
VELDI
KOMPETENS
VELDI

Course outline week 3

- **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

-

Functions

VELDI

Functions… what is that?

- A function is a block of code that you can reuse as many times as you want
- All C++ programs consists of at least one function. (The “main()” function is also function)
- In C++, a function is a group of statements that is given a name, which can be called from some point of the program
- The most common syntax is:

  type name (parameter1, parameter2, ...) { statements }

- Functions are good to use to give a better structure of a program or split code that are used frequency into a separate function.

Example of a simple function

- Here we create a simple function that prints “Hello world”

```
#include <iostream>

void hello()
{
    std::cout << "Hello world";
}
```

- The function will not return any value since we have “void” as return type
- We have nothing inside the “()”, which means that this function does not have any arguments

How to call a function

- Calling the function needs to be done after the function has been created (or have a correct prototype before)
- To use a function, you shall call the function name
- In this case we will call the newly created hello function, with the following command “hello()”
- The “()” here says that we will call a function without any argument

```
int main()
{
    hello();// will call function hello()
    return 0;
}
```

Variable scope functions

- Variables created inside functions are called *local variables* and are not accessible outside the function
- Variables outside functions are called *global variables*
- Global variables can be modified inside a function
- Need to be careful if you have variable with the same name inside a function and the same name on another global variable
- If the variable is declared with the same name inside the function, it will be handled as local variable in that function

Variable scope continue..

- **Notice below example and that “int x;” inside the function makes the x variable as local in function a()**

```cpp
#include <iostream>

int x = 1;   //global

int a() {
    x = 2;//set global x
    return x + 2;
}

int main() {
    std::cout << a();
    std::cout << x;
}
```

```
Output: 42
```

```cpp
#include <iostream>

int x = 1;//global

int a() {
    int x;//x now local
    x = 2;//set local x
    return x + 2;
}

int main() {
    std::cout << a();
    std::cout << x;
}
```

```
Output: 41
```

Arguments to functions

- Functions can have input arguments which can be used by the function
- Below example will have an integer as argument

```
#include <iostream>

void hello(int number) {
     std::cout << "Hello world:" << number;
}

int main() {
     hello(2);   //will call hello function with argument 2
}
```

```
Output:  Hello world:2
```

Return value to function

- Sometimes you want the function to return a value
- Returning something is done with the *return*-keyword
- Below example will return number 4
- Note that we need to have “int” as type now to return 4

```
#include <iostream>

int duble()
{   //int here describes return value type
     return 4;
}

int main() {
     std::cout << duble();  //will return and print 4 as value
}
```

```
Output:  4
```

Default argument values

- Functions can have default values for the arguments
- If the user does not input anything, it will use the default-value instead

```
#include <iostream>

int defau(int i = 10) {   //default value is 10
     return i;
}

int main() {
     std::cout << defau() << std::endl;   //will return default 10
     std::cout << defau(2) << std::endl;  //will return 2
}
```

```
Output: 10
         2
```

Recursive function

- Sometimes you want a function that repeats itself until a specific state is given
- Need to be very careful here to not end up in a infinite loop when using recursive functions
- Need to have a check in the function so you won´t end up in a loop like below code:

```
void infinite_bad_loop() {
    infinite_bad_loop();    //This is bad, we will call
                            //Outself everytime we call
                            //this function, and loop
                            //forever.
}

int main() {
    infinite_bad_loop();    //Will never end….
}
```

Recursive function continue..

- Here we create a function that adds “0” until we have reached a length of 20 characters.

```
std::string zero_adder(std::string input) {
    if (input.length() < 20)
        return zero_adder(input+ "0"); //recall and add 0
    else
        return input;
}
```

- Will be called from main with the following code

```
std::cout << zero_adder("Hello");
```

```
Output: Hello00000000000000
```

References as argument

- Sometimes you want to return both a status and all or many of the arguments in a function
- To solve this, you can add a reference variable as an argument to a function instead
- The reference will be like a pointer that points to the place where the value are stored and update it
- If you want to pass an argument and update the value in the function you can use a reference, references will be used by adding the “&” before the variable like “int &car_number;”

