TDT4102

Prosedyre- og objektorientert programmering

Arrays (Tabeller)

Dagens forelesing

- Repetisjon
  - Funksjoner og parametere
  - Kahoot den 16/1
- Nytt stoff
  - Testing
  - Arrays (tabeller)
    - Syntaks og bruk
    - Kapittel 5
  - Datatypen char og C-strenger
- Husk (studieteknikk)
  - Kode-eksempler og slides legges ut, du trenger ikke skrive av underveis

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 (dvs. peker som parameter)

Kahoot test 1 (16/1-2017)

- Resultater (280 deltakere)
  - 1 x 10 rette
  - 1 x 9 rette
  - 6 x 8 rette
  - Ca. 100 hadde 5 el. flere rette
- Passe vanskelighetsgrad?
  - Stor forskjell på kahoot i timen og eksamen 4 timer!
- Kahoot og kode lagt på Its'learning

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

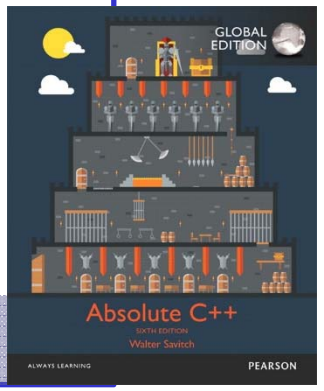
```
#include <iostream>
using namespace std;

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;
}
```

