
VELDI
KOMPETENS
VELDI

Course outline Week 2

- **Week 1**
  - Introduction
  - First program ”Hello world”
  - Integer Datatype
  - ”if” and ”else” statement
  - IDE
- **Week 2**
  - Datatypes continued
  - Namespace
  - For and while loops
  - Switch and jump statements
  - Arrays
- **Week 3**
  - Functions
  - Pointers
  - Exceptions
  - Lists
- **Week 4**
  - Preprocessor
  - Classes and Objects
  - Constructor and Destructor
  - Class methods
  - Class inheritance
- **Week 5**
  - Dynamic memory
  - File handling
  - Multiple files and headers
  - Libraries

VELDI

Introduction to C++
Part 1
-
More Data types

VELDI

Data types - different types of variables

- String – text
  - Many string types exists, we will focus on std::string here
  - Strings should be written within double quotation marks
  - Example, `std::string car = "hello world!“;`

- Boolean – either true or false
  - Should be lower case true and false
  - Example, `bool car = false;`
  - true is same as writing 1
  - false is same as writing 0
  - Example, `bool car = 1; //same as true`

Data types - different types of variables

- Signed and unsigned – numbers
  - Values can be either signed or unsigned
    - Unsigned can only handle positive values
    - Signed can handle both positive and negative values
  - Default is Signed

- Integer – numbers
  - Possible to use them for calculation
  - Can have bort “short” and “long” as prefix
  - Example, short int car = 7;

Data types - different types of variables

- Char – characters (or numbers)
  - Example, char car = ‘7’;
- Float – decimal numbers
  - Make sure to use dot and not comma
  - Since it is a “digit” it can also be both signed and unsigned
  - Example, float car = 7.3;
  - Can also use “double”, can handle bigger values
  - Example, double car = 7.3;
  - Can be both signed and unsigned, default is Signed
    - Unsigned can only handle positive values
    - Signed can handle both positive and negative values

Data types - different types of variables

- Const declaration
  - If you add the word “const” before a variable it will be declared as a constant and can not be changed

- Auto – uses last type
  - If you have a datatype and want to create a new of the same type
  - Example auto volvo = car //volvo will have same type as car

Data types - different types of variables

- In for example embedded applications it is important to know the size of the variables
- The size of a variable in byte
  - char - 1 byte
  - short int - 2 bytes
  - Int - 4 bytes
  - long - at least 4 bytes, normally 8 bytes
  - Float - 4 bytes
  - Double - 8 bytes
- Be careful to not go above max value (or below min)

```
short int tal = 32767;
std::cout << "value of tal: " << tal << std::endl;      //output 32767
tal++;
std::cout << "new value of tal: " << tal << std::endl;  //output -32768
```

VELDI

Checking the data type size

- If you want to check the size of a variable, you can use the sizeof() function

```
#include <iostream>
#include <list>             //we use list from here

int main()
{
      std::cout << sizeof(short int) << std::endl;
      std::cout << sizeof(long) << std::endl;
      std::cout << sizeof(int) << std::endl;
      std::cout << sizeof(float) << std::endl;
}
```

```
(The output might differ on your computer)
output: 2
        4
        4
        4
```

Type conversion

- C++ supports *implicit conversion*: 1 + 2.3 → 3.3
- And *explicit conversion*: int(2.6) → 2 //Not 3
- Explicit conversion is called *casting*: bool(2.3) → true
- Any value can be implicitly converted to true except for false, none, 0, 0.0 and empty containers like {}, or ""
- You can also convert strings to integer with std::stoi(string)
- And convert integer to strings with std::to_string(integer)

Reading strings

- std::cin >> string //reads data from input until a space appears

  (Note that the direction : “>>” is the opposite to the one we used in cout)
- getline(cin,string) //Reads a line until a enter appears
- If you want “firstname lastname” as input, you need to use the *getline* variant. Otherwise only “firstname” will be given

Validate input

- If you want to validate the input to be only digits you can use the *find_first_not_of()* on the string
- Then it will search the whole string and check so only the specified characters are there, in below case 1234567890

```
#include <iostream>

int main() {
	std::string number;
	std::cout << "Enter a Number: ";
	std::getline (std::cin, number);

	if (number.find_first_not_of ("1234567890") != std::string::npos) {
		std::cout << "Invalid character input : " << number;
	}
	else {
		std::cout << "Thanks, only numbers in: " << number;
	}
}
```

