3.8. Control structures

1.变量定义
2.控制结构 if else elseif

Examples on GitHub

- Repository
- Latest ZIP

3.8.1. Conditional blocks

3.8.1.1. Simple examples

Example of using `if` command with `NO` / `YES` constants and variables with `NO` / `YES` values:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

if(YES)
  message("Condition 1")
endif()

if(NO)
  message("Condition 2")
endif()

set(A "YES")
set(B "NO")

if(A)
  message("Condition 3")
endif()

if(B)
  message("Condition 4")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hif-simple -B_builds
Condition 1
Condition 3
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

Adding `else` / `elseif` :

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

set(A "TRUE")
set(B "FALSE")

if(A)
  message("Condition 1")
else()
  message("Condition 2")
endif()

if(B)
  message("Condition 3")
else()
  message("Condition 4")
endif()

set(C "OFF")
set(D "ON")

if(C)
  message("Condition 5")
elseif(D)
  message("Condition 6")
else()
  message("Condition 7")
endif()

set(E "0")
set(F "0")

if(E)
  message("Condition 8")
elseif(F)
  message("Condition 9")
else()
  message("Condition 10")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hif-else -B_builds
Condition 1
Condition 4
Condition 6
Condition 10
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.1.2. CMP0054

Some of the `if` commands accept `<variable|string>` arguments. This may lead to quite surprising behavior.

For example if we have variable `A` and it is set to empty string we can check it with:

```
set(A "")
if(A STREQUAL "")
  message("Value of A is empty string")
endif()
```

You can save the name of variable in another variable and do the same:

```
set(A "")
set(B "A") # save name of the variable
if(${B} STREQUAL "")
  message("Value of ${B} is empty string")
endif()
```

If CMake policy `CMP0054` is set to `OLD` or not present at all (before CMake 3.1), this operation ignore quotes:

```
set(A "")
set(B "A") # save name of the variable
if("${B}" STREQUAL "") # same as 'if(${B} STREQUAL "")'
  message("Value of ${B} is empty string")
endif()
```

It means operation depends on the context: do variable with name `${B}` present in current scope or not?

```
cmake_minimum_required(VERSION 3.0)
project(foo LANGUAGES NONE)

set("Jane Doe" "")
set(A "Jane Doe")

message("A = ${A}")

if("${A}" STREQUAL "")
  message("A is empty")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hcmp0054-confuse -B_builds
A = Jane Doe
A is empty
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

居然会有这种奇葩结果！！！！！！！！！！

3.8.1.3. Try fix

Since CMake accepts any names of the variables you can't filter out `<variable>` from `<variable|string>` by adding "reserved" symbols:

```
cmake_minimum_required(VERSION 3.0)
project(foo LANGUAGES NONE)

set("Jane Doe" "")
set("xJane Doe" "x")
set("!Jane Doe" "!")
set(" Jane Doe" " ")

set(A "Jane Doe")

message("A = ${A}")

if("x${A}" STREQUAL "x")
  message("A is empty (1)")
endif()

if("!${A}" STREQUAL "!")
  message("A is empty (2)")
endif()

if(" ${A}" STREQUAL " ")
  message("A is empty (3)")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Htry-fix -B_builds
A = Jane Doe
A is empty (1)
A is empty (2)
A is empty (3)
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.1.4. Fix

To avoid such issues you should use CMake 3.1 and `CMP0054` policy:

```
cmake_minimum_required(VERSION 3.1)
project(foo LANGUAGES NONE)

set("Jane Doe" "")
set("xJane Doe" "x")
set("!Jane Doe" "!")
set(" Jane Doe" " ")

set(A "Jane Doe")

message("A = ${A}")

if("x${A}" STREQUAL "x")
  message("A is empty (1)")
endif()

if("!${A}" STREQUAL "!")
  message("A is empty (2)")
endif()

if(" ${A}" STREQUAL " ")
  message("A is empty (3)")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hcmp0054-fix -B_builds
A = Jane Doe
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.1.5. Workaround

For CMake before 3.1 as a workaround you can use `string(COMPARE EQUAL ...)` command:

```
cmake_minimum_required(VERSION 3.0)
project(foo LANGUAGES NONE)

set("Jane Doe" "")
set("xJane Doe" "x")
set("!Jane Doe" "!")
set(" Jane Doe" " ")

set(A "Jane Doe")

message("A = ${A}")

string(COMPARE EQUAL "${A}" "" is_empty)
if(is_empty)
  message("A is empty")
else()
  message("A is not empty")
endif()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hcmp0054-workaround -B_builds
A = Jane Doe
A is not empty
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

Stackoverflow

- CMake compare to empty string with STREQUAL failed

3.8.2. Loops

3.8.2.1. foreach

CMake documentation

- foreach

Example of `foreach(<variable> <list>)` command:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

message("Explicit list:")
foreach(item "A" "B" "C")
  message("  ${item}")
endforeach()

message("Dereferenced list:")
set(mylist "foo" "boo" "bar")
foreach(x ${mylist})
  message("  ${x}")
endforeach()

message("Empty list")
foreach(x)
  message("  ${x}")
endforeach()

message("Dereferenced empty list")
set(empty_list)
foreach(x ${empty_list})
  message("  ${x}")
endforeach()

message("List with empty element:")
foreach(i "")
  message("  '${i}'")
endforeach()

message("Separate lists:")
set(mylist a b c)
foreach(x "${mylist}" "x;y;z")
  message("  ${x}")
endforeach()

message("Combined list:")
set(combined_list "${mylist}" "x;y;z")
foreach(x ${combined_list})
  message("  ${x}")
endforeach()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hforeach -B_builds
Explicit list:
  A
  B
  C
Dereferenced list:
  foo
  boo
  bar
Empty list
Dereferenced empty list
List with empty element:
  ''
Separate lists:
  a;b;c
  x;y;z
Combined list:
  a
  b
  c
  x
  y
  z
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

As you may notice `foreach(x "${mylist}" "x;y;z")` is not treated as a single list but as a list with two elements: `${mylist}` and `x;y;z`. If you want to merge two lists you should do it explicitly `set(combined_list "${mylist}" "x;y;z")` or use `foreach(x ${mylist} x y z)` form.

3.8.2.2. foreach with range

Example of usage `foreach(... RANGE ...)` command:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

message("Simple range:")
foreach(x RANGE 10)
  message("  ${x}")
endforeach()

message("Range with limits:")
foreach(x RANGE 3 8)
  message("  ${x}")
endforeach()

message("Range with step:")
foreach(x RANGE 10 14 2)
  message("  ${x}")
endforeach()
```

3 和 8 都会包括进去的

偏移量为2

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hforeach-range -B_builds
Simple range:
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
  10
Range with limits:
  3
  4
  5
  6
  7
  8
Range with step:
  10
  12
  14
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

也是从0开始索引的

3.8.2.3. while

Example of usage `while` command:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

set(a "")
set(condition TRUE)

message("Loop with condition as variable:")
while(condition)
  set(a "${a}x")
  message("  a = ${a}")
  string(COMPARE NOTEQUAL "${a}" "xxxxx" condition)
endwhile()

set(a "")

message("Loop with explicit condition:")
while(NOT a STREQUAL "xxxxx")
  set(a "${a}x")
  message("  a = ${a}")
endwhile()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hwhile -B_builds
Loop with condition as variable:
  a = x
  a = xx
  a = xxx
  a = xxxx
  a = xxxxx
Loop with explicit condition:
  a = x
  a = xx
  a = xxx
  a = xxxx
  a = xxxxx
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.2.4. break

CMake documentation

- break

Exit from loop with `break` command:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

message("Stop 'while' loop:")
set(a "")
while(TRUE)
  set(a "${a}x")
  message("  ${a}")
  string(COMPARE EQUAL "${a}" "xxx" done)
  if(done)
    break()
  endif()
endwhile()

message("Stop 'foreach' loop:")
foreach(x RANGE 10)
  message("  ${x}")
  if(x EQUAL 4)
    break()
  endif()
endforeach()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hbreak -B_builds
Stop 'while' loop:
  x
  xx
  xxx
Stop 'foreach' loop:
  0
  1
  2
  3
  4
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.2.5. continue

Since CMake 3.2 it's possible to continue the loop:

```
cmake_minimum_required(VERSION 3.2)
project(foo NONE)

message("Loop with 'continue':")
foreach(x RANGE 10)
  if(x EQUAL 2 OR x EQUAL 5)
    message("  skip ${x}")
    continue()
  endif()
  message("  process ${x}")
endforeach()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hcontinue -B_builds
Loop with 'continue':
  process 0
  process 1
  skip 2
  process 3
  process 4
  skip 5
  process 6
  process 7
  process 8
  process 9
  process 10
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

CMake documentation

- CMake 3.2 release notes

3.8.3. Functions

3.8.3.1. Simple

Function without arguments:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

function(foo)
  message("Calling 'foo' function")
endfunction()

foo()
foo()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hsimple-function -B_builds
Calling 'foo' function
Calling 'foo' function
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.3.2. With arguments

Function with arguments and example of `ARGV*`, `ARGC`, `ARGN` usage:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

function(foo x y z)
  message("Calling function 'foo':")
  message("  x = ${x}")
  message("  y = ${y}")
  message("  z = ${z}")
endfunction()

function(boo x y z)
  message("Calling function 'boo':")
  message("  x = ${ARGV0}")
  message("  y = ${ARGV1}")
  message("  z = ${ARGV2}")
  message("  total = ${ARGC}")
endfunction()

function(bar x y z)
  message("Calling function 'bar':")
  message("  All = ${ARGV}")
  message("  Unexpected = ${ARGN}")
endfunction()

foo("1" "2" "3")
boo("4" "5" "6")
bar("7" "8" "9" "10" "11")
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hfunction-args -B_builds
Calling function 'foo':
  x = 1
  y = 2
  z = 3
Calling function 'boo':
  x = 4
  y = 5
  z = 6
  total = 3
Calling function 'bar':
  All = 7;8;9;10;11
  Unexpected = 10;11
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.3.3. CMake style

CMake documentation

- CMakeParseArguments

`cmake_parse_arguments` function can be used for parsing:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

include(CMakeParseArguments) # cmake_parse_arguments

function(foo)
  set(optional FOO BOO)
  set(one X Y Z)
  set(multiple L1 L2)

  # Introduce:
  # * x_FOO
  # * x_BOO
  # * x_X
  # * x_Y
  # * x_Z
  # * x_L1
  # * x_L2
  cmake_parse_arguments(x "${optional}" "${one}" "${multiple}" "${ARGV}")

  string(COMPARE NOTEQUAL "${x_UNPARSED_ARGUMENTS}" "" has_unparsed)
  if(has_unparsed)
    message(FATAL_ERROR "Unparsed arguments: ${x_UNPARSED_ARGUMENTS}")
  endif()

  message("FOO: ${x_FOO}")
  message("BOO: ${x_BOO}")
  message("X: ${x_X}")
  message("Y: ${x_Y}")
  message("Z: ${x_Z}")

  message("L1:")
  foreach(item ${x_L1})
    message("  ${item}")
  endforeach()

  message("L2:")
  foreach(item ${x_L2})
    message("  ${item}")
  endforeach()
endfunction()

function(boo)
  set(optional "")
  set(one PARAM1 PARAM2)
  set(multiple "")

  # Introduce:
  # * foo_PARAM1
  # * foo_PARAM2
  cmake_parse_arguments(foo "${optional}" "${one}" "${multiple}" "${ARGV}")

  string(COMPARE NOTEQUAL "${foo_UNPARSED_ARGUMENTS}" "" has_unparsed)
  if(has_unparsed)
    message(FATAL_ERROR "Unparsed arguments: ${foo_UNPARSED_ARGUMENTS}")
  endif()

  message("{ param1, param2 } = { ${foo_PARAM1}, ${foo_PARAM2} }")
endfunction()

message("*** Run (1) ***")
foo(L1 item1 item2 item3 X value FOO)

message("*** Run (2) ***")
foo(L2 item1 item3 Y abc Z 123 FOO BOO)

