

## TDT4102

## Prosedyre- og objektorientert programmering

The image features a large, stylized 'C++' logo in blue. Behind the logo is a snippet of C++ code in a monospaced font, tilted at an angle. The code includes a header, namespace declaration, and a main function that prints 'Hello World!'.

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!\n";
    return 0;
}
```

## Arrays (Tabeller)

# Dagens forelesing

- Repetisjon
  - Funksjoner og parametre
  - Testing
- Arrays (tabeller)
  - Syntaks og bruk
  - Kapittel 5
- Datatypen char og C-strenger

# Funksjoner repetisjon



- Bruk funksjoner, tenk funksjoner, lag funksjoner
- Funksjoner har:
  - Navn, identifiserer funksjonen
  - Parameterliste, definerer rekkefølge og type
  - Returtype som definerer datatypen til resultatet
  - En implementasjon
- Call-by-value, Call-by-reference, Call-by-pointer

# Kort repetisjon – finn 2 feil.

```
#include <iostream>
using namespace std;

void mySwap(double x, double y);

int main(){
    double a = 1.5;
    double b = 5.8;
    cout << "Før swap: a = " << a << ", b = " << b << endl;
    mySwap(a, b);
    cout << "Etter swap: a = " << a << ", b = " << b << endl;
}

void mySwap(double x, double y){
    int temp = x;
    x = y;
    y = temp;
}
```

Vi har glemt å bruke  
pekere eller call by  
reference

I tillegg har vi **int temp**  
Som gjør at x konverteres til int før  
det konverteres tilbake til double

# Swapping – med pekere

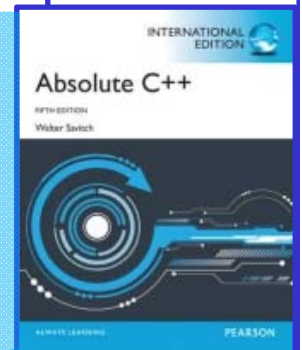
```
#include <iostream>
using namespace std;

void mySwap(double* x, double* y);

int main(){
    double a = 1.5;
    double b = 5.8;
    cout << "Før swap: a = " << a << ", b = " << b << endl;
    mySwap(&a, &b);
    cout << "Etter swap: a = " << a << ", b = " << b << endl;
}

void mySwap(double* x, double* y){
    double temp = *x;
    *x = *y;
    *y = temp;
}
```

Plassering  
av \*  
Side 451



# Swapping – med referanser

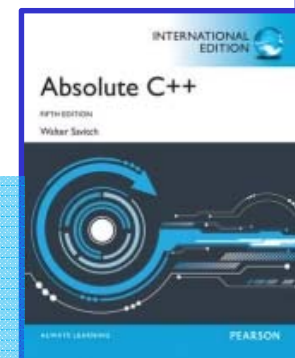
```
#include <iostream>
using namespace std;

void mySwap(double& x, double& y);

int main() {
    double a = 1.5;
    double b = 5.8;
    cout << "Før swap: a = " << a << ", b = " << b << endl;
    mySwap(a, b);
    cout << "Etter swap: a = " << a << ", b = " << b << endl;
}

void mySwap(double& x, double& y){
    double temp = x;
    x = y;
    y = temp;
}
```

Side  
176 - 181



# Testing



- Det er fort gjort å gjøre feil ved koding
  - Kompileringsfeil (compile time)
    - F.eks. syntaksfeil m.m. som gjør at kode ikke kompilerer
  - Logiske feil
    - Programmet gjør ikke det programmereren har forsøkt å beskrive i programmet (programmeringsfeil)
  - Kjøretidsfeil (run time)
    - Gir et program som "kræsjer"
      - F.eks. divisjon med 0
      - F.eks. skrivning «bak slutten av en tabell»
    - Kan gi evig løkke (Kan også være logiske feil)
  - Noen feil oppstår bare av og til («Heisenbug»)
    - Uinitialiserte variable, pekere som peker feil
    - Kan være fatale → bruk «defensiv programmering»
- Testing er en sentral del av moderne programvareutvikling

# Hva kan gå galt?

