Lecture 3: Pointers and functions

Jonas Kusch and Martina Prugger

University of Innsbruck

September 28, 2022

Last goals: You are able to

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

Last goals: You are able to

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

Today's learning goals: You will be able to

- □ generate dynamic and static arrays
- □ understand pointer arithmetics
- □ use functions

Last goals: You are able to

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

Today's learning goals: You will be able to

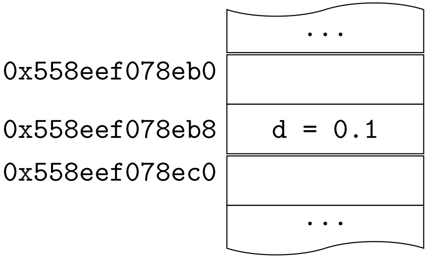
- □ generate dynamic and static arrays
- □ understand pointer arithmetics
- □ use functions

Ask questions any time!

Addresses

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

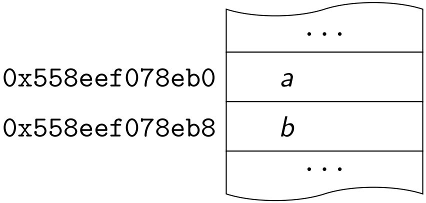
```
double d = 0.1;
std::cout<<"Address of d is "<< &d <<std::endl;
```



Now it’s up to you...

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

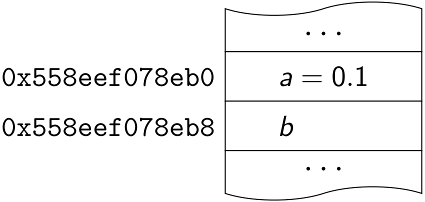
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



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;
```



...

0x558eef078eb0 $a = 0.1$

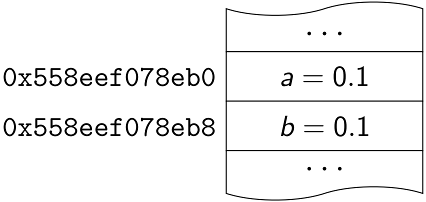
0x558eef078eb8 b

...

Now it's up to you...

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

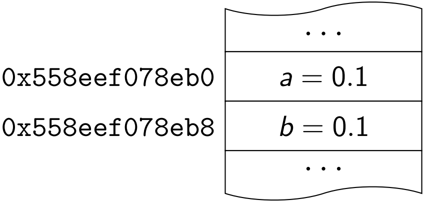
```
double a,b;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
a = 0.1;
b = a;
std::cout<<"Addresses: "<< &a << " " << &b <<std::endl;
```



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;
```

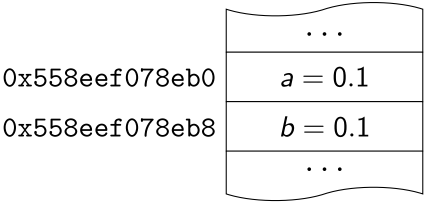


- Changing the value does not change address!
- Is there a datatype for addresses?

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;
```



...

0x558eef078eb0 $a = 0.1$

0x558eef078eb8 $b = 0.1$

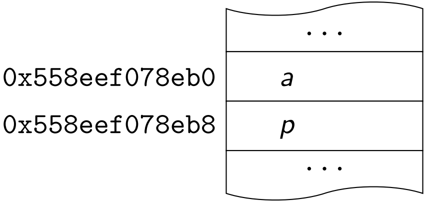
...

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

Pointers

- Datatypes to store an address is a pointer:

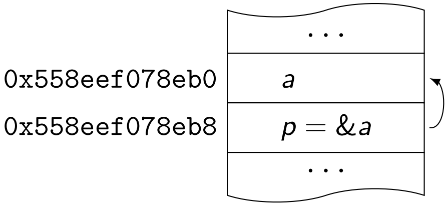
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *p <<std::endl;
std::cout<<"Addresses: "<< &a << " " << p << " " << &p;
```



Pointers

- Datatypes to store an address is a pointer:

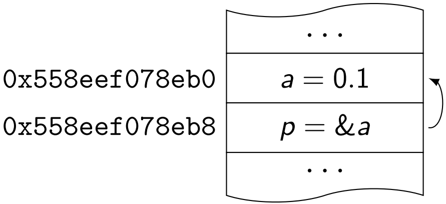
```
double a;
double* p;
p = &a;
a = 0.1;
std::cout<<"Values: "<< a << " " << *p <<std::endl;
std::cout<<"Addresses: "<< &a << " " << p << " " << &p;
```



Pointers

- Datatypes to store an address is a pointer:

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



. . .

