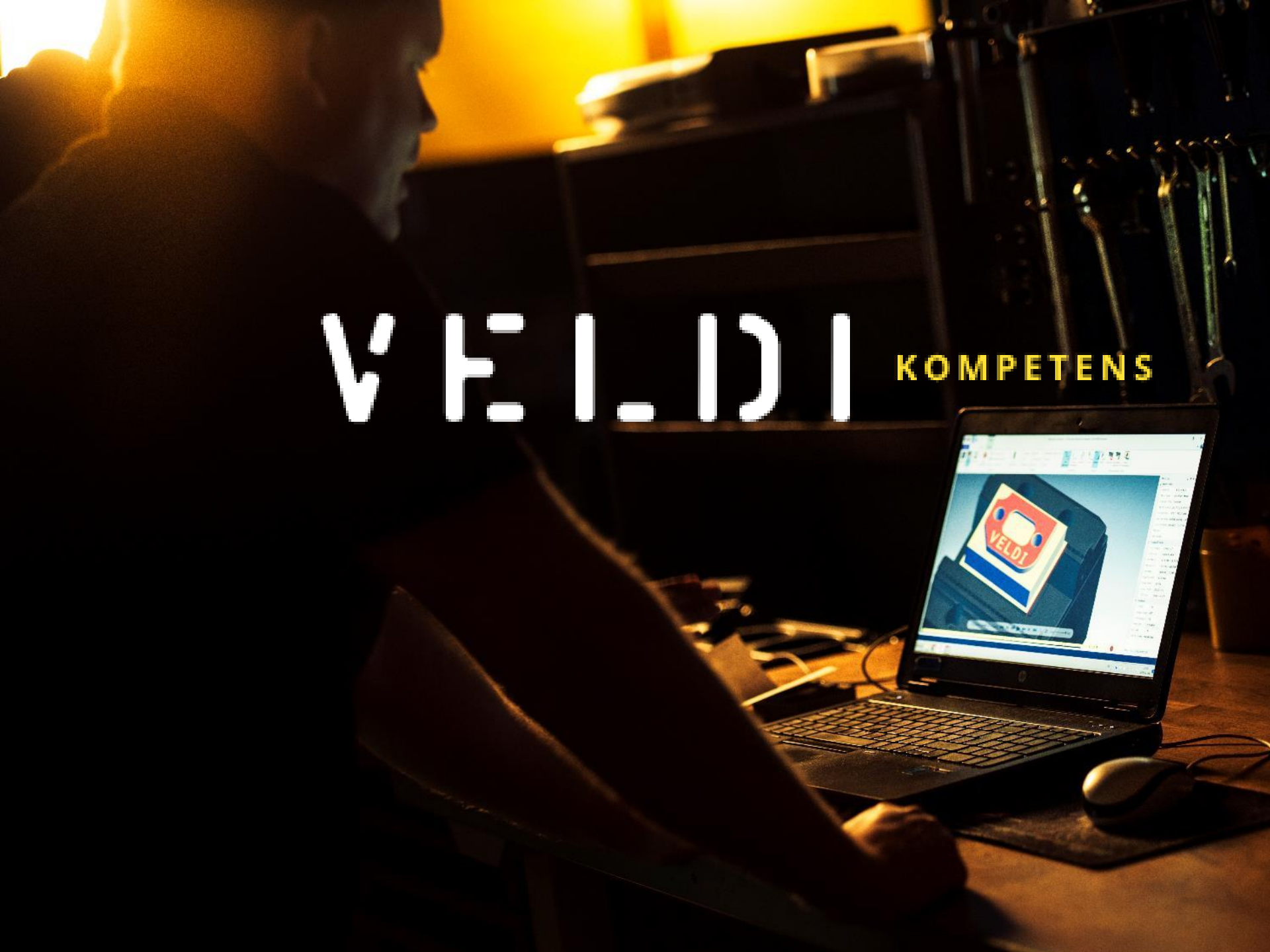

VELDI
KOMPETENS
VELDI

Course outline week 4

- **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
-
Introduction Preprocessor

VELDI

Preprocessor… what is that?