```
int main(){
    char meny = 'A';
    double fart = 0.0;
    double tid = 0.0;
    double avstand = 0.0;
    do {
        cout << "\nBeregne Avstand eller Tid (bruk A eller T): ";
        cin >> meny;
    } while ((meny != 'A') && (meny != 'T'));

    if (meny == 'A'){
        cout << "\nOppgi fart og tid: ";
        cin >> fart >> tid;
        avstand = beregnAvstand(fart, tid);
        skrivResultat(fart, tid, avstand);
    } else {
        cout << "\nOppgi fart og avstand: ";
        cin >> fart >> avstand;
        tid = beregnTid(fart, avstand);
        skrivResultat(fart, tid, avstand);
    }
}
```

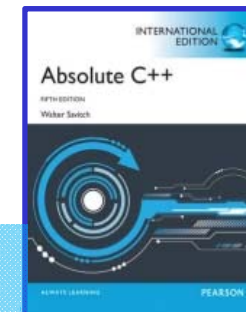
```
double beregnTid(double kmt, double avstand){
    return ((60 / kmt) * avstand);
}
```

Vi må tenke gjennom hvordan funksjonen skal fungere hvis kmt og/eller avstand er 0 eller negativt



# Dagens tema: Arrays (tabeller)

Kap. 5



- Så langt kun brukt enkelt-variabler
- Ofte behov for å kunne håndtere data som består av **mange verdier av samme type**
- **Arrays** (tabeller)
- Refererer til en samling dataelementer av lik type vha. ett variabelnavn og en indekseringsmekanisme («utplukkingmekanisme»)

# Anvendelse

- array (tabell) i C++ er en "lavnivå" konstruksjon
  - kan oppleves både rigid og upraktisk
  - viktig del av språket
    - å kjenne til og viktig å kunne bruke
- C++ biblioteket inneholder en rekke andre typer som kan brukes til lagring av samlinger av verdier
  - Som har samme dynamiske egenskaper som du kjenner fra Matlab eller Python

# Deklarering og initialisering

- Tabeller deklarerer som vanlige datatyper
  - men vi bruker **[n]** for å fortelle at det er tabell av denne datatypen (hvor n er størrelsen på tabellen)
  - tabell med plass til 5 heltall: **int score[5];**
  - setter av plass til tabellen i minnet, men husk at tabellen i dette tilfellet er uinitialisert (har tilfeldig innhold)
- Initialisering med opplisting i krøllparanteser :

initialiserer alle  
fem elementene

```
int score[5] = {44, 66, 32, 21, 43};  
int score[5] = {44, 66, 32};  
int score[5] = {};
```

initialiserer bare de  
tre første elementene,  
mens de to siste har  
ukjent verdi

initialiserer alle  
elementer til 0

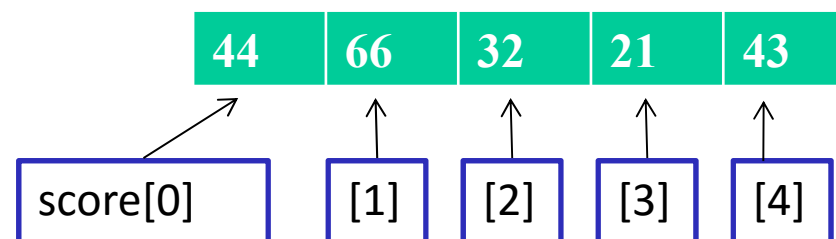
# Tabeller:

## Lese og skrive verdier

- Velg element med indeksoperatoren [ ]

```
int score[5] = {44, 66, 32, 21, 43};
```

- **NB! indeksen begynner på 0!**



- Skrive verdi:

```
score[3] = 80;
```



- Lese verdi:

```
int temp = score[3];
```

# Tabeller og minne

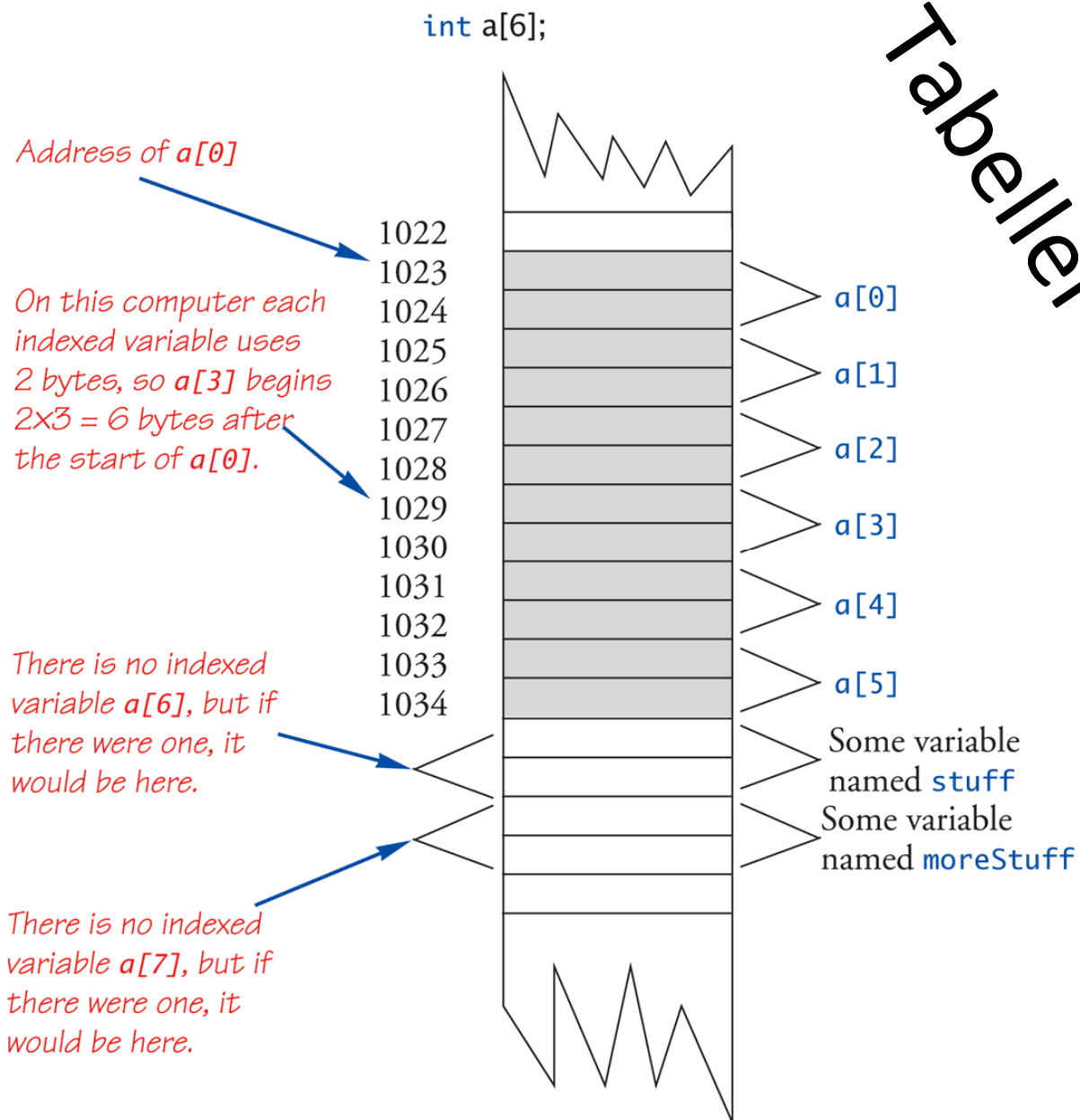
- En tabell bruker en blokk av minnet (definert størrelse)
- Datatypen forteller hvor mange bytes som må avsettes per element, tallet i klammer sier hvor mange elementer vi skal ha plass til

- Størrelsen må vi selv holde orden på