Introduction to C++

Part 2

-

Introduction to Namespace

VELDI

Namespace

- You might have noticed in C++ that “std::cout” sometimes only is a “cout”, the reason behind this is namespace
- “cout” is a part of the standard library and if you add the line “using namespace std;” it will tell the compiler to directly refer to the standard library (std)

```
#include <iostream>
using namespace std;

int main()
{
      cout << "namespace let us call directly to cout, ";
      std::cout << "but can still call from std";
      return 0;
}
```

```
Output:  namespace let us call directly to cout, but can still call from std
```

Create Namespaces

- You can also create own namespace
- Only one entity can exist with a particular name in a particular scope
- Can use more than one namespace
- Namespaces allow us to group named entities that otherwise would have global scope into narrower scopes, giving them namespace scope. This allows organizing the elements of programs into different logical scopes referred to by names
  - Syntax:

```
namespace identifier
{
  named_entities
}
```

```
//Both namespaces have x and y
namespace parallelogram {
      int x,y;
}
namespace rectangle {
      int x,y;
}
rectangle::x = 2;
rectangle::y = 2;
parallelogram::x = 1;
parallelogram::y = 1;
```

VELOCITY

Create Namespaces..

- Below example will create 2 namespaces and us one of them together with std

```
#include <iostream>

namespace ns1
{
	int get_value() { return 4; }
}

namespace ns2
{
	int get_value() { return 5; }
}
//using both ns1 and std
using namespace ns1;
using namespace std;

int main()
{
	cout << get_value() << endl;		//will print 4
	cout << ns2::get_value() << endl;	//will print 5
	return 0;
}
```

- We can not use both “ns1” and “ns2”, because get_value() exists in both and have same name

Introduction to C++

Part 3

-

The for-loop

VELDI

For-loop… what is that?

- **Use the *for-loop* to do something for each element in a sequence**
- **“for (initialization; condition; increase) statement;”**

```
int i;
for (i = 0; i<3; i++)
//init i to 0,as long as i is lower than 3, do i++
{
 std::cout << i << std::endl;
}
```

```
Output: 0
        1
        2
```

Range based for loop

- The range based for-loops are good for iterating through a list or strings
- “for (declaration : range) statement;”

```
std::string str {"Hello!"};
for (char c : str)  //loop through all strings as chars
{
    std::cout << "[" << c << "]";
}
```

```
Output: [H][e][l][l][o][!]
```

Introduction to C++

Part 4

-

The while-loop

VELDI

The while-loop… what is that?

- Use the *while-loop* to check a condition repeatedly
- Works as an alternative to the for-loop
- Here you need to ensure that the condition becomes False at some point if you want to break the loop
- “while (expression) statement”

```
int number = 0;
while (number < 3)
{
    std::cout << number++;    // print, then increase
}
```

```
Output: 012
```

The do-while loop

- The do-while loop works like a “while loop” turned upside down
- The syntax is

  “do statement while (condition);”

```
int number = 0;
do
{
    std::cout << number++;// print, then increase
} while (number < 3);
```

```
Output: 012
```

Use cases for the while-loop

- **While-loops are good if you do not know beforehand how many times you want to iterate something**
- **In the below case something outside the loop will trigger the break**

```
/* Here a function is_button_pressed()
   will report true if button on has been
   pressed */

while (!is_button_pressed()) //wait for button press
{
    //wait for button
}

std::cout << "Button has been pressed";
```

Introduction to C++
Part 5
-
switch statement

VELDI

Switch statement

- The switch statement start by a “switch ()” condition
- Then follows “case” and finally a “default”
- A “break” tells when to break the switch condition

```
switch (expression)
{
        case constant1:
        //do something
        //..
        break;
        case constant2:
        //do something
        //..
        break;
        default:
        //do something
        //..
}
```

VELDI

The switch statement

- **Each case needs a break, otherwise it continues**

```
int x = 1;
switch (x) {
  case 1:        //if 1 we continue to next line
  case 2:        //if 2 we continue to next line
  case 3:        //if 3 we continue to next line
    cout << "x is 1, 2 or 3";
    break;
  default:       //this is like a "else" trap
    cout << "x is not 1, 2 nor 3";
  }
```

```
Output:  x is 1, 2 or 3
```

Switch vs if else

- **Below is a switch example and an equivalent if-else variant**

```
switch (x)
{
case 1:
    y = 4;
    break;
case 2:
    y = 8;
    break;
default:
    y = 4;
}
```

```
if (x == 1)
{
    y = 4;
}
else if (x == 2)
{
    y = 8;
}
else {
    y = 4;
}
```