- Before the compiler starts to compile the program, it will run a preprocessor part with the code. Starts with a “#”
- Example of preprocessor directives is
  - #include
  - #pragma
  - #if, #ifdef, #ifndef, #else, #elif and #endif
  - #define, #undef
- Since preprocessor will be parsed before the compiler starts to compile the code, we can not check a “#if” against for example a variable
- We will here focus on a few preprocessor directives

#include

- The #include preprocessor will replace the include line with the code placed in the file inside the “<>”
  - #include <iostream.h>
- If you open the iostream file you will notice that it contains the cin and cout
- Keep in mind to only include the files you need, not just add something at the top of the code to make it easy to code
- Include files can change standard defines without your notice in some include files
  - One file might have “#define PI 3.14”
  - Another file might have “#define PI 3.1”

#ifdef, #elif and #endif

- Sometimes part of code might differ between for example Linux and Windows
- Then you can use the included defines __linux__ and _WIN32 to do low level part and create common macro

```
#ifdef __linux__
    //linux code goes here
#elif _WIN32
    // windows code goes here
#else
    // something else, maybe check for MAC also ?
#endif
```

#define examples

- You can define both values and macros
- Example: Define a constant as PI
  - #define PI 3.14
    - Now you can set float a = PI;
- You can define a new way marco of of “std::cout <<“
  - #define skriv(A) std::cout<<(A)
    - Now you can write, “skriv(“Hello world”); as output
- A complete “ugly” hello world C++ program might look as just a single “B”,see below

```
#include <iostream>
#define skriv(A)std::cout<<(A)
#define B int main(){skriv("Hello world");return 0;}
B
```

- So, use #define with care ☺

Introduction to C++

Part 2

-

Classes and objects

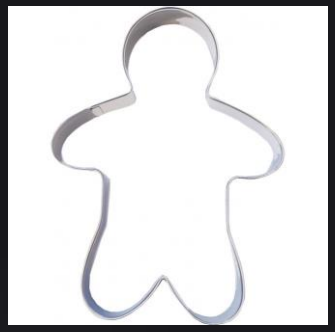
VELDI

Classes… what is that?

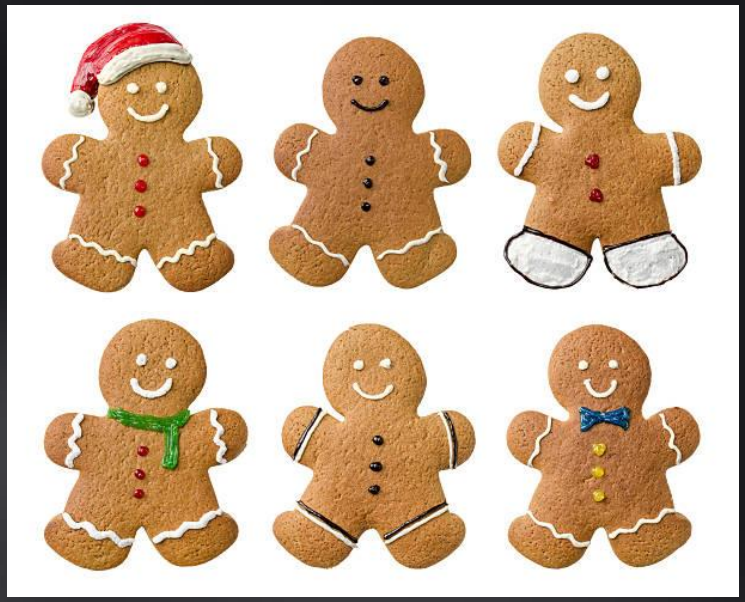
- In C++, and other object-oriented languages, you can create *classes*
- A class is a template for an *object*
- An object is a variable that has been created using the template
- When you create an object, you say that you *instantiate* the object and that the object is an *instance* of the class
- Methods are functions that belongs to the class

Illustration of classes and objects

- A class is like a cookie cutter, it works as a template for a cookie



- Objects are like the cookies, they have the same structure as the class, but they can have different attributes



Other examples of classes and objects

class	objects
Fruit	Apple Banana Mango

class	objects
Car	Volvo Audi Toyota

Classes – why should we use them?

- Like functions, we use classes to avoid repeating ourselves when writing code – humans are still intelligent!
- Classes are excellent for making the code look more elegant and making it easier to read
- Classes also makes it easier to reuse the code
- It bunches things that belong together in one place
- We use classes for bunching variables and functions together when they belong to each other

How to create a class

- Creating a class (the template) is done by using the *class*-keyword
- A class can contain one or more variables and functions
- A member of a class can be either:
  - **Private** - members of a class are accessible only from within other members of the same class (or from their "friends")
  - **Protected** - members are accessible from other members of the same class (or from their "friends"), but also from members of their derived classes
  - **Public** - members are accessible from anywhere where the object is visible
- Default declaration is Private in a Class

VELOI

How to create a class..