Example using references as argument

- Here we create a duplicate function that uses a reference to an int as argument

```
void duplicate(int &a) {
    a=a*2; //double a
}
```

- You can use below code co call that function

```
int a = 2;
duplicate(a);
std::cout << a;
```

```
Output: 4
```

- Will explain more about this in pointer chapter later

Main function arguments

- The main function has a default return statement if you miss it (returns 0)
- If you want to use arguments to main it has built in support for this
  - int main(int argc, char* argv[])
    - argc = number of arguments
    - argv = arguments as char strings

```
#include <iostream>

int main(int argc, char* argv[])
{
     for (int x=0; x < argc; x++) {
          std::cout << argv[x] << std::endl;
     }
}
```

- The arguments will then appear as output when running this

Main function arguments example

- Below is a calculator that needs 3 arguments, example “2 + 2” then it will print the result of 2+2

```
#include <iostream>
int main(int argc, char *argv[]) {
     if (argc == 4) {
          std::cout << argv[1] << argv[2] << argv[3] << "=";
          if (argv[2][0] == '-’)
               std::cout << (std::stoi(argv[1]) - std::stoi(argv[3]));
          else if (argv[2][0] == '+’)
               std::cout << (std::stoi(argv[1]) + std::stoi(argv[3]));
          else if (argv[2][0] == '*’)
               std::cout << (std::stoi(argv[1]) * std::stoi(argv[3]));
          else if (argv[2][0] == '/’)
               std::cout << (std::stoi(argv[1]) / std::stoi(argv[3]));
     }
     else
          std::cout << "Enter <value> <operand(-+*/)> value>";
}
```

- Arguments is more like a bonus in this course but nice to know

What is a prototype ?

- You might have noticed that the compiler sometimes complains something like “no previous declaration for…”
- The reason for this is that the compiler want to have a prototype of the function in the beginning of the file
- A prototype is just the function name, type and arguments of the function that are placed at the beginning of the file to help the compiler to find the functions (or in header file)

```
/* prototype */
void duplicate(int &a);
```

- Some compiler might not even give a warning about this

Prototype example

```
/* below line is a prototype */
void duplicate(int &a);

/* this is the function duplicate */
void duplicate(int &a) {
    a*=2;
}

/* function main shall not have a prototype */
int main() {
    int b=0;
    duplicate(b);
}
```

Introduction to C++
Part 2
-
Introduction Pointers

VELDI

Pointers

- The Pointers in C++ is like the name says, a pointer that points to something
- The data on a memory is placed in bytes placed in the memory. To find where data are located, we use pointers
- The address of a variable is the location in the memory where the operative system store its data
- So far in this course we have used a datatype and a variable name that has been given a specific location in the memory referred by the variable name
- A pointer can for example be used to update or read data in other variables or arrays
- A pointer has the “*” operand like “int *foo;”
- To get a “known” state you can set pointer to “NULL” at init

Address-of operator

- To get the address of a variable in C++ we use the magic keyword “&” (“address-of operator”)

```
#include <iostream>
int main()
{
    int a=12;           //variable with 12 as data
    int *b=NULL;        //create pointer "b" as NULL
    b = &a;             //pointer "b" points to "address-of" "a"
    std::cout << b;     //will print the pointer address
}
```

- This will explain more about the reference in functions previously in this chapter

Dereference operator

- To get the value that are stored at a specific pointer address, the “*” shall be used
- Can be read as “value pointed to by”

```
#include <iostream>

int main()
{
    int a=12;     //variable with 12 as data
    int *b=NULL;  //pointer b

    b = &a;       //pointer b points to "address-of" "a"
    *b=32;        //"value pointed to by "b" = 32"
    //the above 2 lines will result in that the variable "a" now will be 32
    std::cout << a << std::endl;
}
```