```
const int SIZE = 5;
int tab[SIZE] = {1, 4, 1, 2, 4};
int sum = 0;
for (int i = 0; i < SIZE; i++){
    sum += tab[i];
}
int gjennomsnitt = sum / SIZE;
```

- *PS! Bruk av konstant gjør det enkelt å tilpasse koden seinere til annen størrelse siden vi kun trenger å endre EN plass i koden.*

## Display 5.2 An Array in Memory



Tabeller og minne

# Tabeller og tilordning

- Må kopiere ”manuelt” mellom elementene

```
int tabA[5] = {1, 2, 3, 4, 5};  
int tabB[5];  
  
for (int i = 0; i < 5; i++){  
    tabA[i] = tabB[i];  
}
```

- Kan kopiere hele tabellen eller deler for eksempel ved hjelp av for-løkke
- Men pass på at du ikke leser eller skriver utenfor størrelsen til en av tabellene

# Tabeller og funksjoner



- Enkelt-elementer kan sendes som
  - call-by-value
  - call-by-reference
- Eller vi kan sende "hele" tabellen  
(i praksis adresse til første element)

```
// funksjonsprototyper
int sum(int a, int b);
void swapByReference(int& a, int& b);
void swapByPointer(int* a, int* b);
```

```
// bruke funksjonene på enkelt-elementer
int x = sum(tab[1], tab[2]);
swapByReference(tab[1], tab[2]);
swapByPointer(&tab[3], &tab[4]);
```

```
// funksjonsprototyp
int gjennomsnitt(int tab[ ], int size);
```

```
// bruke funksjonen
int y = gjennomsnitt(tab, 5);
```



# Informere om størrelse på tabellen ved funksjonskall



- Tabeller har en definert størrelse!
- «Udefinert» hva som skjer om du leser forbi, men ikke kompilersfeil!
- Vi må derfor informere funksjoner om størrelsen

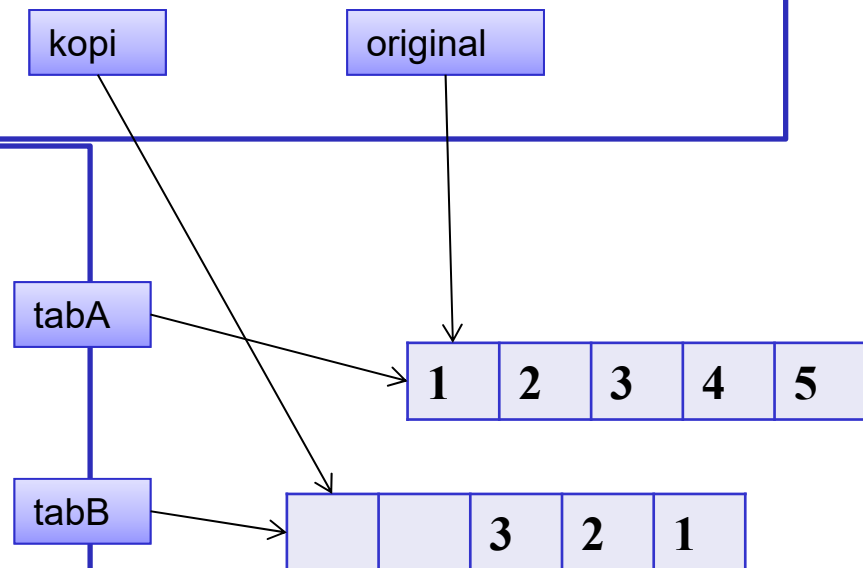
```
int gjennomsnitt (int tab[ ], int s);
```

```
const int SIZE = 5;  
int tab[SIZE] = {1, 4, 1, 2, 4};  
int a = gjennomsnitt(tab, SIZE);
```
- Ved hjelp av global konstant eller parameter i funksjonen

# Tabell-parameter er referanser

```
void reverser(const int original[ ], int kopi[ ], int size){  
    for (int i = 0, j = size - 1; i < size; i++, j--) {  
        kopi[j] = original[i];  
    }  
}
```