- **Classes are created with the following structure:**

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} object_names;              //object_name not required
```

```
class Hello {  //Example Class containing an int
public:
    int hej;
}; //Note no object_name here
```

Creating 2 simple objects

- Creating an *object* of a class is easy, just write “class” and the name after it, like declare variables
- Setting the class variables is also easy, just write the object together with a dot and the variable name

```
class Hello {//Class containing an int
public:
     int hej;
} obj_name;

int main() {
     obj_name.hej = 1;  //Set the public variable hej in obj_name

     Hello hoj;         //Creates a new object hej from Class Hello
     hoj.hej = 0;       //Set the public variable hej in hoj
}
```

- So, *hoj* is an *object* and it contains variable *hej and obj_name is another object with variable hej*

Introduction to C++

Part 3

-

Constructor and Destructor

VELDI

Constructor

- Sometimes you want things to be created directly when the class are being created. This can be done with a constructor
- The constructor will be executed directly when the class are being created.
- The constructor are referred by the same name as the class
- You do not need to create a constructor in a class
- You can have many overloaded constructors in the same class depending how the class are being created. Like:
  - Rectangle rect(1,2)
  - Rectangle rect

Constructor example outside class

- Here you can se an example of a constructor that sets values for width and height, the constructor is here placed outside the class and needs the “Rectangle::” prefix

```
#include <iostream>

class Rectangle {
private:
      int height;
      int width;
public:
      Rectangle();
};

Rectangle::Rectangle() {
      height=1;
      width=2;
}

int main () {
      Rectangle rect2;
      return 0;
}
```

Constructor example inside class

- Here you can se an example of a constructor that sets values for width and height, the constructor is here inside the class and declared there

```
#include <iostream>

class Rectangle {
private:
      int height;
      int width;
public:
      Rectangle() {
            height=1;
            width=2;
      }
};

int main () {
  Rectangle rect2;
  return 0;
}
```

Constructor and members

- **We have said that we shall not leave any variable uninitialized before in this course. When you use both initial value and constructor. The constructor will override the initial value**

```
#include <iostream>

class Rectangle {
private:
	int height=0;		//initially declared as 0
	int width=0;
public:
	Rectangle() {
		height=1;	//constructor will overwrite the initially value on members
		width=2;
	}
	void print() {
		std::cout << height << " " << width;
	}
};

int main () {
  Rectangle rect;
  rect.print();			// will print out 1 2
  return 0;
}
```

Two different Constructors

- **Here you can se an example of 2 constructors and how to use them. Note that you do not have the “()” if you do not have any arguments to the constructor**

```
#include <iostream>

class Rectangle {
public:
    Rectangle(int x, int y){std::cout << "Your value:" << x << " " << y << " gets area:" <<
x*y << std::endl;};   //uses argument (x y)
    Rectangle() {std::cout << "Using default value: 2,2 gets area 4";};     //No arguments
};

int main () {
  Rectangle rect1(12,12);
  Rectangle rect2;    //NOTE here that we can NOT have () at end
  return 0;
}
```

```
Output:  Your value:12 12 gets area:144
         Using default value: 2,2 gets area 4
```

Constructor initialization

- **A constructor can be initialized in many ways, see below example. The constructor will be called in all cases**

```
#include <iostream>

class Circle {
    double radius=0;
  public:
    Circle(double r) { radius = r; }
    double circum() {return 2*radius*3.14159265;}
};

int main () {
  Circle foo (10.0);   // functional form
  Circle bar = 10.0;   // assignment init.
  Circle baz {10.0};   // uniform init.
  Circle qux = {10.0}; // POD-like


  std::cout <<  foo.circum() << " " <<  bar.circum() << " " <<  baz.circum() << " " <<  qux.circum() <<
std::endl;
  return 0;
}
```

```
Output:  62.8319 62.8319 62.8319 62.8319
```

Destructor

- Sometimes you want things to be cleaned up after the class are not used any more
- The destructor will be executed directly after the class no longer are used
- If you have dynamic memory allocation in a constructor, it might be good to place the cleaning of that memory in the destructor
- The destructor looks like the constructor but with a “~” at the beginning
- You do not need to create a destructor when creating a class

Destructor example

