Methodology and Programming Techniques

Department of Telecommunications, IEiT

dr inż. Jarosław Bułat

kwant@agh.edu.pl

outline

- Poll (what is difficult or a problem in lab/lecture)
- Loops - Examples
  - many examples
  - break; statement
  - continue; statement
- Arrays
- Git - rejected push

Two iterators in the loop

```
for(int i=0, j=10; i<5 && j>5; ++i, --j ){
    cout << i << " " ;
    cout << j << endl;
}
```

expression 1: int i=0, j=10

expression 2: i<5 && j>5

expression 3: ++i, --j

- Two iterators “i” oraz “j”
- One condition, could be complex
- Expression 3, can modify all iterators
- Results:
  0 10
  1 9
  2 8
  3 7
  4 6
- Replacing elements in an array *

Modification of loop execution

```
#include <iostream>

int main() {
    size_t width = 1920;

    for (int x = 0; x < width; ++x ) {
        if (x == 2) {
            continue;
        } else if (x == 5) {
            break;
        }
        cout << x << endl;
    }
}
```

- continue; starts the iteration from the beginning
- break; finishes the loop
- Result:
  0
  1
  3 ← missing “2”
  4
- Modification of loop execution

Loop in loop - nesting

```
#include <iostream>

int main() {
   size_t width = 1920;
   size_t height = 1080;

   for (int x = 0; x < width; ++x ) {
      for (int y = 0; y < height; ++y ) {
      // test each pixel of image
      }
   }
}
```

- Nested loop
- Iterate over all pixels of the FullHD image
- Inner loop (iterator y)
- Outer loop (iterator x)
- For one x, all y iterations will be executed
- All iterations y do x times (for every x)
- Any number of nesting, suggesting no more than 3

Loop in loop - nesting

```
#include <iostream>

int main() {
   size_t width = 1920;
   size_t height = 1080;

   for (int x = 0; x < width; ++x ) {
      int z = 9;
      for (int y = 0; y < height; ++y ) {
         // test each pixel of image
      }
   }
}
```

- Each iteration means executing new block of instructions (code)
- Variable “z” is created and initialized for every loop iteration

Loop in loop - nesting

```
size_t width = 10;

for (int x = 0; x < width; ++x ) {
        for (int y = 0; y <= x; ++y ) {
                cout << "(" << x;
                cout << "," << y << ") ";
                // upper right triangle
        }
        cout << endl;
}
```

- The iteration y ends with x
- Results:

(0,0)
(1,0) (1,1)
(2,0) (2,1) (2,2)
(3,0) (3,1) (3,2) (3,3)
(4,0) (4,1) (4,2) (4,3) (4,4)
(5,0) (5,1) (5,2) (5,3) (5,4) (5,5)
(6,0) (6,1) (6,2) (6,3) (6,4) (6,5) (6,6)
(7,0) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7)
(8,0) (8,1) (8,2) (8,3) (8,4) (8,5) (8,6) (8,7) (8,8)
(9,0) (9,1) (9,2) (9,3) (9,4) (9,5) (9,6) (9,7) (9,8) (9,9)

Countdown backwards

```
#include <iostream>

int main(){
	size_t size = 10;

	for (int x = size; x >=0; --x ) {
		cout << x << endl;
	}
}
```

- The iterator does not need to be changed by +=1, it can be decremented
- Result:

10

9

...

1

0

Iterator changed every 2

```
#include <iostream>

int main(){
	size_t size = 10;

	for (int x = 0; x < size; x+=2 ) {
		cout << x << endl;
	}
}
```

- The iterator does not need to be changed by +=1
- Result:

0

2

4

6

8

Change the iterator in the body loop

```
#include <iostream>

int main(){
	size_t size = 10;

	for (int x = 0; x < size; ++x ) {
		cout << ++x << endl;
	}
}
```

- Change the iterator inside the body of the loop
- Result:

  1

  3

  5

  7

  9
- Never Ever !!!

Infinite loop

```
for (;;) {
        char c;
        cin >> c;
        if (c=='x') {
                break;
        }
}
```

```
char c;
cin >> c;
while (c != 'x') {
        cin >> c;
}
```

```
char c;
do {
        cin >> c;
} while (c != 'x');
```

```
while (true) {
        char c;
        cin >> c;
        if (c == 'x') {
                break;
        }
}
```

- An infinite loop when we do not know the number of iterations
- End of loop after input of '**x**'
- Declare “c”' inside the loop
  - scope!
- Narrow the range of variable scope

Break out of the inner loop

```
#include <iostream>

int main(){

    for (size_t x = 0; x < 10; ++x) {
        for (size_t y = 0; y < 10; ++y) {
            if (x > 4 && y > 5) {
                break;
                // does not work!!!
            }
        }
    }
  cout << x+10*y << endl;
}
```

- I want to leave **both loops** if x > 4 AND y > 5 and print x + 10 * y
- Break instruction will leave only the inner loop
- What error did i make in cout ???

