Object Oriented Programming I

Topics today:

- Unary Operators
- Interrupting loops (continue, break)
- Arrays
- Vectors
- File Output

Unary Operators

Basic Unary C++ operators

Operator	Explanations
!	Logical negation. If a is true/false then !a returns false/true, i.e. the negated boolean value of a.
++	Increment operator. a++ (postfix) increases the value of a by 1, but returns the original value of a (!). On the other hand, ++a (prefix) also increases the value of a by 1, but returns the new value of a.
--	Decrement operator. Similar to ++, but decrements by 1.

Example:

```
int main()
{
    int c=5, d=5;
    bool y= true;
    cout << (!y) << endl;
    c++;
    cout << c << endl;
    cout << c++ << endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Problem 1

- Declare and initialize a variable **x** of type integer.
- Use a cout command to check what the return values of **(x++)** and **(++x)** are.
- Check what happens to the value of **x** after the command **++x;** respectively **x++;** has been executed.

Interrupting Loops with **continue** and **break**

continue

```
continue;
```

Explanations:

- **continue** interrupts the current iteration of a **while**- or **for**-loop
- The program immediately proceeds to the next iteration
- Use **continue** to discard some iterations (not most!) of a loop

continue： Example 1

Print the numbers in the range 1,2,…,100 to the screen which are not divisible by 13.

Solution

```
1 for(int i=1;i<=100;i++)
2 {
3       if(i%13==0)
4           contimue;
5       cout << i << endl;
6 }
```

Problem 2 (use of **continue**)

a) Print the pairs (a,b) with a=1,…,10, b=1,…,10, to the screen for which a≠b.

b) Write a program that prints the numbers from 1 to 100 on the screen except the numbers which are multiples of 7 or 11.

break

```
break;
```

Explanations:

- **break** terminates a complete for- or while loop immediately
- If a loop has fulfilled its purpose, but is still running, then **break** it

break: Example

Test if 1001 has any nontrivial divisor. When a divisor is found, print it to the screen, and stop the search at once.

Solution

```
1 for(int d=3;d<1001;d+=2)
2       if(1001%d==0)
3       {
4             cout << d << endl;
5             break;
6        }
```

Problem 3 (use of **break**)

- Write a program that reads 5 double values from the keyboard.
- The program should interrupt the input immediately if the absolute value of the product becomes larger than **1e10**. In this case, an error message should be printed to the screen.
- If 5 values are entered successfully, the product of the numbers entered should be printed on the screen.
- Hint: Absolute value of a double **x**: **fabs(x)** (needs #**include<cmath>**)

Arrays and Vectors

Arrays and Vectors

- Arrays and vectors are both collections of values of the same type
- The number of entries of an array or vector is called its size or its length
- Arrays have a fixed size, vectors can be resized
- The number of entries of an array or vector is called its size or its length
- Vectors have many advantages over arrays
- If possible, use vectors!
- Sometimes we still need arrays, for example, for using certain C-libraries

Arrays

```
type arrayName[size];
```

Explanations:

- The statement above creates an array arrayName which contains size values of type type**.**
- The size must be a constant value (not a variable, except constant variables)
- The i-th entry is accessed by arrayName[i]
- The indices run from 0 to size-1, not from 1 to size!
- Using forbidden indices (negative or greater than size-1) is one of the most common programming errors

Arrays: Example 1

Task:

- Declare an array with 3 entries of type **int**
- Initialize all entries with some values
- Print all entries of the array to screen

Solution

```
1 int A[3];  // integer array A with 3 entries
2 A[0]=4;
3 A[1]=5;
4 A[2]=3;  // all entries initialized now
5 for(int i=0;i<3;i++)
6    cout << A[i] << endl;
7 // for operations with all entries we
8 // usually use a for-loop
```

Arrays: Example 2

Task:

- Check what happens if forbidden indices of arrays are used
- Check what happens if entries of arrays are not initialized

Solution

```
1 int A[3];
2 A[100]=4;        // forbidden index, runtime error!
3 cout << A[100];
4 // no runtime error,  but unpredictable  result
5 cout << A[0];
6 // uninitialized entry, unpredictable  result
```

Problem 4

- Declare an array with 100 entries of type **double**
- Initialize the entries with random numbers
- Compute minimum, maximum and average of the entries
- Print the results to the screen

Problem 5

- Find all prime numbers less than 1,000,000
- Store the primes in an array of length 80,000 with entries of type int
- Use the following fact: an odd positive integer n>1 is a prime if and only if it has no prime divisor p with $2 < p \leq \sqrt{n}$

Initializer Lists for Arrays

```
type arrayName[] = {values};
```

Explanations:

- The statement above creates an array arrayName with values as entries
- values is a comma separated list of values of the given type

Initializer Lists: Example

Task:

- Create an integer array containing the numbers 5, 10, 100, 1000 using an initializer list.
- Print all entries of the array on the screen.

Solution

