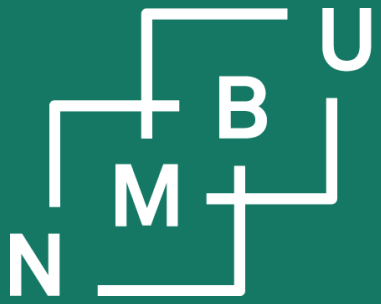
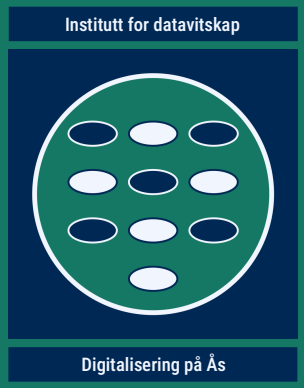


Noregs miljø- og biovitskaplege universitet



INF205

Ressurseffektiv programmering

1 Moderne C++

1.1 Frå Python til C++

1.2 Peikarar og datatabellar

C++ vs. Python: Language features

C++	Python

“What do you know about language features that are different in C++ and Python?”

Resource efficient computing: Why?

"What comes after Moore's law?"

P. J. Denning, T. G. Lewis, doi:10.1145/2976758, **2017**.

C. E. Leiserson *et al.*, doi:10.1126/science.aam9744, **2020**.

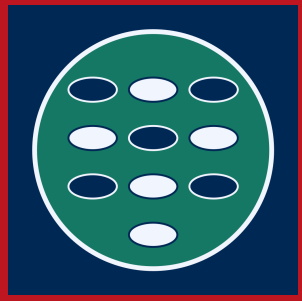
therein, see Tab. 1

Embedded systems

Digitalization entails pervasive computing, including at nodes or components without a great amount of computational resources.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap



Digitalisering på Ås

1 Moderne C++

1.1 Frå Python til C++

INF205

5. februar 2026

Ordliste

Compilation	Kompilering
Fundamental data type	Fundamental datatype
Linking	Lenking
Overloading	Overlasting
Return type	Returtype
Typing	Typing

Samanlikning: Ekvivalente programkodar

C++ code

```
#include <iostream>

bool is_prime(int n)
{
        if(n < 2) return false;
        for(int i = 2; n >= i*i; i++)
                if((n % i) == 0) return false;
        return true;
}

int main()
{
        int x = 900;
        if(is_prime(x))
                std::cout << x << " is prime.\n";
        else std::cout << x << " is not prime.\n";
}
```

Python code

```
def is_prime(n):
    if < 2:
        return False
    for i in range(2, 1 + int(n**0.5)):
        if n%i == 0:
            return False
    return True

x = 900
if is_prime(x):
    print(x, "is prime.")
else:
    print(x, "is not prime.")
```