```
int main(){  
    int tabA[5] = {1, 2, 3, 4, 5};  
    int tabB[5] = { };  
    reverser(tabA, tabB, 5);  
    for (int i = 0; i < 5; i++){  
        cout << tabB[i] << " ";  
    }  
}
```



*Vi sender referanser til funksjonen reverser og endrer på tabellene vi deklarte i main*



# Tabell som returtype? **NEI!**

- Tabeller kan ikke brukes som returverdi
- Men vi kan løse dette ved hjelp av parameter for å returnere resultat
- Husk at tabell-parameter er referanser!

```
void reverser(const int original[ ], int kopi[ ], int size){  
    for (int i = 0, j = size - 1; i < size; i++, j--){  
        kopi[j] = original[i];  
    }  
}
```

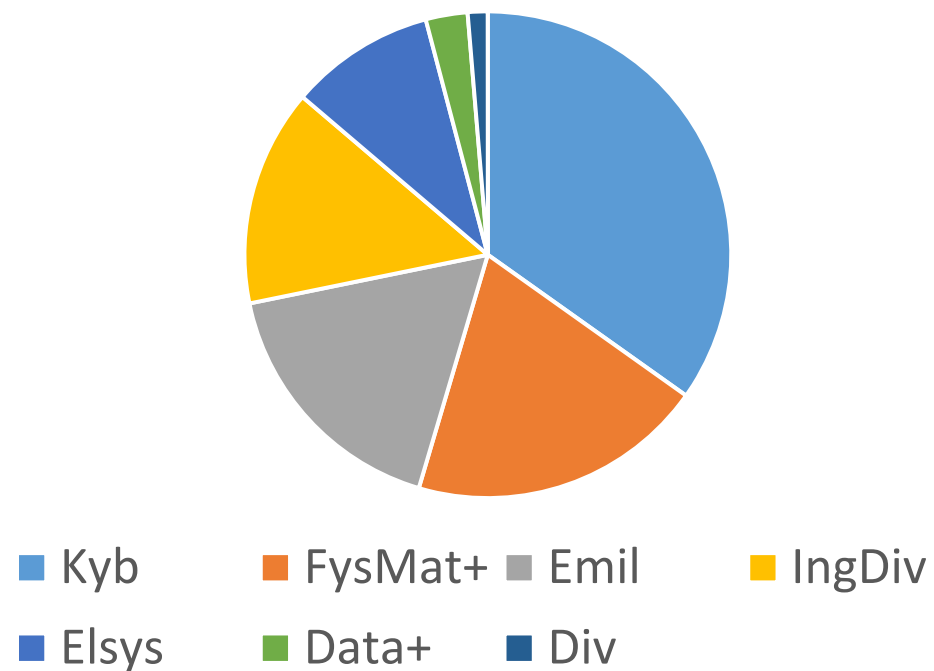
*const brukes for å sikre oss at original ikke endres i funksjonen, kopi er en tabell vi skal skrive til så den skal ikke være const*