Break out of the inner loop

```
for (size_t x = 0; x < 10; ++x) {
    bool exitLoop = false;

    for (size_t y = 0; y < 10; ++y) {
        if (x > 4 && y > 5) {
            exitLoop = true;
            break;
        }
    }

    if (exitLoop) {
        break;
    }
}
```

- I want to leave **both loop** if x > 4 AND y > 5
- Exit from inner loop + set exitLoop tag
- Checking the tag at the end of the outer loop

Break out of the inner loop

```
for (size_t x = 0; x < 10; ++x) {
    bool exitLoop = false;

    for (size_t y = 0; y < 10; ++y) {
        if (x > 4 && y > 5) {
            exitLoop = true;
            break;
        }
    }

    if (exitLoop) {
        break;
    }
}
```

- I want to leave **both loop** if x > 4 AND y > 5
- Exit from inner loop + set exitLoop tag
- Checking the tag at the end of the outer loop

Break out of the inner loop

```
size_t x = 0;
bool looping = true;

do {
// bool looping = false; // not in scope !?!

    for (size_t y = 0; y < 10; ++y) {
        if (x > 4 && y > 5) {
            looping = false;
            break;
        }
    }
} while ( looping && ++x < 10);
```

- I want to leave **both loop** if x > 4 AND y > 5
- Exit from inner loop + set exitLoop tag
- do-while loop checks the condition at the end
- The looping declaration must be outside the loop

Break out of the inner loop

```
#include <iostream>

int main(){

    for (size_t x = 0; x < 10; ++x) {
        for (size_t y = 0; y < 10; ++y) {
            if (x > 4 && y > 5) {
                goto exitLoop;
            }
        }
    }
    exitLoop:

    cout << "end" << endl;
}
```

- I want to leave **both loop** if x > 4 AND y > 5
- Unconditional jump goto
- The temptation is big ;-), but Never Ever !!!

Break out of the inner loop

```
#include <iostream>

int main(){

    for (size_t x = 0; x < 10; ++x) {
        for (size_t y = 0; y < 10; ++y) {
            if (x > 4 && y > 5) {
                goto exitLoop;
            }
        }
    }
    exitLoop:

    cout << "end" << endl;
}
```

- I want to leave **both loop** if x > 4 AND y > 5
- Unconditional jump goto
- The temptation is big ;-), but Never Ever !!!

I have many of the same type

how to store them?

Arrays

- A way to organize multiple elements of one type
  - each element can be individually addressed
  - elements are stored in continuous address space
  - cannot resize the array after creation
  - array is a variable (applies to all rules for variables):
    - scope
    - inability to change type
    - the need for memory allocation
    - name

Simple array

```
#include <iostream>
int main(){

    int tab[5];

    tab[0] = 1;
    tab[1] = 4;
    tab[2] = tab[0];
    tab[3] = -10;
    tab[4] = 4;

    for (size_t i = 0; i < 5; ++i) {
        tab[i] = i;
    }
}
```

- Declaration:
  - typ
  - size (number of elements)
- Addressing in square brackets
- **tab[2]** (single element) is an **int** type (in this case)
- **int tab[5]** means a five-element array so tab[0] ... tab[4]
- looks nice with the loop (see condition !!!)

Simple array

```
#include <iostream>
int main(){

    int tab[5];

    tab[0] = 1;
    tab[1] = 4;
    tab[2] = tab[0];
    tab[3] = -10;
    tab[4] = 4;
    // tab[5] do not exist !!!
    tab[5] = 1123;
    // will work and create
    // huge problem !
}
```

- Compiler/runtime does not check boundary!!!
- tab [5] will be executed although it does not exist!!!
- The most common source of “buffer overflow” errors
- Very effective way to access memory but dangerous !!!
- x = 0;
  table[x-1];
- valgrind is used to search for addressing errors

Array initialization

```
#include <iostream>

int main(){

        int tab[5] = {0,1,2,3,4};
        int tax[] = {0,1,2,3,4};

        for (int i = 4; i >= 0; --i ) {
                tab[i] = i*10;
        }

        int x = tab[0];     // ? value ?
        tab[++x] = 7;
        tab[tax[4]] = tab[1];
}
```