Introduction to C++
Part 6
-
Jump statements

VELDI

The break statement

- Use the *break*-statement for terminating a loop

```
int guess = 0;
int number = 4;

while (1)     //can have "true" here also
{
     std::cout << "guess number:";
     std::cin >> guess;
     if (number == guess)
     {
          std::cout << "Correct, now we break";
          break;
     }
     else
          std::cout << "Wrong, guess again:";
}
```

- Here we loop until user guess 4

The continue statement

- **Use the *continue*-statement for stopping the current iteration and continue with the next iteration**

```
for (int n=3; n>0; n--)
{
    if (n==2) continue; //skip 2 and continue loop
    std::cout  << n;
}
```

```
Output:  31
```

The goto statement

- **The goto statement requires a destination (“label”) that the goto statement will jump to from the goto command**

```
goto x_test;                    //jump to x_test
 std::cout << "Not valid";     //unreachable
x_test:                         //x_test label
 std::cout << "Hello";
```

```
Output: Hello
```

Introduction to C++

Part 7

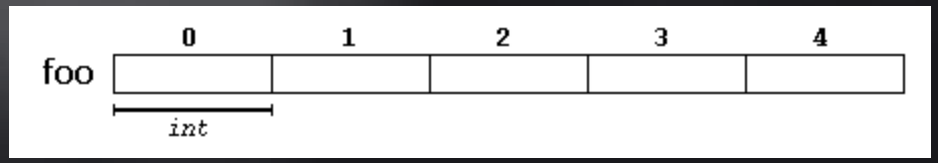
-

Arrays

VELDI

Arrays

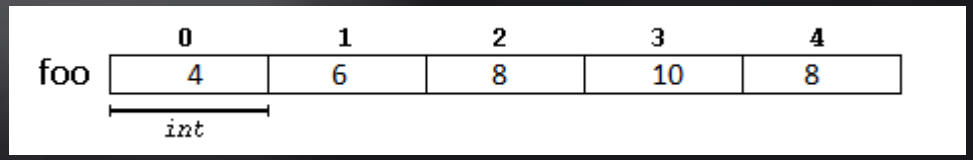
- An array is a series of elements of the same type placed in memory locations that can be individually referenced by adding an index to a unique identifier
- The array syntax uses the “[]” brackets
- Example “int foo[5]”



- Here we allocate the size for 5 integers
- The arrays in c++ are 0 indexed
- To reach the first object use “foo[0]”
- To set value 12 in last index use “foo[4] = 12”

Assign arrays at declaration

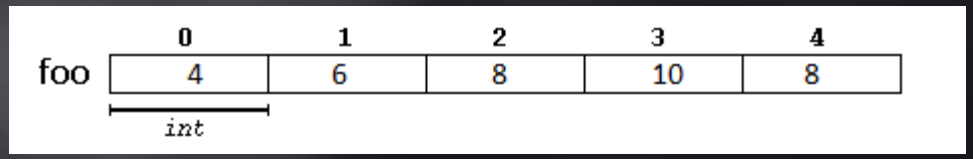
- You can also set the values to an array directly when you declare the array variable to use this use the “{}” brackets after the declaration
- Example “int foo[5] = {4,6,8,10,8};”
- The memory will then look like below:



- It is also possible to skip the “5” digit that tells the size of the array, it will be automatically detected by the numbers we set in the array after
- We can still modify the array with “foo[3]=3”

Const arrays

- If you want the array to be constant and not possible to change you can declare the array as const
- Example “const int foo[5] = {4,6,8,10,8};”
- The memory will then look as before:



- If we try to set a value in the array, we will get an error directly when we compile
- Even pointers to the array where the value are located will stop changing values if it is const declared
- You will also get an error if you try to declare a const array without giving it values. Like “const int[5];” because nothing is declared in the array

VELDI

Printing Arrays

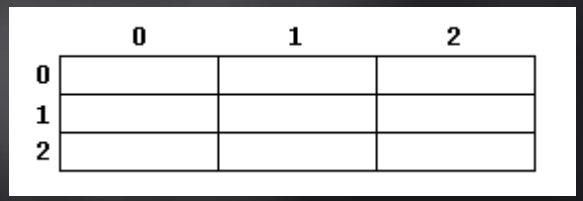
- You can not print the array directly with *cout*. If you do so you will be given the address where the first element are stored. (Same as the address of element I in the array)
- You need to loop through each value in the array individually

```
#include <iostream>
using namespace std;
const int steps_week_days[] = {4210,9231,101,329,9228,428,1283};
int main() {
     for (int i = 0; i < 7; i++){
          cout << steps_week_days[i] << endl;
     }
}
```