```
#include <iostream>

class Rectangle {
public:
    Rectangle() {std::cout << "constructor" << std::endl;};
    ~Rectangle(){std::cout << "destructor";};
};

int main () {
  Rectangle rect2; //NOTE here that we can NOT have () at end
  std::cout << "Program will here end," << std::endl;
  return 0;
}
```

```
Output: constructor
        Program will here end,
        destructor
```

VELDI

Introduction to C++

Part 4

-

Class methods

VELDI

Class methods… what is that?

- Classes can have functions and they are called *methods*
- Methods can be declared within the class or outside the class with the Class name followed by the "::"

```
#include <iostream>

class Car {
      std::string brand,color;
  public:
      Car(std::string b,std::string c){brand=b;color=c;}; //inside class declaration of constructor
      void print_info();                                  //method, will be declared outside
};

void Car::print_info() {                                  //uses Class and :: since outside class
      std::cout << brand << " " << color;
}

int main () {
      Car my_car("Volvo","Green");
      my_car.print_info();
      return 0;
}
```

- So, *my_volvo* is one *object* and it contains three things: *brand*, *color* and the method *print_info()*

VELOX

Class methods… public and private

- Below example will illustrate that the public variables and methods can be reached but not the private

```
class Box {
   public://can be accessed outside the class
      void setWidth(int w) {width = w;};
      int height;
   private://can not be accessed outside the class
      int width;
};

int main()
{
       Box my_box;
       /* below is the only way to set width and height in the class */
       my_box.setWidth(12);
       my_box.height = 2;
}
```

- You can set width directly but need to use the setWidth to set the width

Example of how to use class methods

```cpp
#include <iostream>

class Car {
      std::string brand,color;
  public:
      Car(std::string b,std::string c) {brand=b;color=c;};    //inside class declaration of
constructor
      void print_info();
};

void Car::print_info() {                                  //uses Class and :: since outside class
      std::cout << "The " << brand << " is " << color << std::endl;
}

int main () {
      Car my_volvo("Volvo","Green");
      my_volvo.print_info();
      Car my_tesla("Tesla","Blue");
      my_tesla.print_info();
      return 0;
}
```

Output:

The Volvo is Green
The Tesla is Blue

How to create a class with methods

- Here we create a more advanced class to calculate area

```
#include <iostream>

class Rectangle {
private:                    //below are private within Class
    int width, height;
public:                     //below methods have public access
    void set_height(int);   //will be declared later
    void set_width(int);
    int get_area();
};
//Here we declare private get_area, "Rectangle::" means "in" class Rectangle
int Rectangle::get_area() {
return width*height;        //height and width are from Class Rectangle
}
//Here we declare private set_height, "Rectangle::" means "in" class Rectangle
void Rectangle::set_height(int x) {
  height = x;               //height is from Class Rectangle
}
//Here we declare private set_wdth, "Rectangle::" means "in" class Rectangle
void Rectangle::set_width(int x) {
  width = x;                //width is from Class Rectangle
}
int main () {
  Rectangle rect;           //Create object rect from class Rectangle
  rect.set_height(2);       //Set Public member set_height
  rect.set_width(4);        //Set Public member set_width
  std::cout << "area: " << rect.get_area();
  return 0;
}
```

VELDI

How to create a class with methods..

- **Below is almost the same example, but here we directly set the implementation inside the Rectangle class at declaration**

```
#include <iostream>

class Rectangle {
private:
    int width, height;
public:
    void set_height(int x) { height=x; };
    void set_width(int x)  { width= x; };
    int get_area()         { return width*height; };
};

int main () {
  Rectangle rect;
  rect.set_height(2);
  rect.set_width(4);
  std::cout << "area: " << rect.get_area();
  return 0;
}
```

Lists of classes

- You can also create lists of classes, below example will show you a list containing phone data, then print it