- Possible initialization during declaration
- You do not have to specify the size if initialization during the declaration
- Indexing a table always has a natural number <0,1,2,3 .... size-1>
- What value will x have?
- Which position will be entered 7?
- Indirect indexing

Accumulation of data from the array

```
#include <iostream>

int tab[] = {0,1,2,3,4};

int main(){

    int result = 0;
    for (size_t i = 0; i < 5; ++i ) {
        result += tab[i];
    }

    cout << result;
    cout << endl;
}
```

- Accumulation (adding) all elements from the array
- Initialization of “result” is important

Calculating the size of the array

```
#include <iostream>

int tab[] = {0,1,2,3,4};

int main(){

        size_t size;
        size = sizeof(tab)/sizeof(tab[0]);
        int result = 0;

        for (size_t i = 0; i < size; ++i ) {
                result += tab[i];
        }

        cout << result << endl;
}
```

- Not universal solution
- Will not work for pointers (eg. memory allocation by new/alloc)
- sizeof(tab) gives the size in bytes of the entire array
- sizeof(tab [0]) specifies the size in bytes of a single array element

Declaration of the array - size

```
#include <iostream>

int main(){

        size_t size = 10;
        int tab[size];          // c++98

        for (size_t i = 0; i < size; ++i ) {
                tab[i] = 0;
        }
}
```

- **To** c++98 the array size had to be constant (the value known at compile time)
- **From** c++98 (including), size can be variable (implicit dynamic memory allocation)

Max value in array

```
int tab[] = {1,3,6,2,1,6753,2,341,0,1};

int max = 0;
for (size_t i = 0; i < 10; ++i ) {
        if (max < tab[i]) {
                max = tab[i];
        }
}

cout << max << endl;
```

- » Find the maximum value in the array
- » Initialization of max
- » Iteration over all elements of the array
- » Comparison to each element
- » Assign tab[i] to max if tab[i] is bigger
- » When does the algorithm not work?

Max value in array

```
int tab[] = {1,3,6,2,1,6753,2,341,0,1};

int max = tab[0];
for (size_t i = 1; i < 10; ++i ) {
        if (max < tab[i]) {
                max = tab[i];
        }
}

cout << max << endl;
```

- The fastest way
- It worked for the negative and positive values
- In the first iteration, tab[0] is compared to tab[1]

Multidimensional arrays

```
#include <iostream>

int main(){

    int tab2d[5][10];
    int tensor[2][3][7][5];
    // 210 cells

    tab2d[0][0] = 0;
    tab2d[4][9] = 4*9;
}
```

- Any number of dimensions
- Rules (declarations, indexing) such as for one-dimensional tables

Array of structures

```
struct Person {
        int age;
        float salary;
};

Person employee[10];
Person e = employee[0];
e.age = 30;
e.salary = 4000;

employee[1] = e;
e = employee[2];
Employee[3] = employee[2];

e = employee;  // Błąd !!!
```

- The array can be of any type so also "my type"
- Each element of the array is a single Person structure
- employee[x] is a type Person
- employee is NOT Person type !!!

Array of structures

```
struct Person {
        int age;
        float salary;
};
Person e = {30, 4000};

Person employee[10];

employee[2].age = 30;
employee[2].salary = e.salary;

employee.age;  // Error !!!
```

- Each element of the array is a single Person structure
- Structures in the table can be directly addressed (operator .)
- The variable employee is NOT of the Person type, so you can not directly address the structure elements - not known which element they are referring to

why my push was rejected?

because you have conflits...

GIT - rejected push

developer 1

```
> git add source1.cc
> git commit -am “source1”
```

developer 2

GIT - rejected push

developer 1

```
> git add source1.cc
> git commit -am “source1”
```

developer 2

```
> git add source2.cc
> git commit -am “source2”
> git push
```

GIT - rejected push

developer 1

> **git** add source1.cc

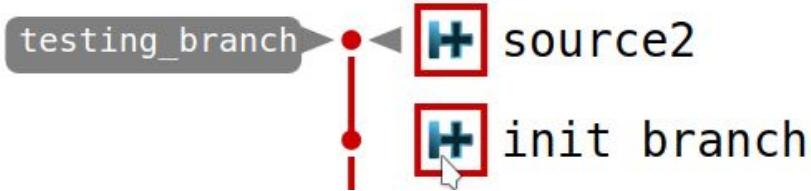
> **git** commit -am “source1”