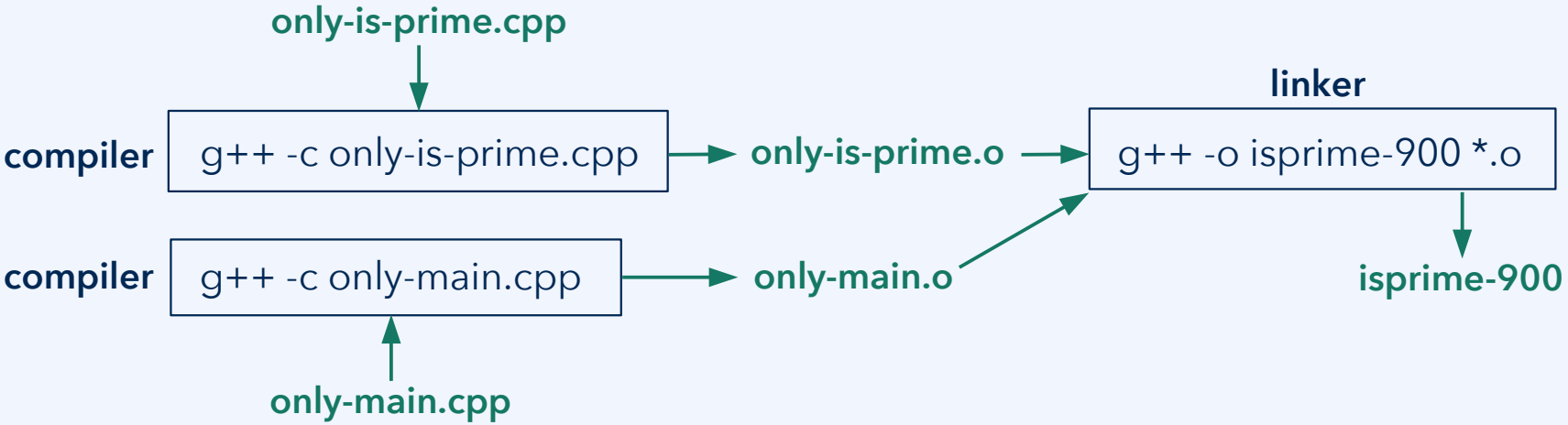
Eksempelfil: **is-prime-900.cpp**

C/C++ as a compiled language

Compile the code from the previous example (file name: isprime-900.cpp), using the GNU C++ compiler: **g++ isprime-900.cpp -o isprime-900**

Alternatively, in a Linux environment, we have GNU make: **make isprime-900**

Normally, codes comprise **multiple code files**. They are compiled separately (creating object files), and then linked. Only after linking there is an executable file. With the GNU C++ compiler, g++ is called both as **compiler** and **linker**:



Example file: is-prime-separate-files.zip

Split into header files (*.h) and code files (*.cpp)

```
#include <iostream>

bool is_prime(int n)
{
        if(n < 2) return false;
        for(int i = 2; n >= i*i; i++)
                if((n % i) == 0) return false;
        return true;
}

int main()
{
        int x = 900;
        if(is_prime(x))
                std::cout << x << " is prime.\n";
        else std::cout << x << " is not prime.\n";
}
```

Before, we split the code into two code files, one for each function.

How does main know is_prime at compile time? The **declaration**

bool is_prime(int n);

must be split from the **definition**:

bool is_prime(int n) { … }

Such declarations are normally stored in **header files** with the ending ".h". In this way, the header can be included by all external code that requires the same declarations.

Example file: is-prime-three-files.zip

GNU make

2.2 A Simple Makefile

Here is a straightforward makefile that describes the way an executable file called `edit` depends on eight object files which, in turn, depend on eight C source and three header files.

In this example, all the C files include `defs.h`, but only those defining editing commands include `command.h`, and only low level files that change the editor buffer include `buffer.h`.

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

(frå manualen til GNU make)

What differences can we see?

C++ code

```
#include <iostream>

bool is_prime(int n)
{
        if(n < 2) return false;
        for(int i = 2; n >= i*i; i++)
                if((n % i) == 0) return false;
        return true;
}

int main()
{
        int x = 900;
        if(is_prime(x))
                std::cout << x << " is prime.\n";
        else std::cout << x << " is not prime.\n";
}
```

Python code

```
def is_prime(n):
    if < 2:
        return False
    for i in range(2, 1 + int(n**0.5)):
        if n%i == 0:
            return False
    return True

x = 900
if is_prime(x):
    print(x, "is prime.")
else:
    print(x, "is not prime.")
```

What **differences between the languages** can we recognize from the introductory example?

C/C++ is a statically typed language

Most compiled programming languages are **statically typed** languages: The **data type** of each variable must be known to the compiler, at compile time.

