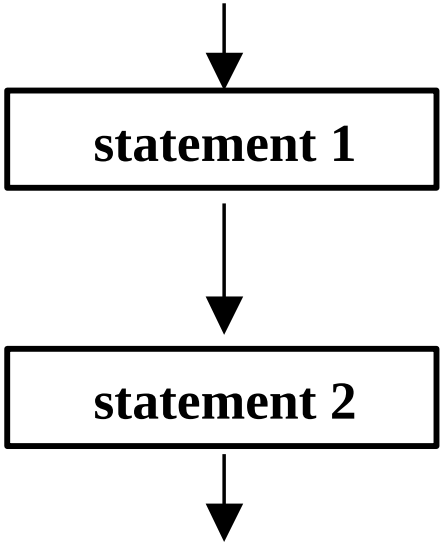
Outline

- Structured Programming
- Top-down stepwise refinement
- Example
- Summary

Structured Programming

- During the 1960s, it became clear that the indiscriminate use of transfers of control through *goto statements* was the root of much difficulty experienced by programmer groups.
- The notion of so-called *structured programming* became almost synonymous with "*goto elimination*."

- Bohm and Jacopini's work demonstrated that all programs could be written in terms of only three control structures:
  - sequence structure
  - selection structure
  - repetition structure



A sequence structure

- The sequence structure is built into C++.
- Unless directed otherwise, the computer executes C++ statements one after the other in the order in which they are written.

- C++ provides three types of selection structures:
  - *if* statement (single-selection structure)
  - *if-else* statement (double-selection structure)
  - *switch* statement. (multiple-selection structure)

- C++ provides three types of repetition structures:
  - *while* statement
  - *do-while* statement
  - *for* statement

- So C++ has only seven control structures: sequence, three types of selection and three types of repetition.

Building programs in good style

- Each C++ program is formed by combining as many of each type of control structures as appropriate for the algorithm the program implements.

- We will see that each control structure has only one *entry point* and one *exit point*. These *single-entry/single-exit control structures* make it easy to build programs.

- One way to build program is to connect the exit point of one control structure to the entry point of the next. This way is called *control-structure-stacking*.

- Another way is to place one control structure inside another control structure. This way is called *control-structure-nesting*.

Indentation

- Consistent applying reasonable *indentation* conventions throughout your programs greatly improves program readability. We suggest a fixed-size tab of about ¼ inch or three blanks per indent.

  For example, we indent both body statements of an *if..else* structure as in the following statement:

```
if (grade >= 60)
    cout << "Passed";
else
    cout << "Failed";
```

Top-down Stepwise Refinement

- Using good control structures to build programs is one of the main principles of structured programming. Another principle of structured programming is *top-down, stepwise refinement*.
- Example: Consider the following problem:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

We begin with a pseudocode representation of the *top*:

Determine the class average for the exam

First Refinement

- Now we begin the refinement process. We divide the top into a series of smaller tasks and list these in the order in which they need to be performed. This results in the following *first refinement*.

  First Refinement:

  Initialize variables

  Input, sum and count the exam grades

  Calculate and print the class average.

  Here only the sequence structure has been used.

Second Refinement

- To proceed to the next level of refinement, we need some variables and a *repetition structure*.

- We need a running total of the numbers, a count of how many numbers have been processed, a variable to receive each grade as it is input and a variable to hold the average.

- We need a loop to calculate the total of the grades before deriving the average.

- Because we do not know in advance how many grades are to be processed, we will use *sentinel-controlled repetition*.

- The program will test for the sentinel value after each grade is input and terminate the loop when the sentinel value is entered by the user.

- Now we come to the pseudo-code of the second refinement.

Second Refinement:

Input the first grade(possibly the sentinel)

While the user has not as yet entered the sentinel

Add this grade into the running total

Add one to the grade counter

Input the next grade(possibly the sentinel)

Calculate and print the class average

Third refinement

- The pseudocode statement

  Calculate and print the class average

  can be refined as follows:

  If the counter is not equal to zero

  set the average to the total divided by the counter

  print the average

  else

  Print "No grades were entered".

- Notice that we are being careful here to test for the possibility of division by zero.

- Now we come to the pseudocode of the third refinement

Third Refinement:

Initialize total to zero
Initialize counter to zero
Input the first grade
While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade
If the counter is not equal to zero
    set the average to the total divided by the counter
    print the average
else
    Print "No grades were entered".

The final C++ program

- Final step: After coding, we come to the following C++ program.

```
#include <iostream.h>
#include <iomanip.h>
int main()
{
   int total, gradeCounter, grade;
   double average;   // number with decimal point for average
    // initialization phase
   total = 0;
   gradeCounter = 0;
   // processing phase
   cout << "Enter grade, -1 to end: ";
   cin >> grade;
```

```
while ( grade != -1 ) {
      total = total + grade;
      gradeCounter = gradeCounter + 1;
      cout << "Enter grade, -1 to end: ";
      cin >> grade;
   }
   // termination phase
   if ( gradeCounter != 0 ) {
      average =  double ( total ) / gradeCounter;
      cout << "Class average is " << setprecision( 2 )
           << setiosflags( ios::fixed | ios::showpoint )
           << average << endl;
   }
   else
      cout << "No grades were entered" << endl;
   return 0;
   }
```

Exercise

Summary

- Structured programming helps to reduce programming errors and to maintain the program easier
- Structured programming requires programmers use control structures that have only one entry point and only one exit point.
- Another principle of structured programming is *top-down, stepwise refinement*