```
1 int A[] = {5,10,100,1000};
2 for(int i=0;i<4;i++)
3     cout << A[i] << endl;
```

Simplified Iteration over Arrays (C++11)

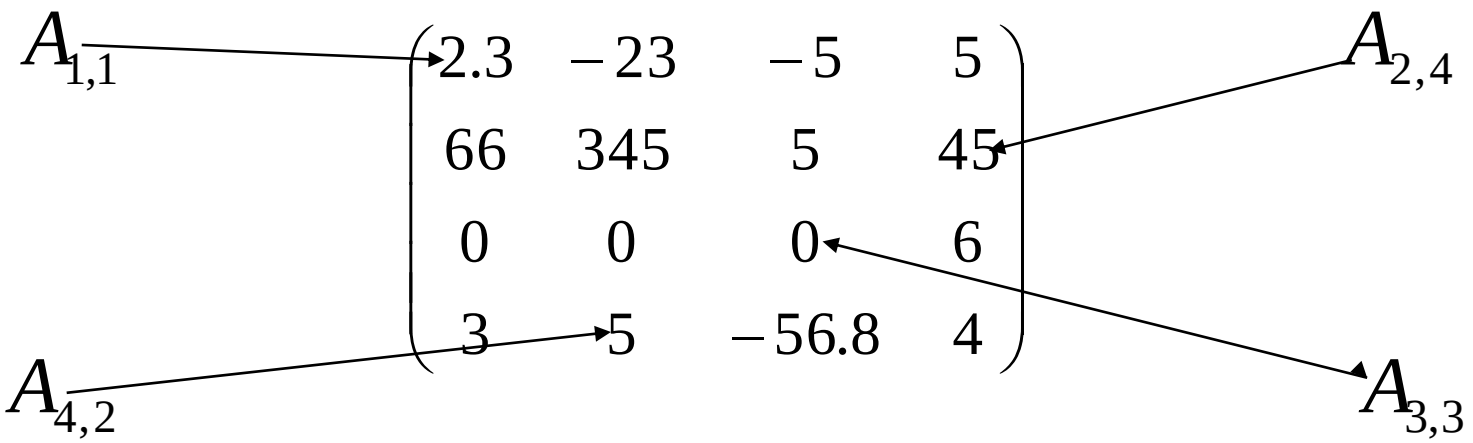
```
for (auto x:arrayName)statement
```

Explanations:

- x will run over the complete array and can be used in the statement
- x can be replaced by any other variable
- Auto is a placeholder type which will be determined automatically: If the array has entries of type **int**, then auto will be **int**

How to Store a Matrix in an Array

The entry in row i and column j of a matrix A is denoted by $A_{i,j}$



How to Store a Matrix in an Array

Matrix M: $\begin{pmatrix} 2.3 & -23 & -5 & 5 \\ 66 & 345 & 5 & 45 \\ 0 & 0 & 0 & 6 \\ 3 & 5 & -56.8 & 4 \end{pmatrix}$ (here $m = n = 4$)

In C++: store a matrix row-wise in an array.

Array A corresponding to M has entries
2.3, -23, -5, 5, 66, 345, 5, 45, 0, 0, 0, 6, 3, 5, -56.8, 4

A[0] corresponds to $M_{1,1}$

A[(i -1)*n + j -1] corresponds to $M_{i,j}$

How to Store a Matrix in an Array

- (mxn) matrix M is stored in an array A of size m*n
- Matrix entry $M_{i,j}$ corresponds to array entry

A[(i-1)*n+j-1]

- We could also use two-dimensional arrays like **int A[2][2]; A[0][0]=234; cout << A[0][0];**
- Not recommended! (tedious, error prone)

Problem 6

Task:

Store the matrix $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ in an array and

print the entries to screen row - wise

Arrays are Dangerous!

```
int A[10];
cout << A[2] << endl;  // error!
```

Result unpredictable since **A[2]** not initialized and thus undefined

---

```
int A[10];
A[10]=10;  // error!
```

Result unpredictable since **A[10]** is no valid array entry

Arrays are Dangerous

```
double A[1000000];  // error!
```

Runtime error, array too big for memory.
Don't use arrays in this way with more than 100,000 elements!

```
int n=5;
double A[n];  // error!
```

Can cause compiler error. Size of an array must be an integer constant.

Arrays of variable size?

