1. **Structured programming**: basic computational structures in high-level languages
   1. **Sequential structure**: processor executes one statement after another until reaching the end.
      1. Most basic structure; basis of instruction sequences.
      2. Most programming implement it.
      3. Reading Execution
         1. Certain instructions permit a shift of control away from the next statement to another in the program.
            1. GOTO: basic machine-level instruction that redirects processor to execute another line of code not in the next line.

Undesirable use in high-level languages.

Results in “spaghetti code”

* 1. Other ways to redirect execution by creating a **codeblock** treated differently by the processor
     1. **Selection (conditional) structures**: statements based in **deductive logic**.
        1. **Single-selection (if statements)**
           1. Necessary for conditions involving int
           2. **Compound conditionals**: there can be multiple conditions using Boolean operators.

&&

||

Etc.

* + - * 1. Example

if (condition)

{

code that is read only if condition is met

}

Next statement

* + - 1. **Double selection (if-else statements)**
         1. Example

if (condition)

{

code that is read only if condition is met

}

else

{

code that is read if no conditions are met

}

* + - * 1. **Double selection structure**: abbreviates if-then-else form.

Example

m > 0 ? m +5 : m\*2

* + - 1. **Multiple selection (if-else-if-else statements)**
         1. Example

if (condition)

{

code that is read only if condition is met

}

else if (condition)

{

code that is read only if previous condition is unmet

}

…

else

{

code that is read if no conditions are met

}

* + - * 1. Limitation: must still evaluate to T or F
        2. **Compound conditionals**: conditional statements can include **shortcuts** of Boolean logic like AND or OR.

Examples

if ((a < 5) && (b > 100))…

Compiler stops after the first expression that is false.

if((a < 5) || (b > 100))

Compiler stops after the first expression that is true.

**Short-circuit evaluation**: asking questions that might otherwise cause a program to crash.

Problem

int x;

if (15/x)

…

/\*if x = 0, program will crash\*/

Fix

if ((x==0) || (15/x))

…

/\*will not execute 15/0 if x==0\*/

* + - 1. **break statement**: when executed in selection (and repetition), it causes an immediate exit from that structure (codeblock) to the statement after the structure.
      2. **Switch case**: similar but different structure to multiple selection.
         1. Not as powerful as if-else-if-else statements.

If-else-if-else statements allow you to ask different questions, but accept only T/F.

Switch can accept different values, not just T/F.

* + - * 1. Only used with char
        2. Example

switch (controlling expression that returns a value)

{

case label\_1:

{

…;

break;

}

case label\_2:

{

…;

break;

}

…

default:

{

…

}

}

* + 1. **Repetition structures (loops)**: permit an action to be repeated several times.
       1. Types
          1. **Sentinel-controlled**: loops which continue running so long as a condition is true, after which the next line is read.
          2. **Counter-controlled**: repetition structure for a fixed number of times.
       2. 3 structures
          1. Counter-controlled

for: carries out the loop continuation test immediately after the counter is incremented at the top of the loop.

Most common loop structure

Prescriptive and formalized

Requires control variable to be defined

Requires a counter variable (number of iterations) to be declared prior to loop.

int n

3 parameters

Initial counter value

Normally 0

Maximum value

When to stop

How the counter is modified after every loop

Typically ++ or –

Handles details of running the loop automatically.

Steps

Executes <init> statement to set initial value of counter variable.

Evaluates <condition> to see if maximum count has been reached.

If false 🡪 exit loop

If true 🡪 keep looping

Execute statements inside loop

Execute <increment statement>

Go back to step 2.

Example 1

for (<init>; <condition>; <increment>)

{

<statements>;

}

Example 2

for (<ctrl var> = <init val>;

<cntr var lessthan/greaterthan/etc. <final value>;

<incrementation def>)

{

<loop body>;

}

Notes

Final value and increment can be mathematical expressions.

There can be decrement.

There can be incrementation by different numbers.

n+= {increment value}

If loop continuation is initially false, the body will never be executed.

Counter variable is not used in body of loop, but it can be.

Arrays

**Infinite loop**: causes computation to continue forever because the end condition is never reached; program will not crash unless it runs out of memory.

One of most hateful bugs.

Difficult to find

Programmer must ensure that the counter variable will converge to the final value.

* + - * 1. Sentinel-controlled

while: entry condition; specifies action to be repeated so long as a condition remains true.

Much less structured than for.

Can be used as counter-controlled.

Variable initialization and updating has to be programmed explicitly.

Checks for loop continuation at the beginning of the loop body.

Could be trouble to place the increment expression after continue.

Syntax

control\_var = <initial val>;

while (loop continuation test on control\_var)

{

<statement1>;

…

<update control\_var somehow>;

}

do/while: exit condition; same as while, except that loop continuation test is done after the body of the loop has been executed.

Least popular

Body is guaranteed to be run at least once.

Syntax

int cont\_var = <initial val>;

do

{

<statements>

<update control variable>

} while (continuing condition);

Method

Executes statements inside braces.

Evaluates continuing condition

If false 🡪 end loop

Else 🡪 repeat

* + - 1. Multiple Conditions
         1. There can be more than one condition in loop structures using the comma operators.

for

for (I = 0, j = 0; i + j < 20; i+= 2, j++)

{

<statements>;

}

* + - 1. Stopping criteria
         1. Sentinel-controlled

Stopping criteria is set for a sentinel variable, and its value is checked at every iteration.

When the value reaches the stopping criterion, the loop ends.

Execution continues to the next statement after the loop.

* + - * 1. Counter-controlled

A variable is set that counts the number of loops.

When the count reaches a pre-set number, the loop ends.

Execution continues to the next statement after the loop.

* + - * 1. Entry condition

The stopping criterion is checked at the start of the loop before any statements are executed.

If the criterion is met, the loop is not executed.

* + - * 1. Exit condition

The stopping criterion is checked after the loop statements are executed once.

Guaranteed to run at least once.

* + - 1. The break statement
         1. When executed in a loop, causes an immediate exit to the first statement after the loop.
      2. The continue statement
         1. Affects execution of repetition structures.

In for: causes increment expression to be executed and the loop continuation test is evaluated thereafter.

In while: loop continuation test is evaluated immediately after continue.

In do/while: loop continuation test is evaluated immediately after continue.

* + - * 1. Causes processor to skip the remaining statements in the loop, but goes on to the next iteration rather than exiting the loop.
        2. Associated with a selection structure within the loop.
      1. The return statement
         1. Causes execution to exit the function altogether.

In main(): ends program execution.

* + - * 1. Can also place a value in its place.

In main(): irrelevant.

1. Interesting Note
   1. C++ allows declaration of variables within the specification of the repetition structure.
      1. Traditionally, C does not.
         1. Some compilers do.
      2. C++ does.
2. Summary
   1. 7 control structures (8 if sequential structure counts)
   2. Repetition structures can be nested within one another at an arbitrary level of nesting.
      1. Nesting causes **combinatorial explosion**: too long to program.
3. Pseudorandom Numbers
   1. Not truly random.
   2. int rand(): returns integer between 0 and RAND\_MAX (32,767)
      1. Found in <stdlib.h>
      2. mod can be used to divide by the maximum number desired.
      3. Will produce the same sequence of numbers every time the program is run.
   3. srand(time(NULL)): seeds srand() with a value from the processor time.
      1. Must include time() with <time.h>.