Side 457



Swapping – med referanser

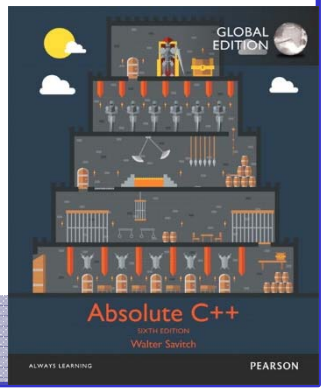
```
#include <iostream>
using namespace std;

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 178 - 194



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. skriving «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

Forskjellige måter å teste på

- Enkel testing mens du programmerer:
  - Prøv ut programmet ditt med forskjellige verdier
  - Skriv ut til cout (eller cerr) verdier av variabler før og etter funksjonskall etc.
  - Bruke debugger; sjekke hvordan variabler endres og hvordan funksjoner kalles
- Vi kan også teste mer systematisk:
  - Vi tester hver "enhet" vi lager separat (i praksis testing av funksjonene)
  - Vi tester på betingelser før kall og etter kall til en funksjon

Regler og betingelser

- Definere regler for dine funksjoner
  - Hva er forventet resultat for gitte verdier?
  - Hva er forventet resultat for "grenseverdier" eller spesialtilfeller?
- Reglene kan testes med betingelser
  - Hvilke betingelser må gjelde før en funksjon kalles
  - Hvilke betingelser skal gjelde etter at den er kalt
- Testing basert på regler og betingelser er en metode for systematisk testing

assert

- I C++ kan vi bruke en assert makro for å teste koden vår
- En assertion (påstand) er et statement som enten er true eller false
- Brukes for å sjekke betingelser (for alle reglene vi har)
  - Preconditions & Postconditions -> før og etter kall til funksjonen
- Syntaks: **assert(<assert_betingelse>);**
  - Ingen returverdi
  - Evaluerer assert_betingelsen
  - Avslutter programmet hvis false, fortsetter hvis true
- Predefined i biblioteket <cassert>
  - Makroer brukes likt med funksjoner
  - Men ved kompilering (preprosessering) vil makroen erstattes med makrokoden

Bruk av assert makroen

- assert bruker du som en vanlig funksjon
- men assert-statements kan "slåes" av og på med preprosessor direktivet #define NDEBUG
  - #define NDEBUG
    #include <cassert>

    assert makroene er AV og programmet kompileres uten at disse er med
  - //#define NDEBUG
    #include <cassert>

    assert makroene er PÅ og programmet vil avslutte hvis et assert-statement => false

    utelater vi #NDEBUG helt vil koden kompileres MED assert

Eksempel

- assert.cpp
- guessANumber.cpp
  - Bruk av srand
  - Feil inntasting av tallverdi

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 deklareres 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 :

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

initialiserer alle fem elementene

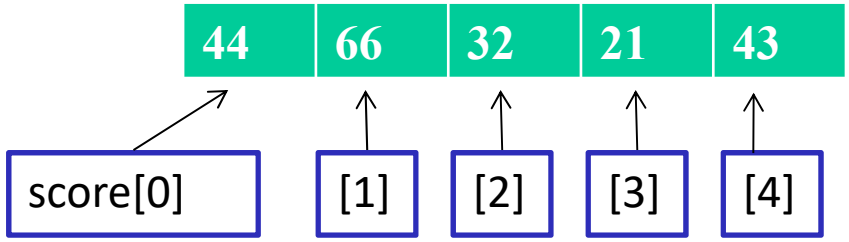
initialiserer bare de tre første elementene, mens de to siste har ukjent verdi

initialiserer alle elementer til 0

Tabeller:
Lese og skrive verdier

