TDT4102
Prosedyre- og objektorientert programmering, Øvingsforelesning veke 4, 25. januar

Sivert Krøvel
sivertkr@stud.ntnu.no
Anna Rekdal
annalr@stud.ntnu.no
Torje Hoås Digernes
torjehoa@stud.ntnu.no

Agenda

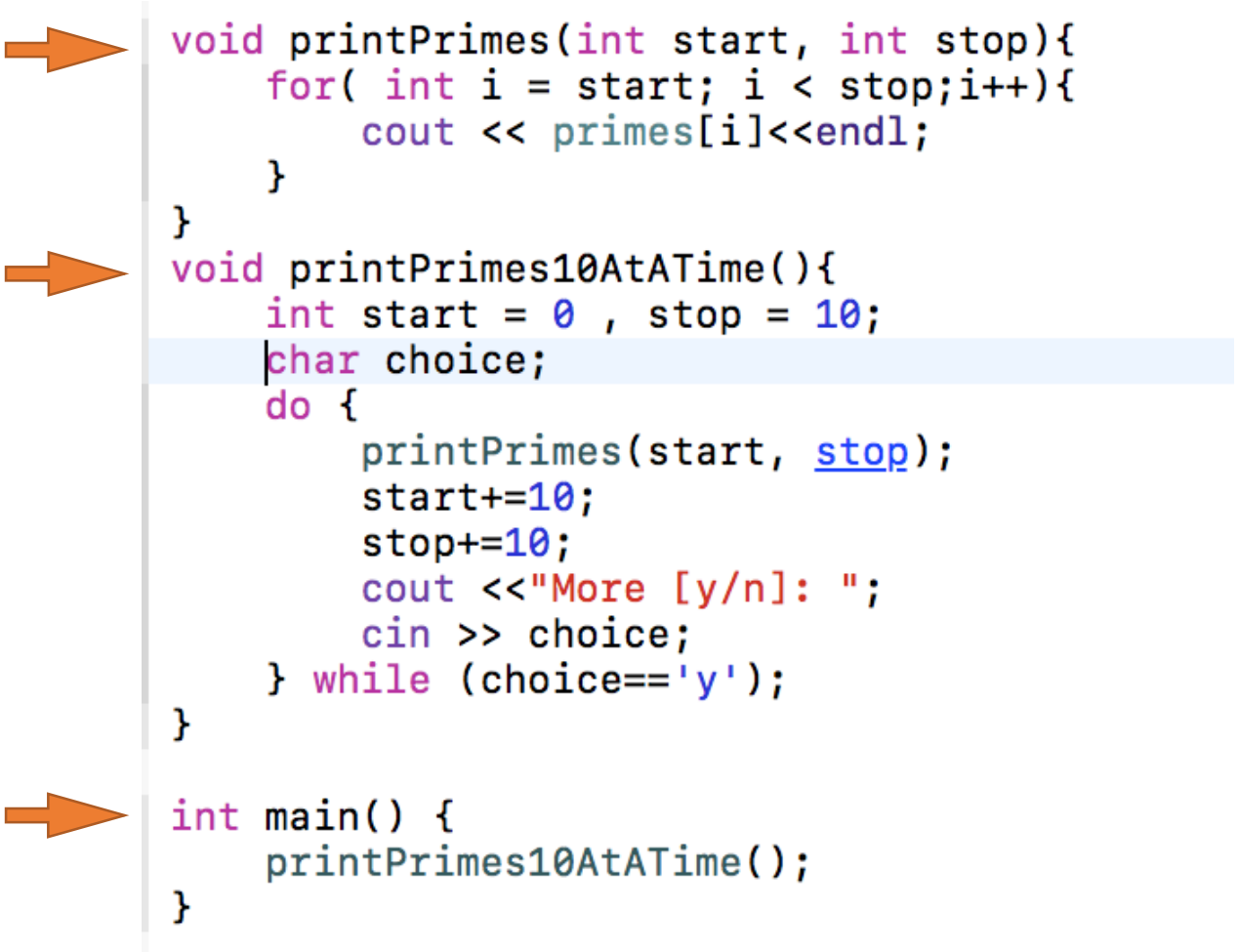
- Løkker og andre kontrollstrukturar
- Kort oppgåve
- Meir om funksjonar
- Kort om peikarar
- Kort om headerfiler
- Oppgåver

