Lecture 2: Basics in C++

Jonas Kusch and Martina Prugger
University of Innsbruck

September 28, 2022

Last goals: You are able to

- ☑ understand main advantages and disadvantages of C++
- ☑ set up your project with VS code, git
- ☑ write and compile your own first code with C++
- ☑ use libraries, print outputs to terminal
- ☑ understand types

Last goals: You are able to

- ☑ understand main advantages and disadvantages of C++
- ☑ set up your project with VS code, git
- ☑ write and compile your own first code with C++
- ☑ use libraries, print outputs to terminal
- ☑ understand types

Today's learning goals: You will be able to

- ☐ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Last goals: You are able to

- ☑ understand main advantages and disadvantages of C++
- ☑ set up your project with VS code, git
- ☑ write and compile your own first code with C++
- ☑ use libraries, print outputs to terminal
- ☑ understand types

Today's learning goals: You will be able to

- ☐ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Ask questions any time!

Vector

- There are more datatypes, which are implemented in standard C++ libraries. Let's start with `std::vector`

```
#include <iostream>
#include <vector>

int main(){
    std::vector<double> v{1, 2, 3};
    v[2] = 1.0;
    std::cout<<v[0]<<" "<<v[1]<<" "<<v[2]<<std::endl;
    return 0;
}
```

- Note that you need to put an `std::` in front of the vector (and `cout`, `endl`). This is a namespace which we will cover later.
- indexing starts at 0 in C++!
- `size`, `reserve`, `resize`

Example

```cpp
#include <iostream>
#include <vector>

int main(){
    std::vector<double> v(5);
    v.reserve(10);
    v[0] = 1.0;
    v[2] = 1.0;
    std::cout<<v.size()<<std::endl;
    std::cout<< v[0] <<" "<<v[1]<<" "<<v[2]<<std::endl;

    return 0;
}
```

Your turn

Task

Write a program which generates two vector $v_1 = [0, 0.5, 1]$ and $v_2 = [0, sin(0.1), 1]$ of type float. Generate a third vector $v_3 = v_1 + v_2$.

What goes wrong?

```cpp
#include <iostream>
#include <vector>
#include <cmath>

int main(){
    std::vector<int> v(2);
    v[0] = 0;
    v[1] = 0.1;
    v[2] = 0.2;

    std::vector<int> v1(2);
    v1 = v + v;

    std::cout<< v1[0] <<" "<<v1[1]<<" "<<v1[2]<<std::endl;

    return 0;
}
```

Last goals: You are able to

- ☑ understand main advantages and disadvantages of C++
- ☑ set up your project with VS code, git
- ☑ write and compile your own first code with C++
- ☑ use libraries, print outputs to terminal
- ☑ understand types

Today's learning goals: You will be able to

- ☐ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Last goals: You are able to

- ☑ understand main advantages and disadvantages of C++
- ☑ set up your project with VS code, git
- ☑ write and compile your own first code with C++
- ☑ use libraries, print outputs to terminal
- ☑ understand types

Today's learning goals: You will be able to

- ☐ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Conditionals

- Are you familiar with conditionals in programming languages?
- `if`, `else if`, `else`

```
if( boolian == true ){
    // statement
}else if(boolian2 == true){
    // statement
}else{
    // statement
}
```

- New variables defined inside these environments are unknown to the global scope.

Conditionals

```cpp
#include <iostream>

int main(){
    int i = 2, j = 3;

    if( i == j ){
        i = j - 1;
    }else if(i == j - 1){
        i = j;
    }else{
        i = j - 1;
    }
    std::cout<<i;
    return 0;
}
```

conditional

```cpp
#include <iostream>

int main(){
    int i = 3, j = 3;

    if( i == j ){
        i = j - 1;
    }else if(i == j - 1){
        i = j;
    }else{
        i = j - 1;
    }
    std::cout<<i;
    return 0;
}
```

conditional

```cpp
#include <iostream>

int main(){
    int i = 3, j = 3;

    if( i == j ){
        int tmp = j - 1;
        i = tmp;
    }else if(i == j - 1){
        i = j;
    }else{
        i = j - 1;
    }
    std::cout<<i;
    return 0;
}
```

