Lecture 2: Basics in C++

Johannes Gerstmayr and Markus Walzthöni

Material: Jonas Kusch and Martina Prugger
University of Innsbruck

October 6, 2023

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!

More types

- Boolean, integer, floating point numbers
- See `https://en.cppreference.com/w/cpp/language/types`

```cpp
bool b = true;
int i = -123; //usually 32 bit
long j = 12345; //32 or 64-bit
int64_t = 1234; //64-bit
float f = 1.23456e30; //32-bit, 7 digits, != Python!
double d = 1.23456789; //64-bit, 16 digits
char c='A'; //NOTE: UTF-8 should be used if possible!
```

Long short signed auto

- Type modifiers
- See `https://en.cppreference.com/w/cpp/language/types`

```
signed int i = -123;       //same as int!
unsigned long j = 12345; //4-5 does not make sense
short int k = 1234; //at least 16-bit
unsigned char c=(unsigned char)255; //also works for char
void* p;     //void = incomplete type
void v;      //impossible!
auto x = k; //x get's type of k
double a=2,b=3,c=4; //don't do this!
```

- Use only small set of data types which you need and know.

Const qualifier

- Make your C++ code safer

```
int n = 200;
... //long code
n=8;
... //long code
if (i < n) {...} //expected n to be 200

const int n = 200; //cannot be modified
n = 8; //compiler error
char str[n]; //only with const
```

- Compilers can optimize code if variables are const!

What goes wrong?

```
#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

- ☐ more types, variables and scope
- ☐ use conditional statements
- ☐ use loops and control flow
- ☐ 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;
}
```

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 2

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.

Scope

- In C++, there is a strict variable scope (different from Python)
- How long do variables 'live'?

```
int n = 200;
{
    int n = 30;
    n = n + 50;
    std::cout << n << "\n"; //n=80
}
std::cout << n << "\n"; //n=200
```

- But try to avoid reusing same variable names within a function!

Scope2

- How long do variables 'live'?
- `https://en.cppreference.com/w/cpp/language/scope`

```
for (int i = 0; // scope of i begins
     i < 10; // i is in scope
     ++i)    // i is in scope
{
    std::cout << i << ' '; // i is in scope
} // scope of i ends
```

- Note: desctructor called at end of scope!
- Namespaces allow further scope operations.

Switch

- `switch`: Available in Python as `match` only since 3.10
- see example: `https://en.cppreference.com/w/cpp/language/switch`

```
std::cin >> i;
switch (i)
{
    case 1:{
            int x = 1;
            std::cout << x << '\n';
            break; } // scope of 'x' ends here
    case 2:{
            std::cout << 2 << '\n';
            break; } // fallthrough without break!
    default:
        std::cout << "default\n"; // no error if i==3
        break;
}
```

- Always add: `break`, {...}, and `default:` !

Preprocessing directives / macros

- Do not use extensively `#define`, etc.
- But you will find them often in other codes!

```
#define ABC
#ifdef ABC
  #include "myfile.h" //also a preproc directive
#endif //ABC
#define MYCONST 42
int a = MYCONST;
#define MYCONST2 42+3
int x = MYCONST2*3; //guess x=?
```

- Prefer `const` or `constexpr`

Home work assignment

Task 3

Write a C++ program which inputs an arbitrary number. As an output, write if the number is an integer or a floating point number. Try to add switch/case to your implementation.

Addresses

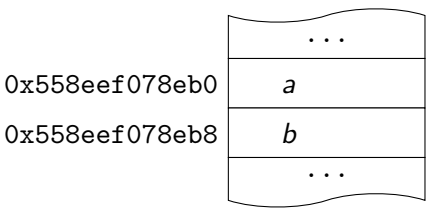
- Every variable has a certain place in memory, called its address.
- Access address via & operator

```
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?

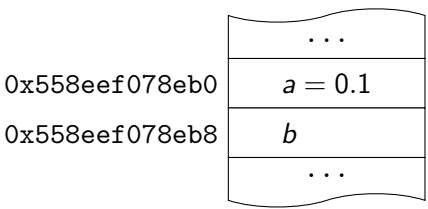
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



Now it's up to you...

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

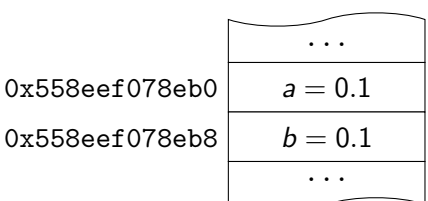
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



Now it's up to you...

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

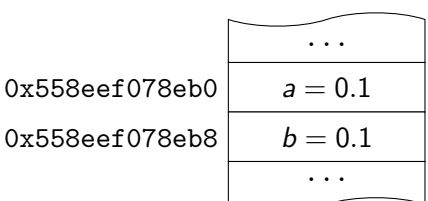
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



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;
```

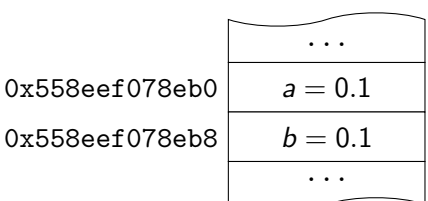


- Changing the value does not change address!
- Is there a datatype for addresses?