Therefore, the type of a variable must be given when the variable is declared.

float, double
- single-precision and double-precision floating-point numbers

int
- the default signed integer type

short (int), long (int), long long (int)
- less/more memory and smaller/larger range of values

unsigned, unsigned short (int), unsigned long (int), …
- holds natural number (or zero); modulo-arithmetic applies: $-n = 2^k - n$

bool
- integer-like; *meant to* hold the value **false** (0) or **true** (1, or any value ≠ 0)

char, wchar_t
- integer-like; *meant to* hold a ASCII (char) or Unicode (wchar_t) character

Funksjonar treng argumenttypar og returtype

```
// deklarasjon:
ret_type funksjonsnamn(argtype_a argnamn_a, argtype_b argnamn_b, …);

// definisjon:
ret_type funksjonsnamn(argtype_a argnamn_a, argtype_b argnamn_b, …)
{
    …
    return return_value;  // returverdien må vera av type ret_type
}
```

Funksjonsoverlasting:

Same funksjonsnamn, forskjellige argumentlister, forskellige implementeringar

```
// får eit heiltal som argument
//
void do_something(int n){ … }
```

```
// får eit flyttal som argument
//
void do_something(double x){ … }
```

Glossar

Fundamental datatype

Definisjon: datatype på éin enkel eller skalar verdi, eller ein variabel som kan få ein slik verdi, og som ikkje består av fleire datafelt, delvariablar e.l.

I mange objektorienterte programmeringsspråk, deriblant C++, er det eit skilje mellom *fundamentale datatyper*, som blir instansiert av *fundamentale dataelement*, og *klasser*, som blir instansiert av *objekt*.

I motsetning til det kjenner Python utelukkande objekt; i Python er alt eit objekt.

Glossar

Kompilering

Definisjon: prosessen med å omsetja ein menneskeleg lesbar kjeldekode til ein maskinkode på lågare nivå; det gjerst med ein kompilator

- I C/C++ blir kompilatoren anvend på kjeldefiler; i C++ får desse filene vanlegvis filendinga *.cpp. Kompilatoren blir ikkje direkte anvend på header-filer (som vanlegvis får *.h-ending).
- Maskinkoden laga gjennom kompilering av ei kjeldefil med ein C/C++-kompilator kallast for objektkode; slik kode blir lagra i objektfiler, som vanlegvis får *.o-filending.
- Objektfilene som blir produsert ved kompilering er ikkje køyrbare av seg sjølve: Dei må lenkjast saman til ei frittståande køyrbar fil.

Glossar

Overlasting

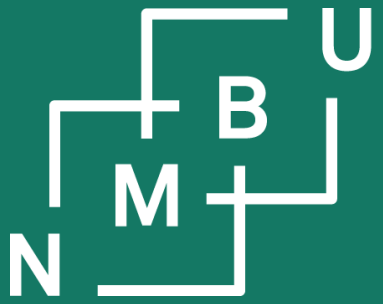
Definisjon: bruk av fleire funksjonar med same namn og forskjellige parametertypar

```
bool is_prime(int n)
{
        if(n < 2) return false;
        for(int i = 2; n >= i*i; i++)
                if((n % i) == 0) return false;
        return true;
}
```

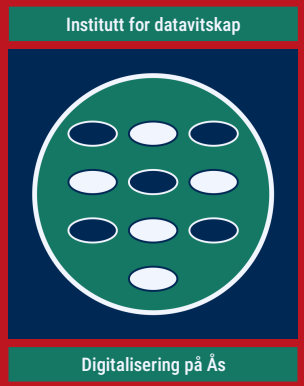
string-to-integer

```
bool is_prime(std::string s)
{
        return is_prime(std::stoi(s));
}
```

Jf. **overlasting.cpp**



Noregs miljø- og biovitskaplege universitet



1 Moderne C++

1.1 Frå Python til C++

1.2 Peikarar og datatabellar

INF205

5. februar 2026

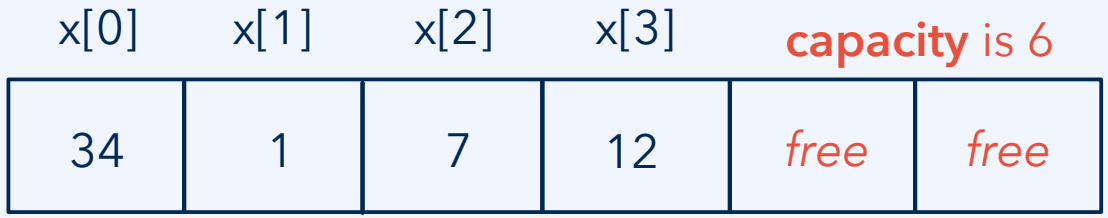
Ordliste

Argument passing by reference	Referanseoverføring
Argument passing by value	Verdioverføring
Array	Datatabell (array, rekkje)
Dereferencing (indirection)	Dereferering (indireksjon)
Pointer	Peikar
Reference	Referanse