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push



GIT - rejected push

developer 1

> **git** add source1.cc

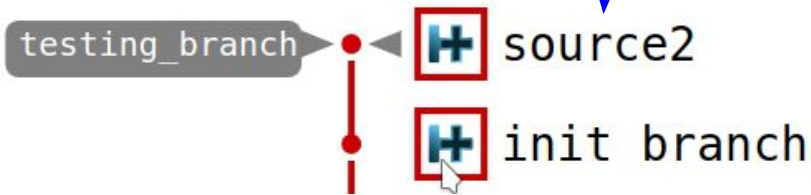
> **git** commit -am “source1”

developer 2

> **git** add source2.cc

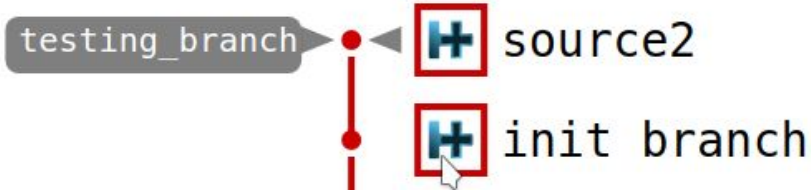
> **git** commit -am “source2”

> **git** push



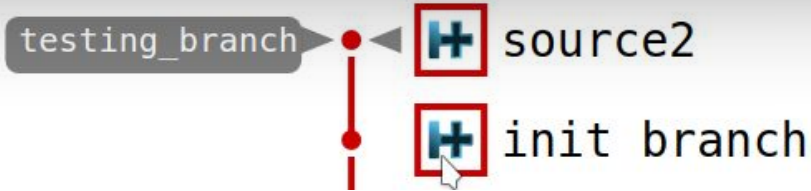
GIT - rejected push

developer 1	developer 2
> **git** add source1.cc	
> **git** commit -am “source1”	
	> **git** add source2.cc
	> **git** commit -am “source2”
	> **git** push
> **git** push	



GIT - rejected push

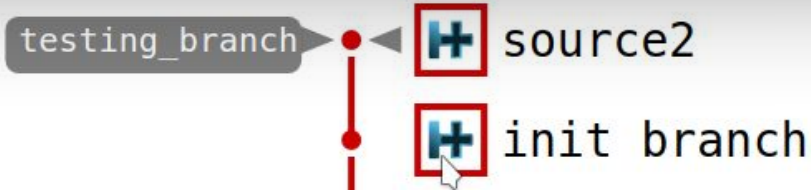
```
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```



GIT - rejected push

```
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

> git
> git
> > g



GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
```

```
  GNU nano 2.9.3                    /home/kwant/git/testing-repo-2016/.git/MERGE_MSG

Merge branch 'testing_branch' of https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016 into testing_branch

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.


^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo
```

GIT - rejected push

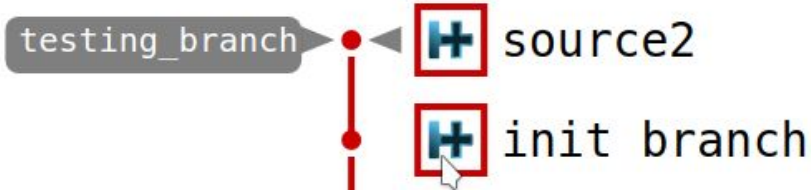
```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
  ebd57ae..dc4ac53  testing_branch -> origin/testing_branch
Merge made by the 'recursive' strategy.
source2.cc | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 source2.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
  ebd57ae..dc4ac53  testing_branch -> origin/testing_branch
Merge made by the 'recursive' strategy.
source2.cc | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 source2.cc
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
-rw-rw-r-- 1 kwant    0 lis  5 23:23 source2.cc
```

GIT - rejected push

developer 1	developer 2
> **git** add source1.cc	
> **git** commit -am “source1”	
	> **git** add source2.cc
	> **git** commit -am “source2”
	> **git** push
~~> **git** push~~	



GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

~~> **git** push~~

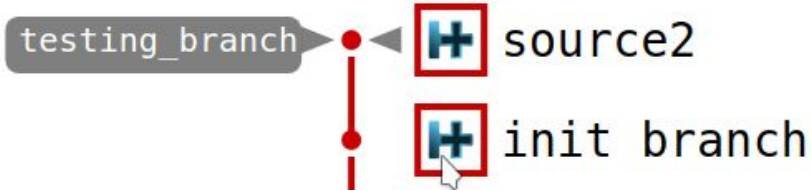
> **git** pull

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push



GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

~~> **git** push~~

> **git** pull

> **git** push

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push

GIT - rejected push

» Conclusion
  – update your local repository frequently
  – (perform **pull** often)
  – “**pull**” before you start working
  – do not be afraid to **merge** your code, get used to it, it is a common practice in git

Thank you