Flow of control

Programmet kjører linje for linje, om ikke:

- vi kaller funksjoner
- vi møter flow of control statements

Når ein startar programmet



```
void printPrimes(int start, int stop){
    for( int i = start; i < stop;i++){
        cout << primes[i]<<endl;
    }
}
void printPrimes10AtATime(){
    int start = 0 , stop = 10;
    char choice;
    do {
        printPrimes(start, stop);
        start+=10;
        stop+=10;
        cout <<"More [y/n]: ";
        cin >> choice;
    } while (choice=='y');
}

int main() {
    printPrimes10AtATime();
}
```

Flow of control statements endrar rekkjefølgja av kodelinjene

- switch
- if else if else
- for
- while
- do while

Switch

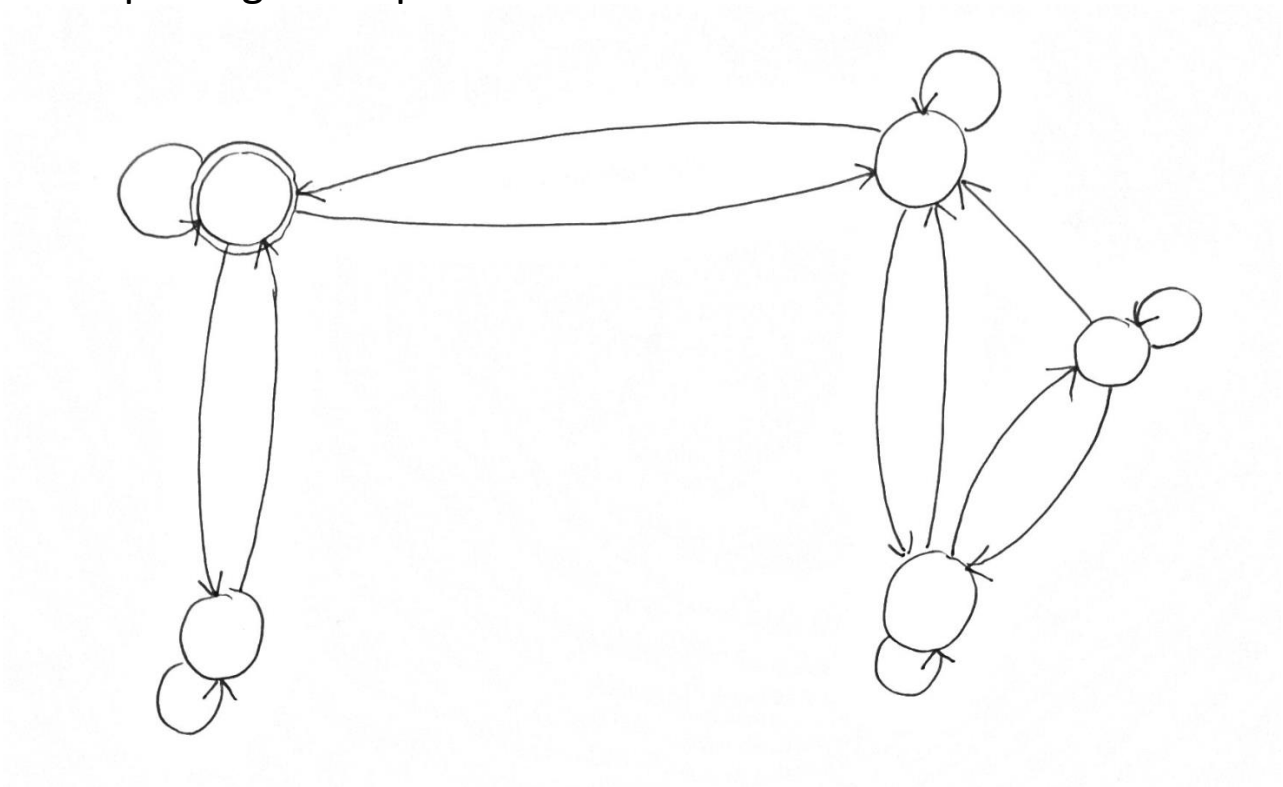
- for å velgje mellom forskjellige tilfelle
- virkar kun på heiltalstypar
- typisk for å velgje kun ein ting, hugs break