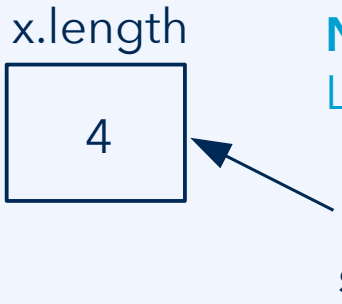
Dynamic arrays (such as lists in Python)

An array is a sequence of data items of the same type that is contiguous in memory. Python lists are contiguous in memory, *i.e.*, they are arrays. They can also grow or shrink in size over time: **Python lists are dynamic arrays**.

Dynamic data structures can change in size and/or structure at runtime. For an array, this can be implemented by **allocating reserve memory** for any elements that may be appended in the future. When the capacity of the dynamic array is exhausted, all of its contents need to be shifted to another position in memory.



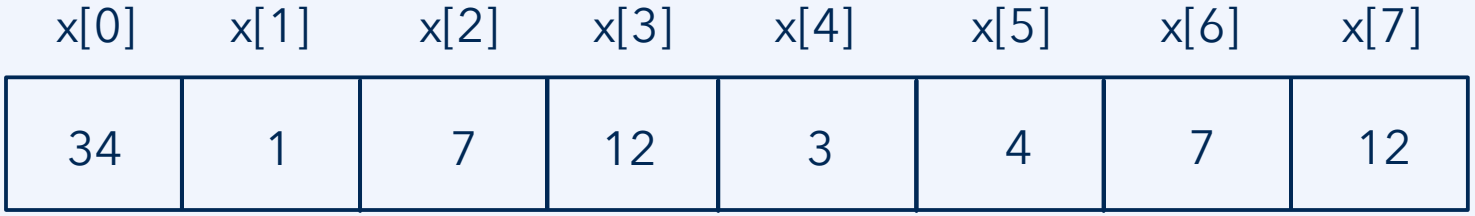
x = [34, 1, 7, 12]



Note: More memory is allocated than strictly necessary.
Like before, the elements are contiguously arranged in memory.

Static arrays

Arrays in C/C++ are static: When declaring the array, the array size is specified and the exact amount of memory required for these data items is allocated. The array size does not change over time.



Accessing elements of an array is highly efficient: When x[i] is accessed, the compiler transforms this into accessing the memory address x + sizeof(int) * i.

How do we declare a array?

- Give the size as constant expression in square brackets; *e.g.*, int values[6];

How do we initialize an array?

- Explicitly give all the values: int values[ ] = {4, 2, 3, -7, 2, 3};
- Initialize to **all zeroes**, indicating the array size: int values[6] = { };

Jf. **array-intro.cpp**

Kva er ein peikar?

Peikarar er variablar – samanlikning med andre datatypar:

- Ein **int** er ein variabel med ein heiltalsverdi som **7**.
- Ein **std::string** er ein variabel med ein streng som **"INF205"** som verdi.
- Ein peikar til X, av type **X***, er ein variabel som får ein minneadresse som verdi, t.d. noko som **0x7ffeaea5174c**. Den verdien kan tolkast som adresse i minnet der ein verdi av type X er blitt lagra.
- Det er god praksis å setja peikarverdien til **nullptr** ("null pointer") mår det er umogleg å tilordne peikaren ein gyldig minneadresse.
- Minne for eit X-objekt kan **allokerast** for hand gjennom **X* pt = new X**.
- Minnet kan **deallokerast** (frigjerast) att ved bruk av **delete pt**.