```
1 int n;
2 cin >> n;
3 int A[n];
```

- Usually considered a bad mistake – sizes of arrays cannot be non-constant variables
- Some compilers (e.g. Dev-C++) allow it
- Don’t use this – rather use vectors (easiest)
- Dynamic memory allocation for arrays is also possible, but error prone

Vectors

Vectors

- Vectors are an alternative to arrays
- Can be used in the same way as arrays except for declaration
- Useful functions are available for vectors
- Recommendation: when possible, use vectors, not arrays

Vector Declaration

```
vector<type> vectorName(size);
```

```
vector<type> vectorName = {values};
```

Explanations:

- This creates a vector with entries of the given type
- The number of entries of the vector is size
- Initializer lists work with vectors the same way they do with arrays
- If type is a fundamental type, all entries are automatically set to 0
- “(size)” can be omitted. Then an empty vector is created
- After they have been created, vectors can be used simply like arrays,
- There are several useful functions available for vectors
- Need **#include<vector>**

Overview of vector examples / problems

1. Declaration, entry initialization, entry access
2. Use of the **size()** function
3. Common mistake: unintentionally empty vectors
4. Use of vector function **push_back()**
5. Common mistake: unintentional zeros in vector
6. If parenthesis in vector declaration, then they must not be empty
7. Use of vector functions **clear(), push_back(), pop_back(), resize(), size()**

Useful Functions for Vectors

clear()	Removes all elements of the vector
pop_back()	Removes last element of the vector
push_back(elt)	Appends `elt` at the end of the vector
resize(n)	Sets the size of the vector to `n`
size()	Returns the size of the vector

For a vector **v**, these functions are used as **v.clear(), v.pop_back(),** etc.

Vectors: Example 1

Task:

- Declare a vector with 2 entries of type **double**
- Initialize the entries with some values
- Print the entries of the vector to screen

Solution

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main()
7  {
8      vector<double> v(2);
9      v[0] = 3.1415926;
10     v[1] = 2.718281828;
11     cout << v[0] << endl << v[1] << endl;
12
13     system("PAUSE");
14     return EXIT_SUCCESS;
15 }
```

Problem 7

- Declare a vector with 100 entries of type **double**
- Use the **size()** function in the condition of a for-loop which prints all entries to the screen
- Hint: the output should be 100 zeros since the entries are automatically set to 0

Vectors: Example 2

Create a vector which contains 100 random numbers

```
1 vector<int> A;
2 for(int i=0;i<100;i++)
3     A[i]=rand();
```

What is wrong?

After line 1, the vector still is empty. The entries A[i] used in line 3 don’t exist (forbidden indices)

Problem 8

- Declare an empty vector with entries of type **int**
- Use the push_back function to put the numbers 1,…,10 into the vector (in this order)
- Print all entries of the vector to the screen

Vectors: Example 3

Create a vector of length 50 whose entries are random numbers.

```
1 vector<int> A(50);
2 for(int i=0;i<50;i++)
3     A.push_back(rand());
```

What is wrong?

After line 1, the vector already contains 50 entries 0. The random numbers are appended to them. At the end, the vector will have a total of 100 entries.

Vectors: Example 4

Create a vector of length 50 whose entries are random numbers.

```
1 vector<int> A();
2 for(int i=0;i<50;i++)
3     A.push_back(rand());
```

What is wrong?

Line 1 does not create an empty vector. Possible would be **vector<int> A;** or **vector<int> A(0);**
Actually, line 1 is a function declaration! (studied later)

Problem 9

After each of the following operations, print all entries of the vector to the screen:

- Create a vector of length 10 and entries of type **int**
- Append two new values to the end of the vector
- Remove the last element of the vector
- Change the size of the vector to 20
- Remove all entries of the vector

Input and Output

Input and Output: Overview

- Most programs need to obtain data from **files** or from user input through the **keyboard**
- The process of reading such data into a program is called input
- Programs perform computations and produce results
- The results can be printed on the **screen** or written into **files**. This process is called output

File Output: Overview

- If a program creates a large amount of data, we should write this data to a file
- In C++, we create an “output stream” perform this operation
- An output stream is a name which we can use similar to **cout**, but which writes to file instead of the screen
- That’s convenient, since we can use out knowledge out **cout**
- Main thing to know: how to declare an output stream

File Output: Syntax

- File output needs **#include<fstream>**
- An “output stream” is created by

```
ofstream streamName(fileName);
```

- After this, streamName can be used completely similarly to **cout**
- fileName can be an absolute path, e.g. **“C:\\test.txt”** or a path relative to the folder of the executable program, e.g. **“output.txt”**

Recall: `cout`

- Use of **cout** requires **#include<iostream>**
- The value of a variable **x** is printed to the screen by **cout << x;**
- Output can be concatenated, for instance, **cout << x << y << endl;**
- A **string** is printed by **cout << "string";**
- A space is printed by **cout << " ";**
- **cout** works for all fundamental types

Example: File Output

Task:

- Write the numbers 1,…,100 to the file **C:\Lecture3\test.txt** (or similar) using the absolute path.