# Litt administrativt

- Diskusjonsforum
  - Stor aktivitet
  - Mye hjelp
  - Ingen garantier for kvalitet, men mye veldig bra
  - En del er mer avansert enn det som kreves for A i faget
  - Er ikke pensum (eksamensrelevant)
  - Bruk det!
- Fordeling av studenter
- Referansegruppen
- Spørsmål ??

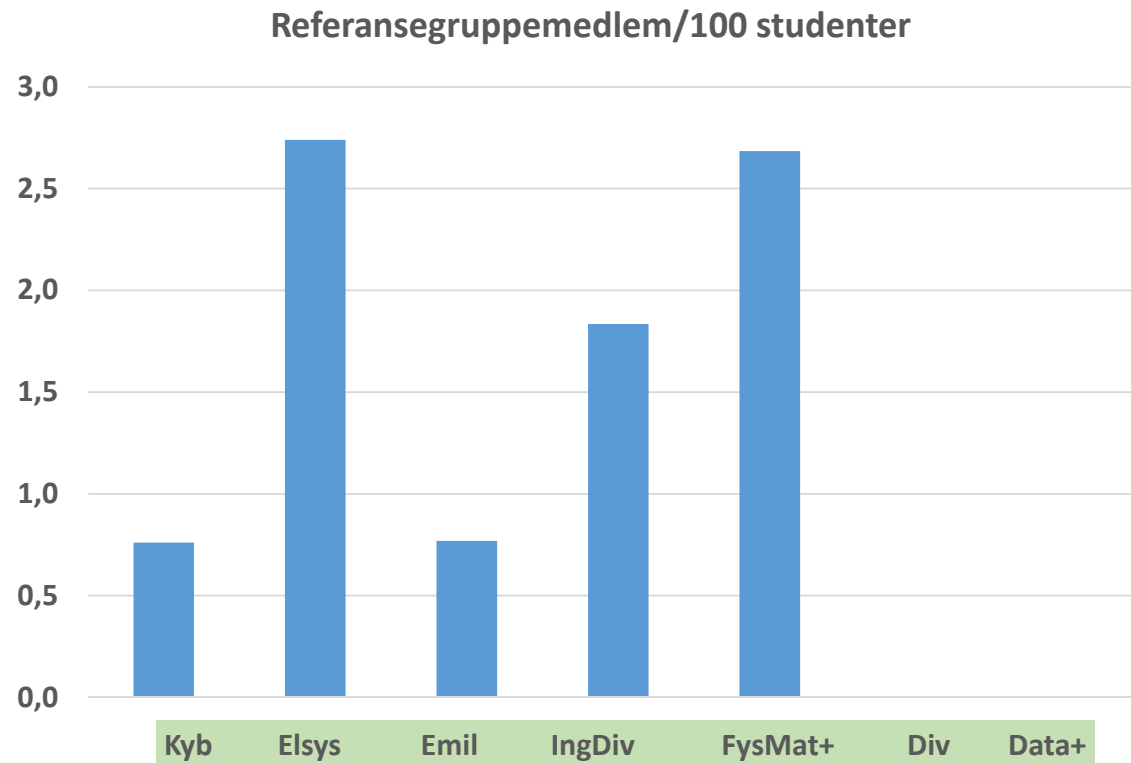
# 773 studenter

Fordeling studenter



# Referansegrupper

## 11 studenter (se Its'learning)



# Litt mer om pekere og referanser

- En tabell-variabel er en referanse
- Referanser og pekere er nesten det samme
- En referanse er en const peker
  - En referanse kan ikke endre hva den “peker” til
  - En peker kan endre hva den peker til
- En peker kan tilordnes en referanse
  - Men en referanse kan ikke tilordnes en peker

# Alternative funksjonsparameter for tabeller