- Velg element med indeksoperatoren [ ]

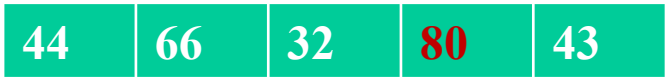
```
int score[5] = {44, 66, 32, 21, 43};
```



- **NB! indeksen begynner på 0!**

- Skrive verdi:

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



- Lese verdi:

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

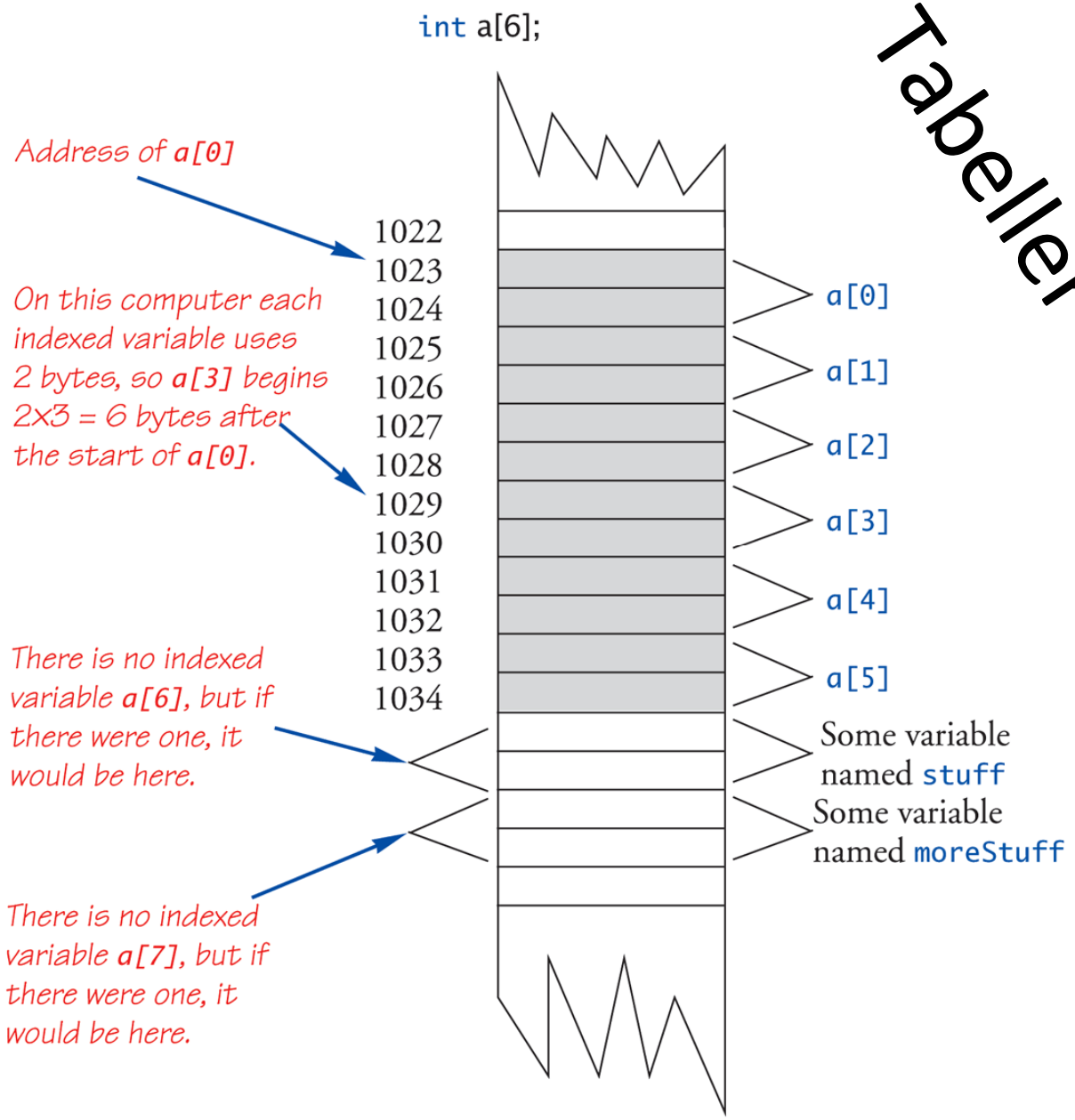
Tabeller og minne

- 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 enkeltelementer
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 kompileringsfeil!
- 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

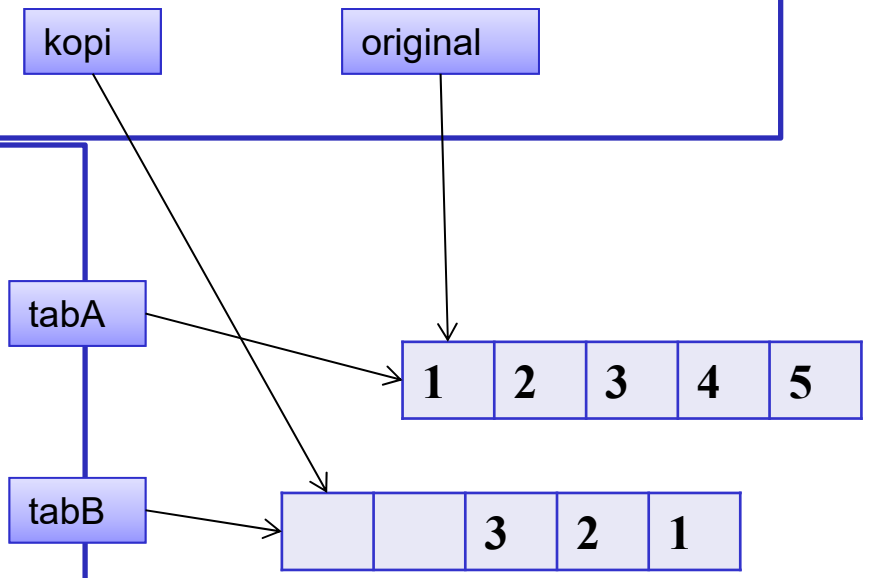
Eksempel

- Reverser tabell

Tabell-parameter er referanser

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

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

Litt administrativt

- Diskusjonsforum
  - Bra aktivitet
  - Mye og rask hjelp
    - Gj.snittlig svartid 11 minutter!
- Fordeling av studenter
- Referansegruppen
- Spørsmål ??

Referansegruppe

- 8 personer har meldt seg på liste men vi mangler
  - en person fra el.sys (ca. 90 studenter)
  - en fra Ind-øk (ca. 60 studenter)

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 funksjons-parameter 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

Side 478-479

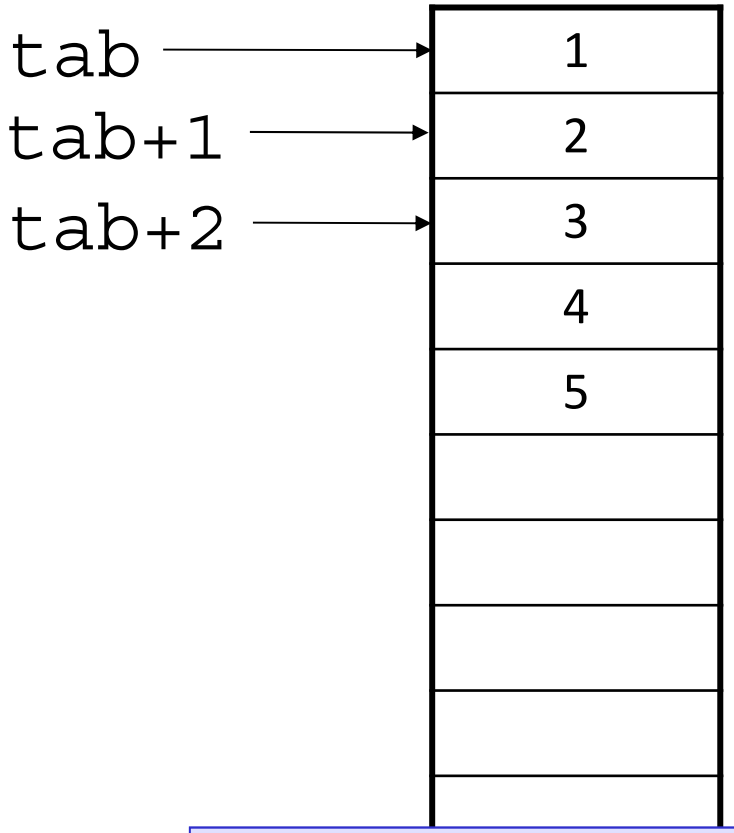
```
#include <iostream>
using namespace std;

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;
    }
}
```



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 todimensjonal tabell

```
int eks1[4][4];
```

```
eks1[0][0] = 5;
eks1[2][3] = 8;
```

5			
			8

- Kan ha så mange dimensjoner vi vil

```
int eks2[5][5][5];
int eks3[4][4][4][10];
```

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" kan vi også teste for 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";
//oppretter 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";
//oppretter 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
for full oversikt

22 ulike funksjoner

Safe versjoner i MS-VS:
* strcpy_s
* strncpy_s
* strcat_s
* strncat_s

- Kopiering av strenger

  strcpy(a, b) *// kopierer fra streng b til a*
  strncpy(a, b, 3) *// kopierer n antall tegn fra b til a*

- Konkatenering

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

- Søking

  strchr(a, 'x') *// finner peker til første tegn som er 'x'*

  strstr(a, "abc") *// finner peker til første forekomst av "abc"*

- 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 engelskAlfabet[27];
for (char c = 'a'; c <= 'z'; c++) {
    engelskAlfabet[c - 'a'] = c;
}
// cout << engelskAlfabet << endl;
// skriver ut for mye, mangler 0-terminering, kjøretidsfeil
engelskAlfabet[26] = '\0';
cout << engelskAlfabet << endl;
```

- 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