

# A tutorial-based introduction to C

---

Martin Sparre

Potsdam University

# About C

The aims for this course are

- C is a low-level programming language.
- C is very fast!
- Commonly used in high-performance computing.

# Hello world!

A hello world program reads like:

```
#include <stdio.h>
```

```
int main(){
```

```
    printf("Hello world!\n");
```

```
    return 0;
```

```
}
```

# C is a compiled programming language

- compile with: `gcc HelloWorld.c -o HelloWorld`
- then run with: `./HelloWorld`

## Example: Loops are fast in C

A program that calculates the mean of the numbers between 0 and  $10^8$ :

In C:

```
int main(){
    int i;
    double mean;
    mean = 0.0;
    for(i=0;i<100000000;i++)
        mean+=i*1.0/100000000;

    return 0;
}
```

In Python:

```
mean = 0.0
for i in range(100000000):
    mean+=i/1e8
```

The C-program is  $> 10$  times faster than the Python program.

# Pointers and arrays

---

# Pointers handles memory

A pointer is a variable which holds a memory address.

Some syntax:

- Define a pointer to a double: **double \*x;** x is then the address of a double.
- Allocate memory and point x at the address:  
**x = malloc(sizeof(double));**  
or **x = malloc(sizeof(double)\*100).**  
(for a double or an array of 100 doubles, respectively).

# Pointers and arrays

Pointers and arrays are equivalent in C.

- To get the value at the address of the pointer `x` type `*x`; or alternatively `x[0]`;
- The following two statements are equivalent `*(x+i)`; and `x[i]`; It takes the  $i$ 'th element of the array `x`.  
Pointers and arrays are equivalent because `*(x+i) == x[i]`



# free()

The memory in a pointer can be de-allocated with the free command. i.e. **free(x);**

# Workflow with pointers/arrays

*//We declare two arrays*

```
double *x;
```

```
double *y;
```

*//Allocate memory for two arrays:*

```
x = malloc(sizeof(double)*NArray);
```

```
y = malloc(sizeof(double)*NArray);
```

*//Do stuff here... We now have two arrays x[0],x[1], x[2],...*

*//and y[0],y[1], y[2],...*

*//It is good practice to free arrays after use:*

```
free(x);
```

```
free(y);
```

See example6.c!

## Tip: It is good practice to initialise as NULL

*//We declare two arrays*

```
double *x;
```

```
double *y;
```

*//Initialise as NULL*

```
x = NULL;
```

```
y = NULL;
```

*//Allocate memory for two arrays:*

```
x = malloc(sizeof(double)*NArray);
```

```
y = malloc(sizeof(double)*NArray);
```

*//Do stuff here... We now have two arrays x[0],x[1], x[2],...*

*//and y[0],y[1], y[2],...*

*//It is good practice to free arrays after use:*

```
free(x);
```

```
free(y);
```