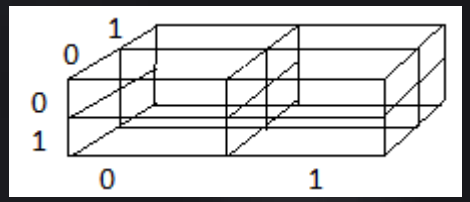
- Or better up, use the “range based” loop

Multidimensional Arrays

- The previously arrays was one dimensional, it is possible to create multidimensional arrays also in C++
- You just add more “[]” in the declaration
- Example: 2-dimensional, size 3x3 “int foo[3][3];”



- Example: 3-dimensional, size 2x2x2 “int foo[2][2][2];”



VELOI

Mixed arrays

- ..is not easy supported in C++. But you can for example mix them with a namespace to group them better together

```
#include <iostream>

namespace step_data
{
    const std::string name_of_days[] =
{"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"
};
    const int steps_week_days[] = {4210,9231,101,329,9228,428,1283};
}

int main() {
    for (int i = 0; i < 7; i++){
        std::cout << step_data::name_of_days[i] << ":" <<
step_data::steps_week_days[i] << std::endl;
    }
}
```

VELDI

Size of an arrays

- The sizeof() command in C++ gets the size of the element
- If you have an array of integer the size of each integer in the list is normally 4 bytes
- If you use sizeof(my Interer array) you will be given a value that are 4 times bigger than the number of elements. To solve that, you can divide it with the size of the first element in the list.

```
#include <iostream>
int main() {
     const int steps [] = {4210,9231,101,329,9228,428,1283};
     int size = (sizeof(steps)/sizeof(steps[0]));
     for (int i = 0; i < size; i++)   {
          std::cout << steps[i] << std::endl;
     }
}
```

VELDI

Introduction to C++

Bonus

-

String operations

VELDI

Find in strings

- There is a function in std::string called find() which you can use to search index in a string. Will return -1 if not found
  - Example: index = my_string.find(string)

```cpp
#include <iostream>
#include <string>

int main()
{
	std::string my_string = "Volvo has the coolest cars";
	int position = my_string.find("coolest");// Found at index (14)
	std::cout << position << std::endl;
	position = my_string.find("Tesla");// Not found (-1)
	std::cout << position << std::endl;
}
```

```
Output: 14
        -1
```

String replacement

- There is a function in std::string called *replace()* which you can use for replacing something in your string
- Write what you want to replace, together with the position and length, within the parentheses
  - Example: my_string.replace(position, length, string)

```
#include <iostream>
#include <string>

int main()
{
	std::string my_string = "Tesla is the best car in the world";
	my_string.replace(0,5, "Volvo");
	std::cout << my_string;
}
```

```
Output: Volvo is the best car in the world
```

String replacement continued (single)

- If you have dynamic string and don´t know the index you can combine “replace” with “find” and “length”

```
#include <iostream>
#include <string>

int main()
{
     std::string my_string = "Sam says Tesla is best";
     std::string repl = "Tesla";
     my_string.replace(my_string.find(repl),repl.length(), "Volvo");
     std::cout << my_string;
}
```

```
Output: Sam says Volvo is best
```

Replacing multiple parts

- **To replace or remove all occurrence of a string we need to loop the string**

```
#include <iostream>
#include <string>

int main()
{
	std::string my_txt = "Sam says Tesla is best, I also like Tesla";
	std::string re_st = "Tesla";

	while (-1 != (int) my_txt.find(re_st))	//loop until -1 (not found)
	{
		my_txt.replace(my_txt.find(re_st),re_st.length(),"Volvo");
	}

	std::cout << my_txt << std::endl;
}
```

```
Output:  Sam says Volvo is best, I also like Volvo
```

Modify the casing of a string

- Use *toupper()*-function for setting a char to uppercase
- Use tolower*()*-function for setting a char to lowercase
- To update a string loop through all characters

```
#include <iostream>

int main()
{
     std::string my_txt = "Sam says Tesla is best, I like Volvo";

     for (int i=0; i < (int)my_txt.length(); i++)
     {
          my_txt[i]=toupper(my_txt[i]);    //or tolower for lower case
     }
     std::cout << my_txt;
}
```

Output: SAM SAYS TESLA IS BEST, I LIKE VOLVO


veldikompetens.se
Thank you!
VELDI KOMPETENS