Size of Pointers

- We have previously shown that the size of different datatypes differs depending on how much data it stores
- The pointer will have the same size even if it is a char* pointer or a long* pointer
- The reason for this is that all pointers only stores the address to the location where data are located
- Size of pointer may differ from hardware and operating system

```
int a;
char b;
double c;
std::cout << sizeof(a) << std::endl;//4
std::cout << sizeof(b) << std::endl;//1
std::cout << sizeof(c) << std::endl;//8
int *aa;
char *bb;
double *cc;
std::cout << sizeof(aa) << std::endl;//8 (depends on pointer size)
std::cout << sizeof(bb) << std::endl;//8 (depends on pointer size)
std::cout << sizeof(cc) << std::endl;//8 (depends on pointer size)
```

VELDI

Pointers and arrays

- Pointers are good to have when using arrays since arrays contains multiple data
- A pointer will then point to data in the array
- To modify the pointer, we can use some alternatives:
  - The pointer++, ++pointer, pointer-- or --pointer
  - (pointer + X) or (pointer –X) (where X is digit)

```
#include <iostream>
using namespace std;
int main()
{
	int my_array[]={2,4,8};
	int *my_array_pointer = NULL;			//point to NULL
	my_array_pointer = my_array;			//point to first element
	cout << *my_array_pointer << endl;		//will print 2
	my_array_pointer++;				//move pointer 1 step forward
	cout << *my_array_pointer << endl;		//will print 4
	my_array_pointer= (my_array_pointer-1);		//move pointer 1 step back
	cout << *my_array_pointer << endl;		//will print 2
	return 0;
}
```

Array Pointer example

```
#include <iostream>
int main()
{
	int numbers[5];
	int *p;
	p = numbers;
	*p = 10;
	p++;
	*p = 20;
	p = &numbers[2];
	*p = 30;
	p = numbers + 3;
	*p = 40;
	p = numbers;
	*(p + 4) = 50;
	for (int n = 0; n < 5; n++)
		std::cout << numbers[n] << ", ";
	return 0;
}
```

```
Output:	10,20,30,40,50,
```

VELDI

Introduction to C++
Part 3
-
Handling exceptions

VELDI

Handling exceptions

- There are a lot of things in programming that can happen that the programmer have not thought of
- If these exceptions are not handled properly, the program will crash (which we want to avoid!)
- Luckily, C++ has something that can help us out which is called *try-catch blocks*

The try-catch block

- Put the code that has potential failures in the try-block
- If something goes wrong, the program jumps to the catch-block
- C++ can use built in exceptions, or create own by the throw keyword

```
#include <iostream>

int main()
{
	try
	{
		// Block of code to try
		throw exception; // Throw an exception when a problem arise
	}
	catch () //catch the Exception
	{
		// Block of code to handle errors
	}

	return 0;
}
```

Raising own exceptions

- Sometimes you need to raise exceptions yourself
- Use the throw-keyword

```
#include <iostream>
int main() {
	signed int number;
	std::cout << "Enter number (1 to 9):";
	try {
		std::cin >> number;
		if (number > 9) throw 10; //exception 10 means 10 or higher
		if (number < 1) throw 0;  //exception 0 means 0 or lower
	} catch (int x){                //x has the number from “throw” above
		if (10 == x)
			std::cout << "Too high exception raised";
		else if (0 == x)
			std::cout << "Too low exception raised";
	else
		std::cout << "Unknown exception raised";
	}
	return 0;
}
```

- Try to enter values above, both in range or out of range

Introduction to C++
Part 4
-
Introduction to lists

VELDI

How to create a list

- Here we will use std::list
- Accessed by “<list>” as include
- It may be empty
- Can be pointed out by an iterator

```cpp
#include <list>

int main()
{
    std::list<int> even_numbers = {2, 4, 6, 8};
    std::list<int> initially_empty;
}
```

Adding to a list

- Use the *push_back()*-function for adding something to a list at back
- Use the push_front() function for adding something to the top of the list
- A list can contain elements of any type

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

int main()
{
	std::list<std::string> mylist;
	mylist.push_back("Winter");
	mylist.push_back("Spring");
	mylist.push_front("Summer");
	mylist.push_back("Fall");
	for (std::string n : mylist) {           //easy way to loop through
		std::cout << n << " " ;
	}
}
```