```
#include <iostream>
#include <list>

class Phone {
public:
	std::string name="";
	std::string color="";
	void Set(std::string n, std::string c) {name = n; color=c;};
};

std::list<Phone> my_phones;

int main() {
	Phone tmp_phone;
	tmp_phone.Set("Samsung","pink");
	my_phones.push_back(tmp_phone);
	tmp_phone.Set("Apple","brown");
	my_phones.push_back(tmp_phone);

	for (auto l: my_phones){
		std::cout << "The " << l.name << " is " << l.color << std::endl;
	}
}
```

Introduction to C++

Part 5

-

Class inheritance,

VELDI

Classes can inherit other classes

- **Classes have the possibility of inheriting other classes**
- **The new class will be extended with all the variables and methods from the inherited class**
- **This is particularly useful if you have a base class with common things that can be part of your other classes**
- **Inherit a class uses syntax:**
  - class derived_class_name: public base_class_name { /*...*/ };
- **To call an inherited constructor with arguments use :**
  - derived_constructor_name (parameters) : base_constructor_name (parameters) {...}

Example of class inheritance

- Below example contains a Car class that inherits from a Vehicle class.
- The Car class will contain the stuff from Vehicle class

```
#include <iostream>

class Vehicle { //a Vehicle class
public:
	Vehicle() {std::cout << "Vehicle" << std::endl;};
	int wheels = 4;
};

//class that inherits from Vehicle
class Car : public Vehicle{//inherits from Vehicle
public:
	Car() {std::cout << "Car" << std::endl;};
	int vehicle_type = 1; //Let say 1 means a car
};

int main() {
	Car my_car;
}
```

```
Output:		Vehicle
		Car
```

VELDI

Inheritance with special constructor

```cpp
#include <iostream>

class Vehicle {                         //a Vehicle class containing number of wheels
      int wheels;
public:
      Vehicle(int x) {wheels=x;};       //constructor with int argument
      void print_wheels() {std::cout << wheels;};
};

class Car : public Vehicle {            //inherits from Vehicle and add a type
      std::string v_type;
public:
      //Vehicle constructor uses first argument from Car
      Car(int x, std::string y) : Vehicle(x) {v_type = y;};
      void print_vehicle() {std::cout << v_type;};
};


int main() {
      Car my_car(4,"Car");
      my_car.print_wheels();            //from inherited Vehicle in Car
      my_car.print_vehicle();           //from Car class
      return 0;
}
```

Output: 4Car

Multiple inheritance

```
#include <iostream>
class Father {
public:
	Father() {		//Constructor with no argument
		std::cout << "Father: With no parameters\n"; }
	Father(int a) {		//constructor with int argument
		std::cout << "Father: With int=" << a << " parameter\n"; }
};

class Daughter: public Father {//inherit class Father
public:
	Daughter(int a) {//default Father constructor with no argument will be used
		std::cout << "Daughter: With int=" << a << " parameter\n"; }
};

class Son: public Father {//inherit class Father
public:
	Son(int a) : Father(a) {//uses Father constructor with int argument
		std::cout << "Son: With int=" << a << " parameter\n";}
};

int main() {
	Daughter kelly(1);
	Son bud(2);
	return 0;
}
```

```
Output:		Father: With no parameters
		Daughter: With int=1 parameter
		Father: With int=2 parameter
		Son: With int=2 parameter
```

Introduction to C++

Bonus

-

This-keyword

VELDI

This keyword

- The keyword this represents a pointer to the object whose member function is being executed. It is used within a class's member function to refer to the object itself
- One of its uses can be to check if a parameter passed to a member function is the object itself

```
#include <iostream>

class Dummy {
public:
	bool isitme(Dummy &param);
};

bool Dummy::isitme(Dummy &param) {
	if (&param == this)
		return true;
	else
		return false;
}

int main() {
	Dummy a;
	Dummy *b = &a;
	if (b->isitme(a))
		std::cout << "yes, &a is b\n";
	return 0;
}
```

This keyword.. (Bonus)

- It is also frequently used as a copy operator “=“

```
#include <iostream>
class Box {
   public:
      double getVolume(void) {return length * height;};
      void setLength( double len ) {length = len;};
      void setHeight( double hei ) {height = hei;};
      /* copy operand */
      Box& operator= (const Box& b) {
      this->length=b.length;
      this->height=b.height;
      return *this;
      }
   private:
      double length;      // Length of a box
      double height;      // Height of a box
};

int main() {
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here
   // box 1 specification
   Box1.setLength(11.0); Box1.setHeight(11.0);
   // box 2 now calls copy operand in class
   Box2=Box1;

   volume = Box1.getVolume();std::cout << "Volume of Box1 : " << volume << std::endl;   // volume of box 1
   volume = Box2.getVolume();std::cout << "Volume of Box2 : " << volume << std::endl;   // volume of box 2
   return 0;
}
```


veldikompetens.se
Thank you!
VELDI KOMPETENS