message("*** Run (3) ***")
foo(L1 item1 L1 item2 L1 item3)

message("*** Run (4) ***")
boo(PARAM1 123 PARAM2 888)
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hcmake-style -B_builds
*** Run (1) ***
FOO: TRUE
BOO: FALSE
X: value
Y:
Z:
L1:
  item1
  item2
  item3
L2:
*** Run (2) ***
FOO: TRUE
BOO: TRUE
X:
Y: abc
Z: 123
L1:
L2:
  item1
  item3
*** Run (3) ***
FOO: FALSE
BOO: FALSE
X:
Y:
Z:
L1:
  item1
  item2
  item3
L2:
*** Run (4) ***
{ param1, param2 } = { 123, 888 }
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.3.4. Return value

There is no special command to return value from function. You can set variable to the parent scope instead:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

function(boo)
  set(A "123" PARENT_SCOPE)
endfunction()

set(A "333")
message("Before 'boo': ${A}")
boo()
message("After 'boo': ${A}")

function(bar arg1 result)
  set("${result}" "ABC-${arg1}-XYZ" PARENT_SCOPE)
endfunction()

message("Calling 'bar' with arguments: '123' 'var_out'")
bar("123" var_out)
message("Output: ${var_out}")
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hreturn-value -B_builds
Before 'boo': 333
After 'boo': 123
Calling 'bar' with arguments: '123' 'var_out'
Output: ABC-123-XYZ
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.3.5. Return

CMake documentation

- return

You can exit from function using `return` command:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

function(foo A B)
  if(A)
    message("Exit on A")
    return()
  endif()

  if(B)
    message("Exit on B")
    return()
  endif()

  message("Exit")
endfunction()

foo(YES NO)
foo(NO YES)
foo(NO NO)
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hreturn -B_builds
Exit on A
Exit on B
Exit
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

3.8.3.6. CMAKE_CURRENT_LIST_DIR

Value of `CMAKE_CURRENT_LIST_FILE` and `CMAKE_CURRENT_LIST_DIR` is set to the file/directory from where function **is called**, not the file where function **is defined**:

```
# Top-level CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project(foo NONE)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/cmake/Modules")

include(foo_run)

foo_run("123")

add_subdirectory(boo)
```

```
# boo/CMakeLists.txt

foo_run("abc")
```

```
# Module cmake/Modules/foo_run.cmake

set(FOO_RUN_FILE_PATH "${CMAKE_CURRENT_LIST_FILE}")
set(FOO_RUN_DIR_PATH "${CMAKE_CURRENT_LIST_DIR}")

function(foo_run value)
  message("foo_run(${value})")

  message("Called from: ${CMAKE_CURRENT_LIST_DIR}")
  message("Defined in file: ${FOO_RUN_FILE_PATH}")
  message("Defined in directory: ${FOO_RUN_DIR_PATH}")
endfunction()
```

```
[control-structures]> rm -rf _builds
[control-structures]> cmake -Hfunction-location -B_builds
foo_run(123)
Called from: /.../control-structures/function-location
Defined in file: /.../control-structures/function-location/cmake/Modules/foo_run.cmake
Defined in directory: /.../control-structures/function-location/cmake/Modules
foo_run(abc)
Called from: /.../control-structures/function-location/boo
Defined in file: /.../control-structures/function-location/cmake/Modules/foo_run.cmake
Defined in directory: /.../control-structures/function-location/cmake/Modules
-- Configuring done
-- Generating done
-- Build files have been written to: /.../control-structures/_builds
```

CMake documentation

- CMAKE_CURRENT_LIST_DIR
- CMAKE_CURRENT_LIST_FILE

3.8.3.7. Recommendation

To avoid function name clashing with functions from another modules do prefix name with the project name. In case if function name will match name of the module you can verify that module used in your code just by simple in-file search (and of course delete it if not):

```
include(foo_my_module_1)
include(foo_my_module_2)

foo_my_module_1(INPUT1 "abc" INPUT2 123 RESULT result)
foo_my_module_2(INPUT1 "${result}" INPUT2 "xyz")
```

See also

- Module names
- Cache names