```
void foo(int a[ ], int size){  
    for (int i = 0; i < size; i++) {  
        cout << a[i] << endl;  
    }  
}
```

```
void foo(int* a, int size){  
    for (int i = 0; i < size; i++) {  
        cout << a[i] << endl;  
    }  
}
```

*Disse funksjonene fungerer helt likt!  
Parameter for endimensjonale tabeller kan erstattes med peker.*



# Peker-aritmetikk

```
#include <iostream>
using namespace std;
```

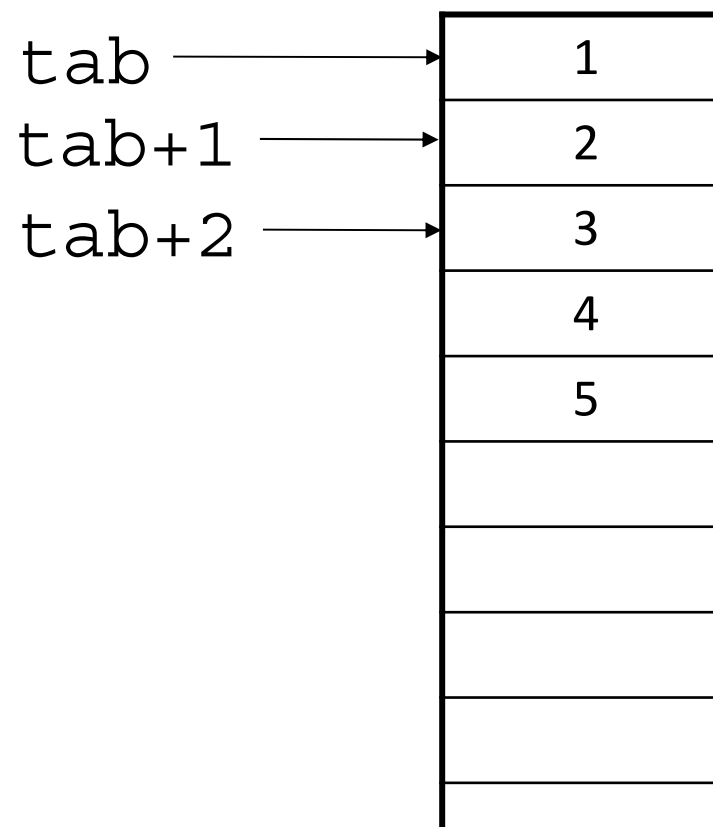
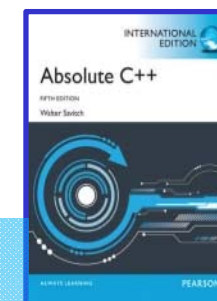
```
int main( )
{
    int tab[5] = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i++) {
        cout << tab[i] << endl;
    }

    for (int i = 0; i < 5; i++) {
        cout << *(tab + i) << endl;
    }

    for (int* i = tab; i < (tab + 5); i++) {
        cout << *i << endl;
    }
}
```

Side 472



Vi kan også bruke tab++  
for å flytte pekeren til  
neste element



# Flerdimensjonale tabeller

- En tabell kan ha flere dimensjoner
- Spesifiseres med [dim1][dim2]
- Gir oss en `int eks1[4][4];` todimensjonal tabell  
`eks1[0][0] = 5;`  
`eks1[2][3] = 8;`
- Kan ha så mange dimensjoner vi vil  
`int eks2[5][5][5];`  
`int eks3[4][4][4][10];`

5			
			8

# Flerdimensjonale tabeller og funksjoner

- Les og skriv slik som for endimensjonale tabeller  
MEN: kan **ikke** bruke `tab[][]` som parametertype
- For endimensjonale tabeller er det nok for kompilatoren å kjenne datatypen til hvert element
- For flerdimensjonale tabeller må kompilatoren også kjenne størrelsen på 2,3,... dimensjon
  - Derfor må vi for todimensjonale tabeller deklarere størrelsen av andre dimensjon: `int eks[100][400];`
  - `void doSomething(int tab[ ][400]);`
  - ellers kan ikke kompilatoren beregne adressen til elementene riktig!

# Må vi bruke eksakt størrelse?

- Fullt mulig å deklarere tabeller som er større enn det antallet elementer vi faktisk bruker, eks:

**const int MAX = 5000;**

*maks antall elementer vi har lov til å lagre = faktisk størrelse*

**int size = 5;**

*variabel som inneholder antallet plasser som er i bruk*

- size kan endres ved behov og vil være den verdien som vi bruker ved iterasjon
- ved endring av size for å tilpasse størrelsen til nye elementer må vi sjekke at ikke **size > MAX**

# Datatypen char

- Er en datatype for enkelttegn
- Bruker 1 byte
- Kan også brukes som “heltallstype”
  - 1 byte = 256 verdier