Ein **peikar** er ein variabel som har ein **minneadresse** som verdi.

- double* b er ein peikar til ein adresse som blir brukt til å lagre ein double.
- Adressen av ein variabel får vi ved å **referere** variablen, t.d. pt = &var;
- pt er addressen; den kan vi **dereferere** (*pt) og få tilgang til innhaldet.

Operators for referencing (&) and dereferencing (*)

Referencing operator &:

- Used to obtain the address of a variable: &x is the address of x.
- If x has type X, the address has the type X*, *i.e.*, "pointer to X."

```
int x = 5;  int* y = &x;
```

Dereferencing (indirection) operator *:

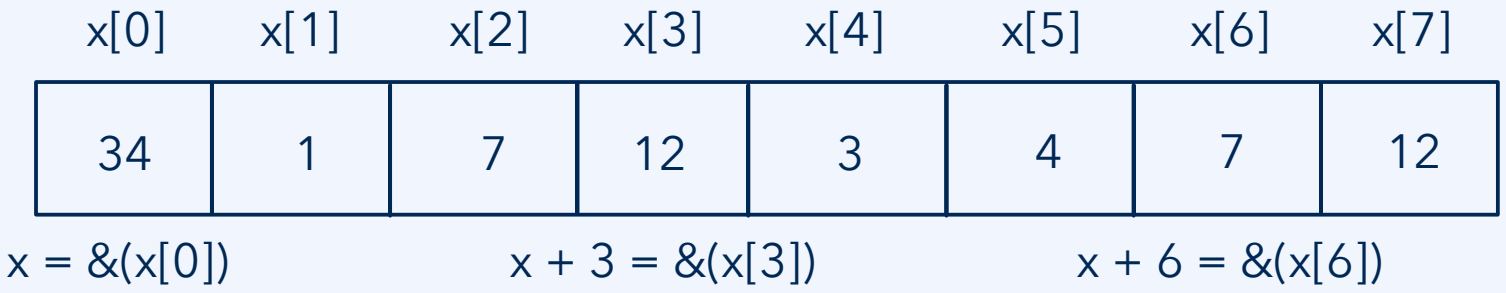
- If y is a pointer of type X* (pointer to X), the value of y is an address.
- To access the value stored at the address y, we dereference it as *y.
- The value stored at y, and accessed by *y, is then of type X.
- & and * are inverse operators, therefore, *(&x) is the same as x:

```
int x = 5;  int* y = &x;  cout << x << " is the same as " << *y;
```

C/C++ arrays are pointers

See example create-simple-segfault.cpp

An array contains a sequence of elements of the same type, arranged **contiguously in memory**. This supports fast access using **pointer arithmetics**. Once created, the size of a C/C++ array is fixed; we cannot append elements.



In C/C++, the type of an array such as **int[] is the same as the corresponding pointer type int***, *i.e.*, **the array actually is a pointer**. Its value is an address at which an integer is stored, namely, the memory **address of the first element**.

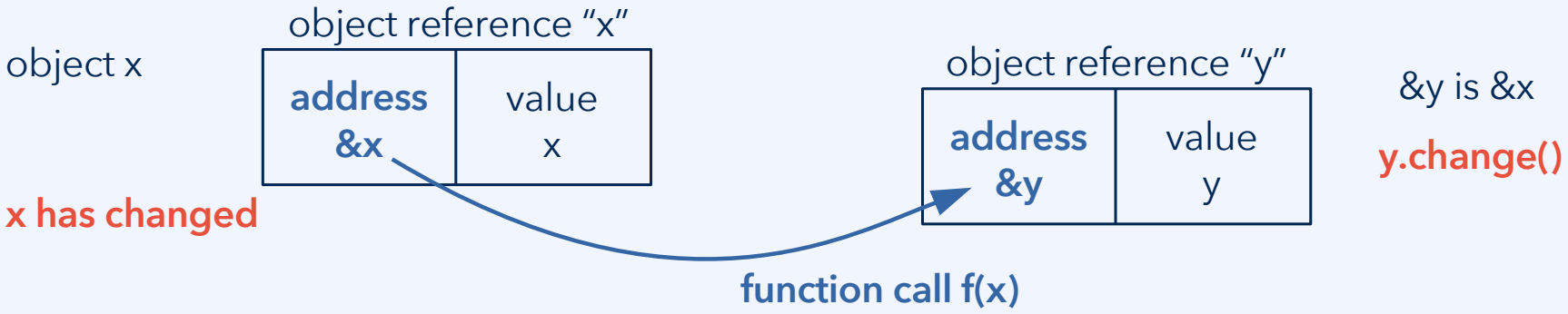
When x[i] is accessed, the compiler transforms this into x + sizeof(int) * i.