0x558eef078eb0 $a = 0.1$

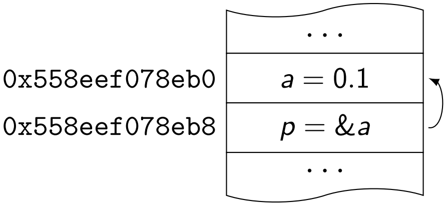
0x558eef078eb8 $p = \&a$

. . .

Pointers

- Datatypes to store an address is a pointer:

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



...

0x558eef078eb0 $a = 0.1$

0x558eef078eb8 $p = \&a$

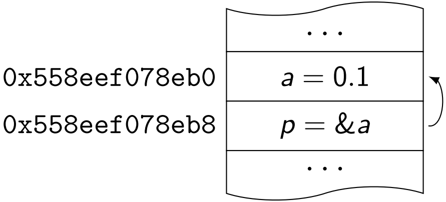
...

- 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 << " " << *p <<std::endl;
std::cout<<"Addresses: "<< &a << " " << p << " " << &p;
```

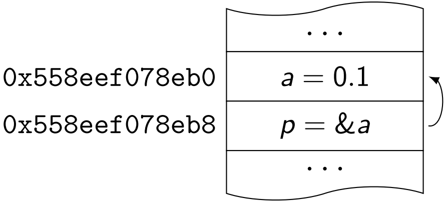


- Changes of address will change *p.
- Address of p remains the same.
- Pointers depend on data types.
- Dereference with *.

Pointers

- Datatypes to store an address is a pointer:

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

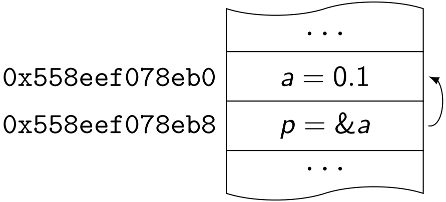


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with $*$.

Pointers

- Datatypes to store an address is a pointer:

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

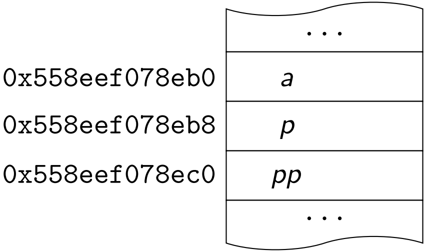


- Changes of address will change `*p`.
- Address of `p` remains the same.
- Pointers depend on data types.
- Dereference with `*`.

Pointers on pointers

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

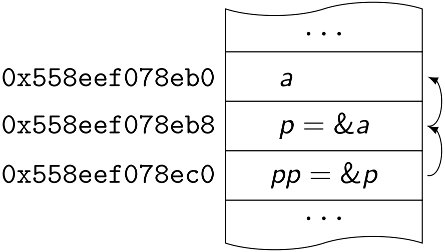
```
double a = 1.0, *p = &a, **pp = &p;
std::cout<<"Values: "<< a << " " << *p << " " << **pp << std::endl;
```



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;
```



...

0x558eef078eb0 a

0x558eef078eb8 $p = \&a$

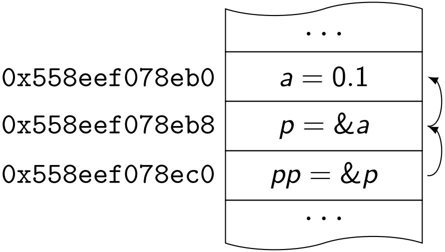
0x558eef078ec0 $pp = \&p$

...