Today's learning goals: You will be able to

- ☑ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Today's learning goals: You will be able to

- ☑ use conditional statements
- ☐ use loops
- ☐ understand memory management
- ☐ start to use pointers

Loops

- Are you familiar with loops in programming languages?
- for, while, do while loops

```
for( long i = 0; i < 10; ++i ){
    // statement
}
while( boolian == true ){
    // statement
}
do{
    // statement
}while( boolian == true )
```

- break and continue

Solving an ordinary differential equation

Consider a simple ODE

$$\dot{y}(t) = \sin(y(t))$$

Forward Euler time discretization: Define grid $\{t_1, \cdots, t_{N_t}\}$ and define $y^n \simeq y(t_n)$

$$y^{n+1} = y^n + (t_{n+1} - t_n) \sin(y^n)$$

Task

Implement a forward Euler method with equidistant time step size $\Delta t = t_{n+1} - t_n = 0.01$. Store the solution at all time points $t \in [0, 1]$ in a vector and write it to a text file.

Today's learning goals: You will be able to

- ☑ use conditional statements
- ☑ use loops
- ☐ understand memory management
- ☐ start to use pointers

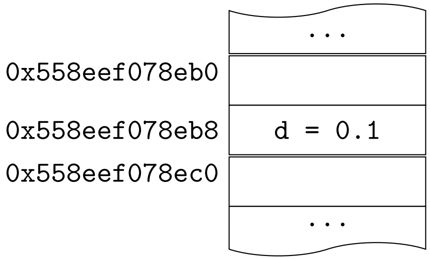
Today's learning goals: You will be able to

- [✓] use conditional statements
- [✓] use loops
- ☐ understand memory management
- ☐ start to use pointers

Addresses

- Every variable has a certain place in memory, called its address.
- Access address via & operator

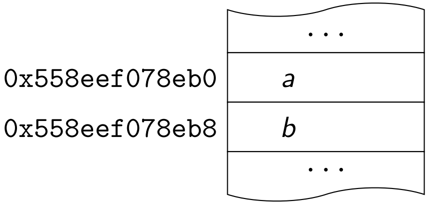
```
double d = 0.1;
std::cout<<"Address of d is "<< &d <<std::endl;
```



Now it's up to you...

- What does the code do? Which output do you expect?

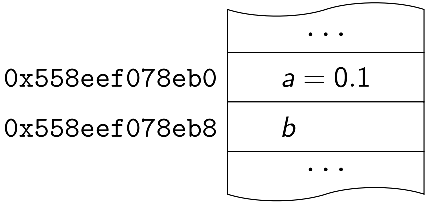
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



Now it's up to you...

- What does the code do? Which output do you expect?

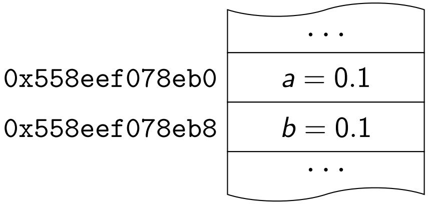
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



Now it's up to you...

- What does the code do? Which output do you expect?

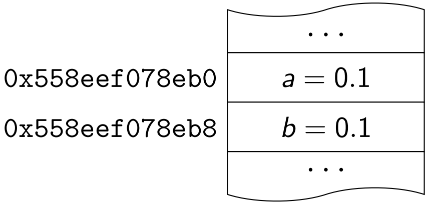
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



Now it's up to you...

- What does the code do? Which output do you expect?

```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```

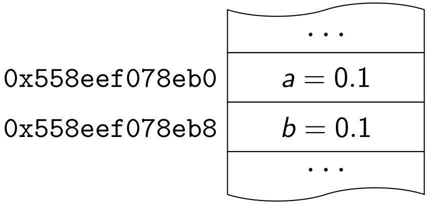


- Changing the value does not change address!
- Is there a datatype for addresses?

Now it's up to you...

- What does the code do? Which output do you expect?