- **Allocation** is done with **new**. Example: **int* i = new int[8]();**
- **Deallocation** is done with **delete[]**. Example: **delete[] i;**

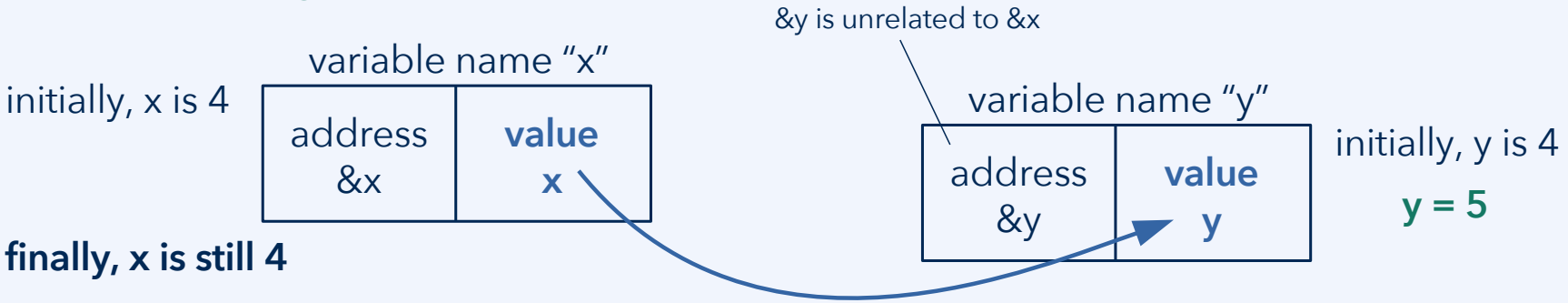
Verdioverføring i C++ (samanlikna med Python)

In Python blir *objektreferansen* overført som verdi (objektreferanseoverføring):

Objektreferanseoverføring i Python (og på liknande måte i Java)



Verdioverføring

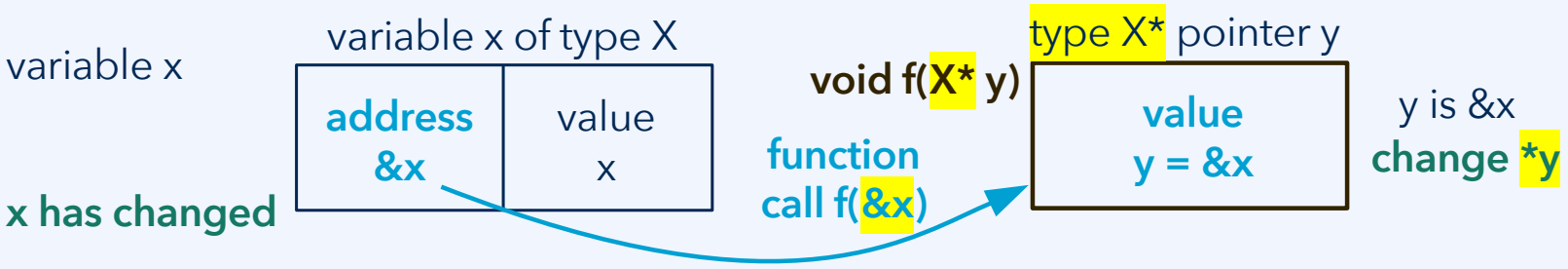


Jf. **pass-by-value-or-reference.cpp**

Pass by reference in C++ (compared to pass by value)

Pass by value: A new copy of the argument value(s) is created in memory. The function works with the copy. The function cannot access the original variable.