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;
```



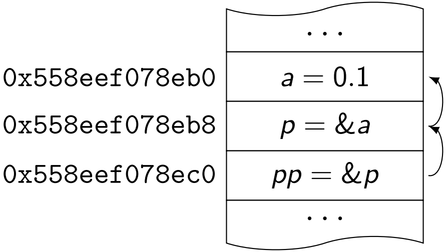
	...
0x558eef078eb0	$a = 0.1$
0x558eef078eb8	$p = \&a$
0x558eef078ec0	$pp = \&p$
	...

- 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;
```



. . .

0x558eef078eb0 $a = 0.1$

0x558eef078eb8 $p = \&a$

0x558eef078ec0 $pp = \&p$

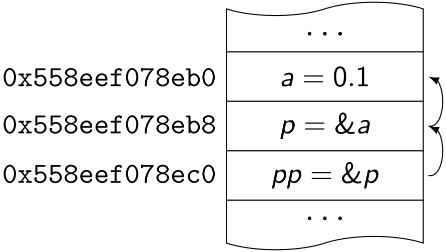
. . .

- 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;
```

Solution

Your turn

Task

Generate an `std::vector` for type `long` of size 3 with values $v = [0, 1, 2]$. Generate a pointer `p` which points on the vector's address. Change the size of `v` to 4 and add a value `v[3] = 3` by only using `p`.

Your turn

Task

Generate an `std::vector` for type `long` of size 3 with values $v = [0, 1, 2]$. Generate a pointer `p` which points on the vector's address. Change the size of `v` to 4 and add a value `v[3] = 3` by only using `p`.

Your turn

Task

Generate an `std::vector` for type `long` of size 3 with values $v = [0, 1, 2]$. Generate a pointer `p` which points on the vector's address. Change the size of `v` to 4 and add a value `v[3] = 3` by only using `p`.

Your turn

Task

Generate an `std::vector` for type `long` of size 3 with values $v = [0, 1, 2]$. Generate a pointer `p` which points on the vector's address. Change the size of `v` to 4 and add a value `v[3] = 3` by only using `p`.

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?

```cpp
#include <iostream>

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

    return 0;
}
```

```cpp
#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;
}
```

Avoid wrong code behaviour

- Uninitialized pointers can point on random space in memory.
- Avoid with NULL or 0 keyword.

```
double *p1;
double *p2 = 0;
std::cout<<*p1<< " might print some value"<<std::endl;
std::cout<<*p2<< " gives segmentation fault"<<std::endl;
```

Equivalent to

```
double *p1;
double *p2 = NULL;
std::cout<<*p1<< " might print some value"<<std::endl;
std::cout<<*p2<< " gives segmentation fault"<<std::endl;
```

Last goals: You are able to

☑ use conditional statements

☑ use loops

☑ understand memory management

☑ start to use pointers

Today's learning goals: You will be able to

□ generate dynamic and static arrays

□ understand pointer arithmetics

□ use functions

Last goals: You are able to

☑ use conditional statements

☑ use loops

☑ understand memory management

☑ start to use pointers

Today's learning goals: You will be able to

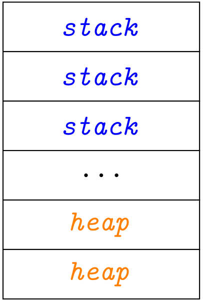
□ generate dynamic and static arrays

□ understand pointer arithmetics

□ use functions

Why?

- Pointers give control over memory.
- Pass data to a different program part without copying it (`functions`,`classes`,...).
- Control when to delete data (dynamic vs. static memory).



stack

stack

stack

. . .

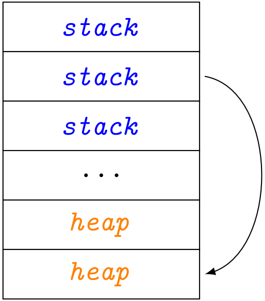
heap

heap

- **static** memory managed by compiler (*stack*)
- **dynamic** memory managed by user (*heap*)
- dynamic memory can be accessed with pointers (stored in stack)
- address space in heap is accessed with `new`

Why?

- Pointers give control over memory.
- Pass data to a different program part without copying it (`functions`,`classes`,...).
- Control when to delete data (dynamic vs. static memory).



- **static** memory managed by compiler (*stack*)
- **dynamic** memory managed by user (*heap*)
- dynamic memory can be accessed with pointers (stored in stack)
- address space in heap is accessed with `new`

Code presentation

Code presentation