```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```

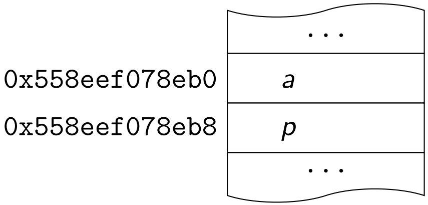


- Changing the value does not change address!
- Is there a datatype for addresses?

Pointers

- Datatypes to store an address is a pointer:

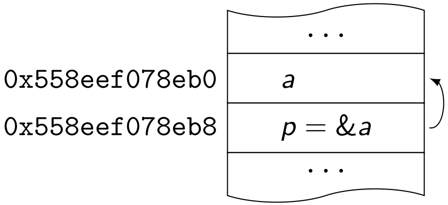
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



Pointers

- Datatypes to store an address is a pointer:

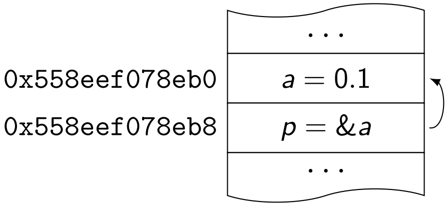
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



Pointers

- Datatypes to store an address is a pointer:

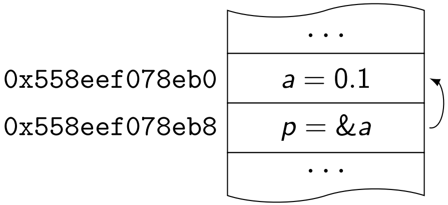
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



Pointers

- Datatypes to store an address is a pointer:

```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```

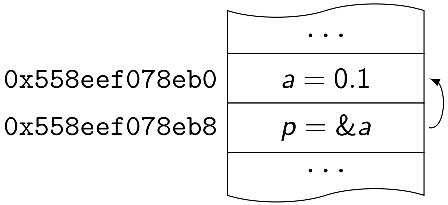


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with `*`.

Pointers

- Datatypes to store an address is a pointer:

```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```

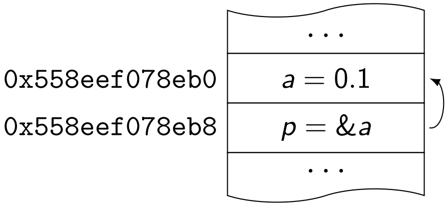


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with `*`.

Pointers

- Datatypes to store an address is a pointer:

```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```

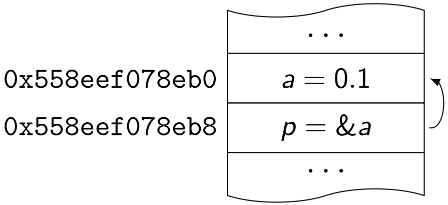


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with $*$.

Pointers

- Datatypes to store an address is a pointer:

```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```

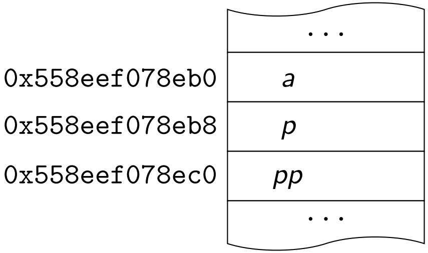


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with `*`.

Pointers on pointers

- Datatypes to store an address of a pointer is a double pointer:

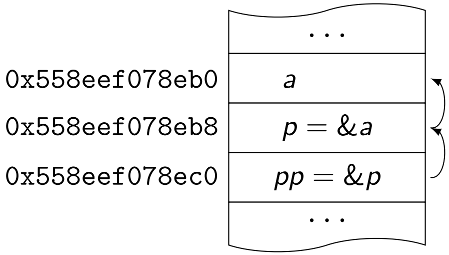
```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



Pointers on pointers

- Datatypes to store an address of a pointer is a double pointer:

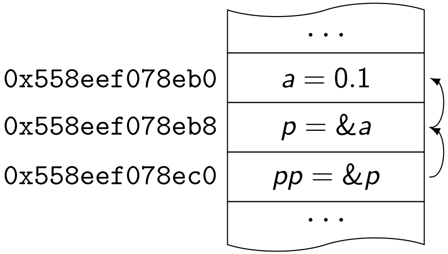
```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