Pass by reference: The function is enabled to access the original variable at its address in memory. No copy is created. Changes affect the original variable.

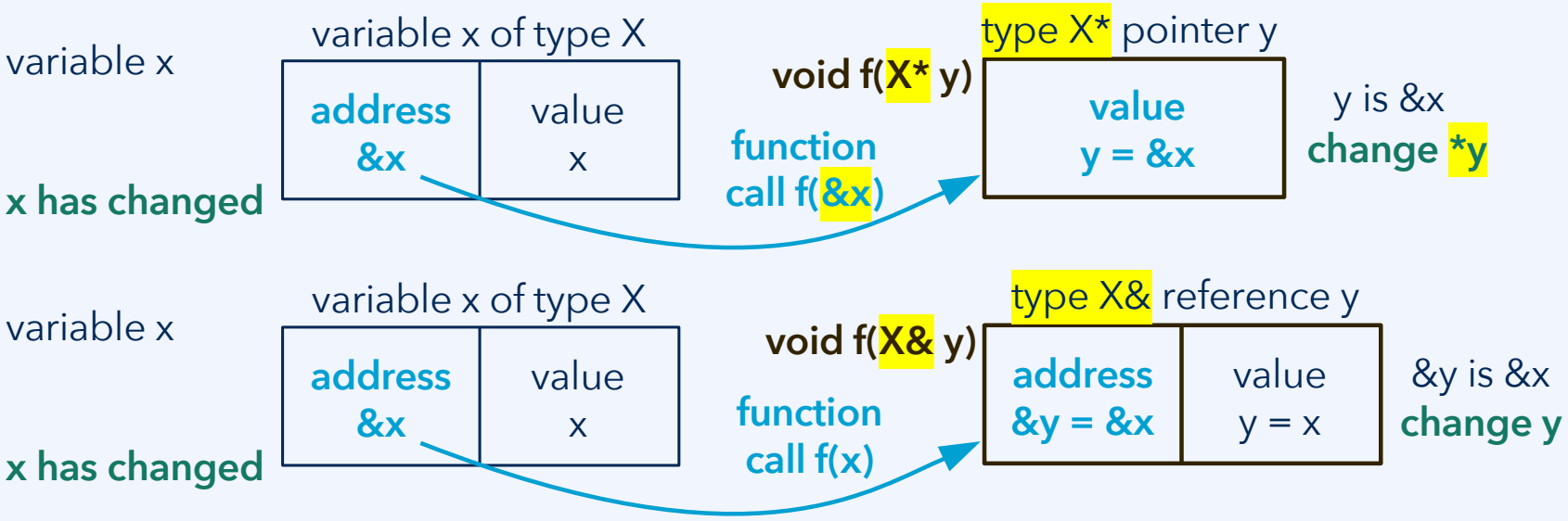


Jf. **pass-by-value-or-reference.cpp**

Pass by reference in C++ (two ways of doing it)

Pass by value: A new copy of the argument value(s) is created in memory. The function works with the copy. The function cannot access the original variable.

Pass by reference: The function is enabled to access the original variable at its address in memory. No copy is created. Changes affect the original variable. C++ has two mechanisms for this: **Passing a pointer** and **passing a reference**.*



*Unfortunately there is some terminology confusion about this. We will call both "**pass by reference**."