```cpp
#include <iostream>

int main(){
    double dStatic = 0.1;
    double *dDynamic = new double; // allocate memory in heap
    *dDynamic = 0.1;

    std::cout<<dStatic<<" "<<*dDynamic<<std::endl;

    delete dDynamic; // free memory

    std::cout<<dStatic<<" "<<*dDynamic<<std::endl;

    return 0;
}
```

Arrays in heap

Arrays in heap

```cpp
#include <iostream>

int main(){
    double *v = new double [2]; // allocate array of size 2 in heap
    v[0] = 0.1;
    v[1] = 0.12;

    delete [] v; // free memory of entire array

    return 0;
}
```

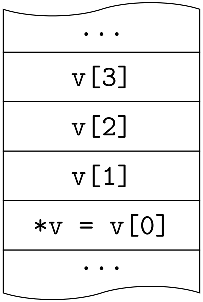
Arrays in heap

```cpp
#include <iostream>

int main(){
    double *v = new double [2]; // allocate array of size 2 in heap
    v[0] = 0.1;
    v[1] = 0.12;

    delete [] v; // free memory of entire array

    return 0;
}
```

Task

Rewrite your ODE solver by using dynamic arrays. Make sure to free your memory before the program terminates.

Arrays in heap



...

v[3]

v[2]

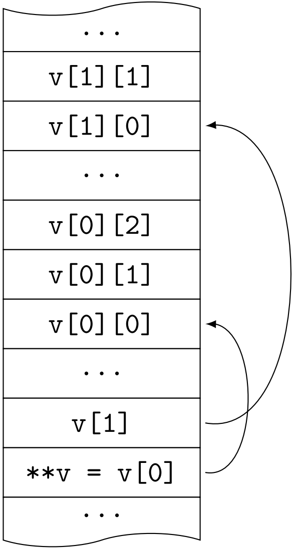
v[1]

*v = v[0]

...

- We store address of v[0] on v.
- Since v[1],... neighbor v[0] we also know their addresses.
- More about this when talking about pointer arithmetics.

Multi-dimensional arrays in heap



Multi-dimensional arrays in heap

Multi-dimensional arrays in heap

```cpp
#include <iostream>

int main(){
    int n = 3, m = 4;
    double** v = new double* [n]; // allocate array of pointers

    for( long i = 0; i < n; ++i)
        v[i] = new double [m]; // allocate double array for every v[i]

    v[0][1] = 0.1;

    for( long i = 0; i < n; ++i)
        delete [] v[i]; // delete array of doubles for every v[i]

    delete [] v; // delete array of pointers

    return 0;
}
```

Multi-dimensional arrays in heap

Task

Implement a 3-dimensional array a with dimension $n_1 = 2, n_2 = 3, n_3 = 4$. Fill the array with numbers $a_{ijk} = i + j + k$. Do not forget to free your memory before the program terminates.

What does the code do? What happens in memory?

```cpp
#include <iostream>

int main(){
    double* d = new double;
    *d = 0.1;
    double* p = d;

    delete p;

    std::cout<<*d<<std::endl;

    return 0;
}
```

What does the code do? What happens in memory?

```cpp
#include <iostream>

int main(){
    bool condition = true;

    if( condition ){
        double* d = new double;
        *d = 0.1;
    }

    std::cout<<*d<<std::endl;

    return 0;
}
```

What does the code do? What happens in memory?