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;
```

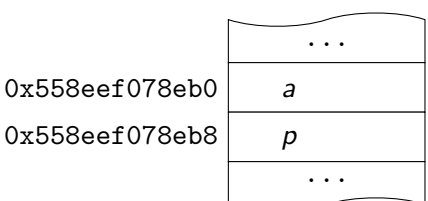


- Changing the value does not change address!
- Is there a datatype for addresses?

Pointers

- Datatypes to store an address is a pointer:

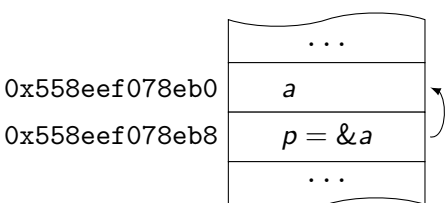
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



Pointers

- Datatypes to store an address is a pointer:

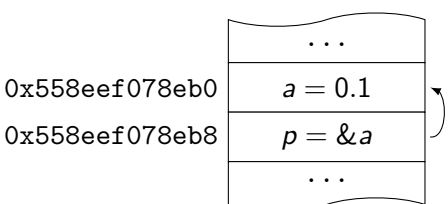
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



Pointers

- Datatypes to store an address is a pointer:

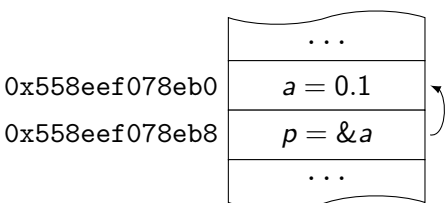
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *b <<std::endl;
std::cout<<"Addresses: "<< &a << " " << b << " " << &b;
```



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;
```

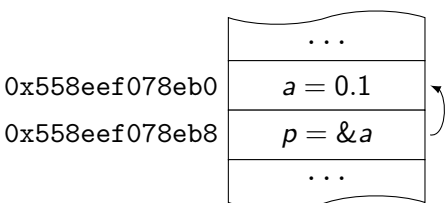


- Changes of address will change *p.
- Address of p remains the same.
- Pointers depend on data types.
- Dereference with *.

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;
```

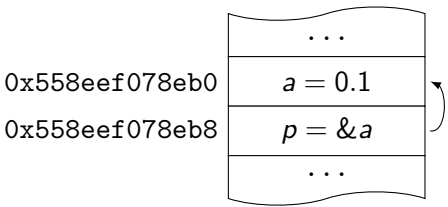


- Changes of address will change *p.
- Address of p remains the same.
- Pointers depend on data types.
- Dereference with *.

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;
```

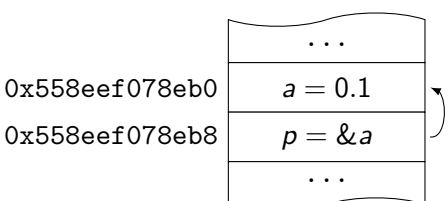


- Changes of address will change *p.
- Address of p remains the same.
- Pointers depend on data types.
- Dereference with *.

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;
```

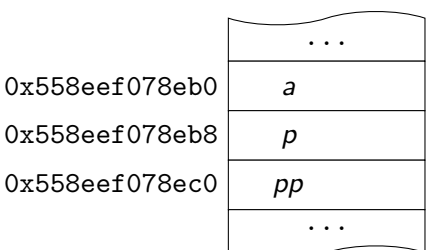


- Changes of address will change *p.
- Address of p remains the same.
- Pointers depend on data types.
- Dereference with *.

Pointers on pointers

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

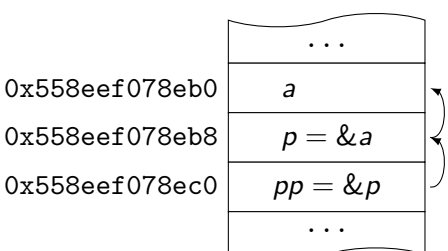
```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



Pointers on pointers

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

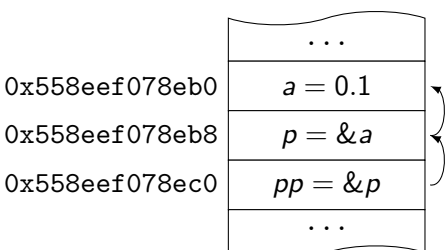
```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



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;
```

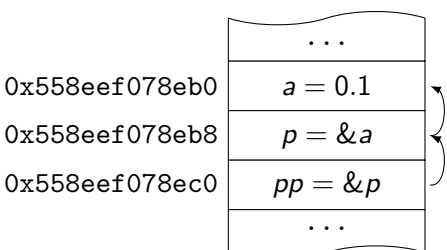


- Pointer on pointer is **pp.
- You can go on with ***p3, ...
- Dereference with **.

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;
```

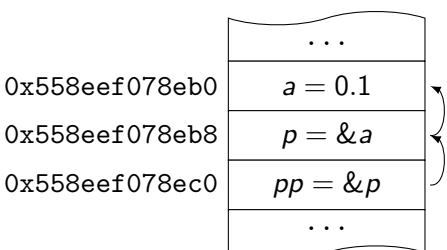


- Pointer on pointer is **pp.
- You can go on with ***p3, ...
- Dereference with **.

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.

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

What does the code do?

```
#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?

```
#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;
}
```

C++ Basics - References

Key differences between references and pointers:

- References **need** to be initialized
- References **can't** be `NULL`
- References **don't** provide no arithmetic operations
- References **don't** need to be dereferenced

Please use References wherever you can. If you **need** pointers then use `std::shared_ptr<T>` !

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ understand memory management
- ☑ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ understand memory management
- ☑ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ understand memory management
- ☑ start to use pointers

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ 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
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ understand memory management
- ☑ start to use pointers

Any questions / remarks ? :) {*johannes.gerstmayr,markus.walzthoeni*}*@uibk.ac.at*

Now it's up to you...

Current learning goals: After homework and self-study

- ☑ use conditional statements
- ☑ variables, types, scope
- ☑ use loops, control flow
- ☑ understand memory management
- ☑ start to use pointers

Any questions / remarks ? :) {*johannes.gerstmayr,markus.walzthoeni*}*@uibk.ac.at*

Next learning goals:

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