Jf. **pass-by-value-or-reference.cpp**

Referanseoverføring vs. verdioverføring

Fordelar ved å overføre eit funksjonsargument som verdi:

- **Minnet** for funksjonsparameterane ligg i kallstakken; **kompilatoren** tek vare på dette. Det er trgyt, og programmeraren treng ikkje å bry seg.
- Kallstakken kan **optimiserast ved komplieringstid**, og åtgangen til minnet i kallstakken fungerer effektivere (færre indireksjonar).
- **Levetida til variabelen** stemmer overeins med køyretida til dei funksjonane der den blir brukt.
- Den opphavlege verdien (i den ytre funksjonen, den som blir overført) er verna mot uforventa eller ugjennomsiktige forandringar.
- Koden er dimed meir modulær, og utviklingsprossessen er enklare.

Kva er fordelane ved å overføre argument som peikarer/referansar?

Det må det finnast ein grunn til … ellers ville ikkje vi ha det som mekanisme. **Men kva?** Når og på kva måte kan det vera lurt å nyte seg av den mekanismen?

Pass by reference using a pointer vs. a reference

Pointers and references are two equivalent notations for the same techniques.

```
void some_function(int& parameter) {
  …
  // convert the reference to a pointer
  int* y = &parameter;
  // now we can work with pointer y
  …
}
```

```
void some_function(int* parameter) {
  …
  // convert the pointer to a reference
  int& x = *parameter;
  // now we can work with reference x
  …
}
```

Advantages of pass-by-reference **using a reference**:

- Some memory-related errors become less likely if we only work with references; *e.g.*, errors from applying incorrect pointer arithmetics.
- Looks more like Java, Python, and other modern high-level languages.

Advantages of pass-by-reference **using a pointer**:

- It is visible to the programmer at all times that we deal with memory.
- Looks more like C, and it is closer to the object-code representation.

Glossar

Datatabell (array, rekkje)

Definisjon: variabel som er lagra i ein samanhengande region i minnet, og som kan innehalde eit fleirtal av elementære variablar eller objekt

- Ein datatabell som inneheld element av type X, er sjølv av type X*, dvs. peikar til X. Adressen er den som høyrer til det fyrste arrayelementet.
- Når eit array har blitt allokert manuelt på haugen, ved bruk av new, må det deallokerast gjennom delete[].

Peikar

Definisjon: variabel med ein minneadresse som verdi

Glossar

Referanse

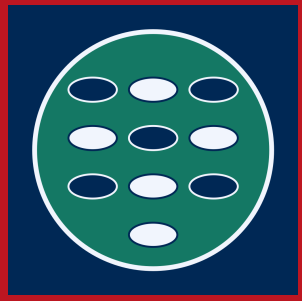
Definisjon: alias for data som (ved programkøyretid) er lagra på ein viss minneadresse

- Referansen medfører minneadressen, men den konkrete adressverdien blir ikkje synleggjort, og programmeraren får ikkje direkte tilgang til den.

- I Python kan variablar utelukkande få objektreferansar som verdi. Dermed er verdiar som blir overført i Python, t.d. ved parameteroverføring til ein funksjon eller metode, alltid objektreferansar. Denne måten å overføre data kallast for objektreferanseoverføring ("pass by object reference"), definert som verdioverføring av ein objektreferanse ("an object reference is passed by value").

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

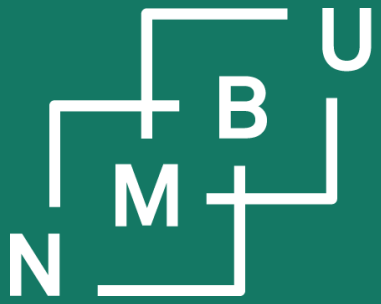


Digitalisering på Ås

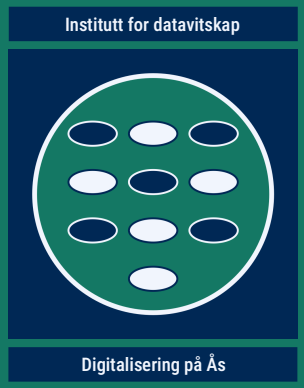
Samandrag / diskusjon

INF205

5. februar 2026



Noregs miljø- og biovitskaplege universitet



INF205

Ressurseffektiv programmering

1 Moderne C++

1.1 Frå Python til C++

1.2 Peikarar og datatabellar