Solution

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main()
7  {
8      ofstream out("C:\\Lecture5\\test.txt");
9      for(int i=1;i<101;i++)
10         out << i << endl;
11 }
```

Example: File Output

Task:

- Write 100 random numbers to the file using a relative path.

Solution

```
 1 #include <cstdlib>
 2 #include <iostream>
 3 #include <fstream>
 4 using namespace std;
 5
 6 int main()
 7 {
 8     ofstream out("test.txt");
 9     for(int i=1;i<101;i++)
10         out << i << endl;
11 }
```

Remark: the file test.txt will be created in the same directory as the executable program (.exe file)

Problem 10

- Write a program that outputs the numbers 1,2, ... ,100 and their square roots to a file **SquareRoots.txt**
- Try two versions: one with absolute and one with relative path
- The content of the file should look like this:

1 1

2 1.41421

3 1.73205

...

Additional Practice Problems

Problem 11

- Create an integer array A with entries 0, 1, ..., 99 in this order
- Reverse the order of the entries of A by executing 50 swaps (A[0] should be swapped with A[99], A[1] with A[98] etc., etc.).
- Print all entries of A to the screen to check if the swapping worked.

- Hint: A[0]=A[99]; A[99]=A[0]; … will NOT work. (why?)

Problem 12

Write a program that generates a random permutation of 0, 1, ..., 99. One possible method:

- Create an integer array **Perm** of length 100 with entries 0,1, ..., 99 in this order.
- For each **i=99,98,...,1** do the following:
  - Determine a random index **j** in the range **0,...,i**.
    Hint: Use **j=rand()%(i+1);**
  - Swap the entries **Perm[i]** and **Perm[j]**

Problem 13

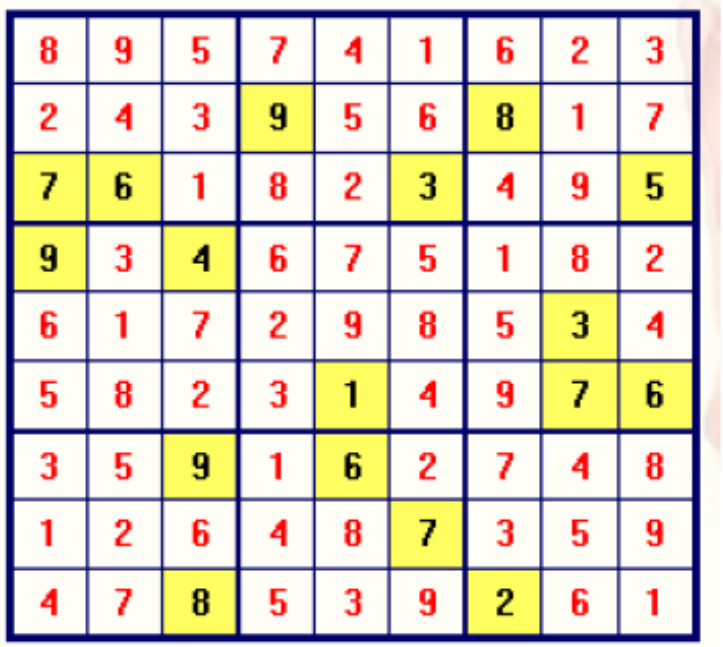
Let Perm[i], i=0,...,n-1, be a permutation of 0,1,...,n-1. We say that Perm has a fixed point if there is an index i with Perm[i]=i.

Write a program that generates 10,000 random permutations of 0, 1, ..., 99, and determines how many of these permutations have no fixed point.

Problem 14

- Write a program that prints the following to the screen:
- The nine rows (i.e. their entries) of the Sudoku square given on the next slide
- The nine columns of this Sudoku square
- The nine 3 × 3 subsquares of this Sudoku square
- Hint: Save the square in an array of size 81 using an initializer list (you can copy and paste the entries from the next slide)

Sudoku



8	9	5	7	4	1	6	2	3
2	4	3	9	5	6	8	1	7
7	6	1	8	2	3	4	9	5
9	3	4	6	7	5	1	8	2
6	1	7	2	9	8	5	3	4
5	8	2	3	1	4	9	7	6
3	5	9	1	6	2	7	4	8
1	2	6	4	8	7	3	5	9
4	7	8	5	3	9	2	6	1

8,9,5,7,4,1,6,2,3,
2,4,3,9,5,6,8,1,7,
7,6,1,8,2,3,4,9,5,
9,3,4,6,7,5,1,8,2,
6,1,7,2,9,8,5,3,4,
5,8,2,3,1,4,9,7,6,
3,5,9,1,6,2,7,4,8,
1,2,6,4,8,7,3,5,9,
4,7,8,5,3,9,2,6,1