Pointers on pointers

- Datatypes to store an address of a pointer is a double pointer:

```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```

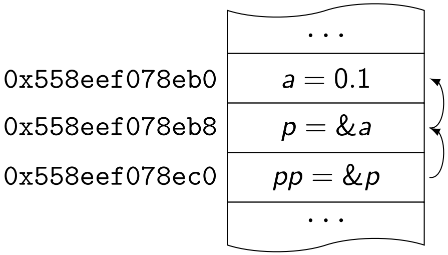


- Pointer on pointer is **pp.
- You can go on with ***p3, ...
- Dereference with **.

Pointers on pointers

- Datatypes to store an address of a pointer is a double pointer:

```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```

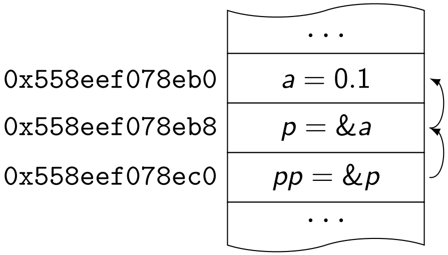


- Pointer on pointer is **pp.
- You can go on with ***p3, ...
- Dereference with **.

Pointers on pointers

- Datatypes to store an address of a pointer is a double pointer:

```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



- Pointer on pointer is **pp.
- You can go on with ***p3, . . .
- Dereference with **.

Your turn

Task

Write a code which changes the value of an integer `i` from 1 to 2 by using pointers. That is, do not use statements like `i = 2`.

Your turn

Task

Write a code which changes the value of an integer `i` from 1 to 2 by using pointers. That is, do not use statements like `i = 2`.

Task

Change the value to 3 with a pointer on a pointer.

Your turn

Task

Print the memory location of `pp`, `p`, and `i` by only using `pp`.

Your turn

Task

Print the memory location of `pp`, `p`, and `i` by only using `pp`.

Task

Given the code below, make sure that `*one = 1`, `*two = 2`, `*three = 3` without changing the first two lines and without using i, j and k.

```c
int i = 1, j = 2, k = 3;
int *one = &j, *two = &k, *three = &i;
```

What does the code do?

```cpp
#include <iostream>

int main(){
    int *i = 1, j = 2;
    std::cout<< i + j;

    return 0;
}
```

What does the code do?

```
#include <iostream>

int main(){
    int *i = 1, j = 2;
    std::cout<< i + j;

    return 0;
}
```

```
#include <iostream>

int main(){
    int i = 1, j = 2;
    int* p = &i;
    *p = *p + 2;
    std::cout << i + j;

    return 0;
}
```

What does the code do?

```cpp
#include <iostream>

int main(){
    int i = 1, *p = &i;
    *p = 2;
    std::cout << i + *p;

    return 0;
}
```

What does the code do?

```cpp
#include <iostream>

int main(){
    int i = 1, *p = &i;
    *p = 2;
    std::cout << i + *p;

    return 0;
}
```

```cpp
#include <iostream>

int main(){
    int i = 1, *p;
    *p = 2;
    std::cout << i + *p;

    return 0;
}
```

Now it's up to you...

Current learning goals: After homework and self-study

- ✓ use conditional statements
- ✓ use loops
- ✓ understand memory management
- ✓ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ use loops
- ☑ understand memory management
- ☑ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ use loops
- ☑ understand memory management
- ☑ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ use loops
- ☑ understand memory management
- ☑ start to use pointers

Any questions / remarks ? :)

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ use loops
- ☑ understand memory management
- ☑ start to use pointers

Any questions / remarks ? :) {*jonas.kusch*, *martina.prugger*}*@uibk.ac.at*

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ use loops
- ☑ understand memory management
- ☑ start to use pointers

Any questions / remarks ? :) {*jonas.kusch, martina.prugger*}*@uibk.ac.at*

Next learning goals:

- ☐ understand heap and stack
- ☐ construct static and dynamic arrays (`new`, `delete`, ...)
- ☐ start using functions