```cpp
#include <iostream>

int main(){
    bool condition = true;
    double* d;

    if( condition ){
        d = new double;
        *d = 0.1;
    }

    std::cout<<*d<<std::endl;

    return 0;
}
```

Last goals: You are able to

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

Today's learning goals: You will be able to

- ☑ generate dynamic and static arrays
- □ understand pointer arithmetics
- □ use functions

Last goals: You are able to

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

Today's learning goals: You will be able to

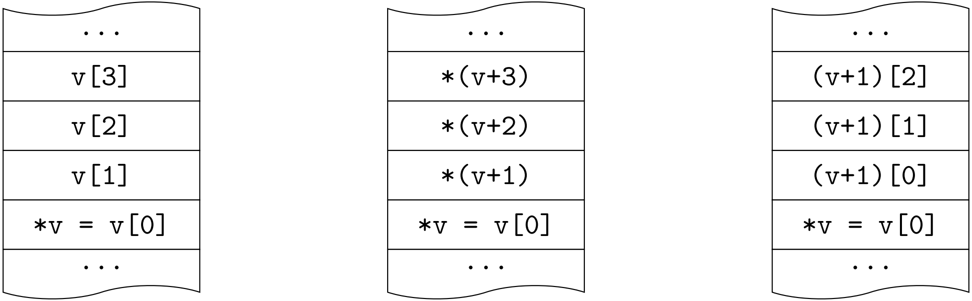
- ☑ generate dynamic and static arrays
- □ understand pointer arithmetics
- □ use functions

Pointer arithmetics

- Arithmetics on pointers allowed.
- `d[i]` equivalent to $= *(d + i)$

```cpp
#include <iostream>

int main(){
    double* d;
    d = new double [4];
    d[0] = 0.0; d[1] = 0.1; d[2] = 0.2;

    std::cout<< *d << " " << *(d + 1) <<std::endl;

    return 0;
}
```

Pointer arithmetics

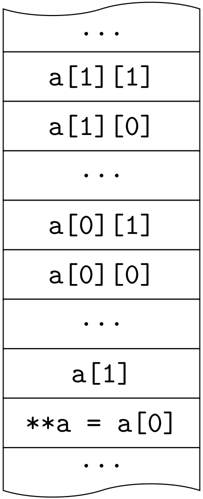


...	...	...
v[3]	*(v+3)	(v+1)[2]
v[2]	*(v+2)	(v+1)[1]
v[1]	*(v+1)	(v+1)[0]
*v = v[0]	*v = v[0]	*v = v[0]
...	...	...

What's the output?

```cpp
#include <iostream>

int main(){
    long** a = new long* [2];
    a[0] = new long [2];
    a[1] = new long [2];

    for( long i = 0; i < 2; ++i )
        for( long j = 0; j < 2; ++j )
            a[i][j] = i + j;

    std::cout<< *(a[1]+1) << " " << (*a - 1)[2] <<std::endl;
    std::cout<< *((a + 1)[0] + 1) <<std::endl;
    std::cout<< (a + 2)[0][2] << std::endl;

    return 0;
}
```

What’s the output?



→ `*(a[1]+1)`

→ `(*a - 1)[2]`

→ `*((a + 1)[0] + 1)`

→ `(a + 2)[0][2]`

Last goals: You are able to

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

Today's learning goals: You will be able to

- ☑ generate dynamic and static arrays
- ☑ understand pointer arithmetics
- □ use functions

Last goals: You are able to

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

Today's learning goals: You will be able to

- ☑ generate dynamic and static arrays
- ☑ understand pointer arithmetics
- □ use functions

Functions

- Is everyone familiar with functions in programming languages?

```
<return_data_type> function_name( <input_1>, <input_2>,... ){
    \\ function body
    return <return_value>
}
```

Functions

- Is everyone familiar with functions in programming languages?

```
<return_data_type> function_name( <input_1>, <input_2>,... ){
    \\ function body
    return <return_value>
}
```

```cpp
#include <iostream>

double add(double a, double b){
    double c = a + b;
    return c;
}

int main(){
    std::cout << add(1,2) <<std::endl;
    return 0;
}
```

Your turn

Exercise

Rewrite your ODE solver as a function which takes start time and time grid as input and returns the solution at each time point as output. Use another function to define the right-hand-side of your ODE.