```
Output: Summer Winter Spring Fall
```

Sort lists

- If you want to sort the list, you can use the “sort()”
- you want to reverse sort the list, you can use “reverse()”

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

int main()
{
      std::list<std::string> mylist;
      mylist.push_back("C");
      mylist.push_back("A");
      mylist.push_back("B");
      mylist.sort();                    //built in support for sorting list
      for (std::string n : mylist) {
            std::cout << n << " " ;
      }
}
```

```
Output:  A B C
```

Remove an element of a list

- **You can remove an element from a list by using the *remove(), pop_back() and pop_front()* functions**

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

int main()
{
      std::list<std::string> mylist;
      mylist.push_back("Winter");
      mylist.push_back("Spring");
      mylist.push_back("Spring");
      mylist.push_back("Fall");
      mylist.remove("Spring");          //"Spring" will be removed
      mylist.pop_back();                //removes last object "Fall"
      for (auto n : mylist) {           //uses auto this time
            std::cout << n << " " ;
      }
}
```

```
Output:  Winter
```

Introduction to C++

Part 5

-

How to create a menu

How to create a menu

- Creating a menu can be done by using a string together with the += operator and the line break character \n
- Use \n in your string to make a line break

```
#include <string>
#include <iostream>
int main() {
     std::string menu_string =  "--- Select ---\n";
                 menu_string += "1: Add\n";
                 menu_string += "2: View\n";
                 menu_string += "3: Exit\n";
     std::cout << menu_string;
     return 0 ;
}
```

```
Output:  --- Select ---
         1: Add
         2: View
         3: Exit
```

Making the menu work

- You can use a while loop to keep the program alive until the user wants to exit the program

```
#include <string>
#include <iostream>
int main() {
char ch=0;
  std::string menu_string=  "--- Select ---\n" //Can also add string like this
                            "1: Add\n"
                            "2: View\n"
                            "3: Exit\n";
  do {
       std::cout << menu_string;
       //if ('1' == ch) do Add, Note that we use '1' and not 1 when comparing
       //if ('2' == ch) do View
  } while ((ch=std::cin.get()) != '3');
  return 0 ;
}
```

Introduction to C++

Bonus

-

Iterate through lists

VELDI

Get element of a list

- The std::list does not have the traditional easy way to get specific objects in the list like other high level programming languages
- When the existing list functions no longer can help you, you might need to use an iterator to loop through the list
- An iterator is any object that, pointing to some element in a range of elements
- You can read more online, how the “iterator” works in C++

Accessing an element of a list

- When we increase the iterator, we step to next element in the list

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

int main()
{
	std::list<std::string> mylist;
	mylist.push_back("Winter");
	mylist.push_back("Spring");
	mylist.push_back("Fall");
	//instead of "std::list<std::string>::iterator ", you can use "auto li"
	std::list<std::string>::iterator li = mylist.begin();

	std::cout << *(li) << std::endl;        //pointer
	std::cout << *(li) << std::endl;        //since pointer not increased => same
	li++;                                   //increase iterator => next value in list
	std::cout << *(li) << std::endl;        //we have increased to next value above
}
```

```
Output:  Winter
         Winter
         Spring
```

Introduction to C++

Bonus

-

Bonus: Built-in exceptions

VELDI

Built-in exceptions

- C++ has built in exceptions, that you need to enable to use
- Some traditional exceptions will not be handles in the built-in exception handling, like the divide by zero. (Needs to be handled by user)
- We will go through an example of built-in exception when using an integer as “cin” input and the user reply with a string

Bonus: Built in exceptions

- Below code will set the “failbit” exception in “cin” when an invalid input appears
- Try to enter “D” or another character as input in below code and the “failbit” exception shall appear

```
#include <iostream>
int main() {
	signed int number;
	//This is a tricky part, you need to enable the fall bit in exceptions
	std::cin.exceptions(std::ios_base::failbit);
	std::cout << "Enter number";
	try {
		std::cin >> number;
		std::cout << "You entered number " << number;
	} catch (...){//"(...)" = catch all errors,
		std::cout << "Invalid number. ";
	}
	return 0;
}
```


veldikompetens.se
Thank you!
VELDI KOMPETENS