```
char a = 97;  
char b = 'a';  
cout << a << ", " << b << endl;
```

- Siden bokstaver “er tall” likhet og større/mindre enn, inkrementere osv.

# Tabeller av char (char[ ])

## C-strenger

- I programmeringsspråket C brukes tabeller av datatypen char for tekststrenger
- I C++ finnes det en klasse-type for strenger (string), men du kommer ofte borti de "gammeldagse" C-strenger
- Du må beherske begge og forstå forskjellen

# char[]

- Er i utgangspunktet en helt vanlig tabell som kan inneholde enkelttegn (char-verdier)

"tekst" lagres som sekvensen 't', 'e', 'k', 's', 't'

- Siden det ofte er vanskelig å vite den eksakte lengden av en tekststreng på forhånd, så benyttes spesialtegnet '\0' for å indikere slutten på strengen

't', 'e', 'k', 's', 't', '\0'

I praksis kan vi da ha char[ ]-variabler som kan inneholde strenger av variabel lengde -så lenge tabellen har plass til strengen + \0

# C-strenger i bruk (i)

- Vi kan deklarere og initialisere strenger **uten** å angi størrelse

```
char t1[] = "tekst";  
//opprettet en t[6] som inneholder 't', 'e', 'k', 's', 't', '\0'
```

- Eller vi kan angi størrelse for tabellen og initialisere med en streng som får plass

```
char t2[10] = "tekst";  
//opprettet en char t2[10] som inneholder 't' 'e' 'k' 's' 't' '\0' ' ' ' ' ' '  
//NB! alt etter '\0' er uinitialisert
```

- Lengden må være minimum antall tegn i selve tekststrengen + 1
  - kompilatoren sørger for å legge til termineringstegnet



## C-strenger i bruk (ii)

- Som for andre tabeller, kan vi ikke kopiere hele C-strengvariabelen til en annen med vanlig tilordning

```
char a[10] = "ole";  
char b[10] = "brumm";  
a = b; //Vil ikke kompilere:
```

- Men det finnes funksjoner som kan brukes i `<cstring>`

```
char a[10] = "ole";  
char b[10] = "brumm";  
strcpy(a, b);  
//a har nå samme innhold som b
```

- NB! Alle funksjoner i `<cstring>` er basert på at C-strengene avsluttes med `'\0'`

# <cstring> eksempler

Se for eksempel  
[www.cplusplus.com](http://www.cplusplus.com)  
for full oversikt

- Kopiering av strenger

`strcpy(a, b)` // kopierer fra streng b til a

`strncpy(a, b, 3)` // kopierer n antall tegn fra b til a

- Konkaterering

`strcat(a, b)` // konkatenerer b til a

`strncat(a, b, 5)` // konkatenerer n tegn fra b til a

- Søking

`strchr(a, 'x')` // finner indeks til første tegn som er x

- Finne lengden på en streng

`strlen(a)` // gir antallet tegn i selve strengen ('0' telles ikke med)

## C-strenger i bruk (iii)

- Vi kan enkelt skrive ut innholdet i en C-stringvariabel med insertion-operatoren

```
char a[10] = "ole";  
char b[10] = "brumm";  
cout << a << ", " << b << endl;
```

- Og vi kan behandle tegn for tegn

```
char name[] = "ole brumm";  
for (int i = 0, j = strlen(name) - 1; i < j; i++, j--){  
    swap(name[i], name[j]);  
}  
cout << name << endl;
```

## Hva med char[ ] hvor teksten ikke avsluttes med '\0'?

- Det er lov å lage tabeller for enkelttegn uten at teksten avsluttes med null-termineringstegnet

```
char alfabet[27];  
for(char i = 'a'; i <= 'z'; i++){  
    alfabet[i - 'a'] = i;  
}
```

- Da kan du IKKE bruke funksjonene beregnet for C-strenger
  - Fordi disse funksjonene bruker '\0' for å finne slutten på teksten og vil gi kjøretidsfeil hvis tegnet ikke finnes
  - Koden kompilerer og kan fungere greit, men vil av og til oppføre seg merkelig

# Alternativ til tabeller

- Tabeller er en lavnivå konstruksjon som du må kunne, men i praktisk programmering bruker vi ofte andre ting
- C++ biblioteket har en rekke "template-typer" som du etter hvert vil bli kjent med

# Eksempel

- ((Repetisjon swap ))
- Bruk av tabell
- Strenger