Switch - døme

```
void menu(){
    int choice;
    do {
        printmenu();
        cin >> choice;
        switch (choice) {
            case 1:
                printPrimes10AtATime();
                break;
            case 2:
            {
                int number;
                cout <<"Which number should we test? ";
                cin >> number;
                isIn_2_or_4_TimesTable(number);
                break;
            }
            case 3:
                sortBox();
                break;
```

Switch - tilstandsmaskin

Ein case per utgåande pil



Eg sa break?

- del av flow of control
- om du er inne i ein switch, eller i ei løkke
- Bryt ut av kontrollstrukturen

Meny utan break

```
void menu_trap(){
    int choice;
    do {
        printmenu();
        cin >> choice;
        switch (choice) {
            case 1:
                printPrimes10AtATime();
            case 2:
            {
                int number;
                cout <<"Which number should we test? ";
                cin >> number;
                isIn_2_or_4_TimesTable(number);
            }
        }
    }while(choice != 0);

}
```

if - else if - else

- if sjekkes alltid, då blir kroppen kjørt
- else if sjekkes kun om ikkje if-en, og ingen if else over heller traff, og bare om den er sann kjøres kroppen
- else kroppen kjøyrast kun om ingen andre gjorde noko

if - else if - else

```
void sortBox(){
    cout << "write height [m] and the largest og width and depth[m]" <<endl;
    cout << "also, I know nothing about sorting boxes :S"<<endl;
    double height, width;
    cin >> height>>width;
    if ( height<0.3 && width < 0.4){
        cout << "The box is in shelf A"<< endl;
    }else if( height < 0.9 && width < 1.2){
        cout << "The box is in shelf B"<<endl;
    }else{
        cout << "The box is outside"<<endl;
    }
}
```

```
void isIn_2_or_4_TimesTable(int number){
    if (number%2==0){
        cout <<number << " is in the 2 times table"<<endl;
    }
    if (number%4==0){
        cout <<number << " is in the 4 times table"<<endl;
    }
}
```

for-løkker

- Vanleg om me veit kor mange iterasjonar me skal gjere

for - døme

```
double calcAverage( double table[], int length ){
    double sum = 0;
    for (int i = 0; i < 25; i++) {
        sum+= table[i];
    }
    double average = sum;
    average/=length;
    return average;
}
```

while-løkker

- Vanleg å bruke om ein ikkje veit kor langt ein må.

While - døme

```
bool isPrime(int number){
    int counter = number - 1;
    bool prime = true;
    while (prime && counter > 1) {
        if ( number % counter == 0){
            prime = false;
        }
        counter--;
    }
    return prime;
}
```

Eg sa break? igjen?

- Virkar på samme vis
- Bryt ut av innerstekontrollstruktur
- Virkar på for-løkker og

do while

- Om me har noko som skal gjerast minst ein gong kan do while vere elegant

do while- døme

```
void printPrimes(int start, int stop){
    for( int i = start; i < stop;i++){
        cout << primes[i]<<endl;
    }
}
void printPrimes10AtATime(){
    int start = 0 , stop = 10;
    char choice;
    do {
        printPrimes(start, stop);
        start+=10;
        stop+=10;
        cout <<"More [y/n]: ";
        cin >> choice;
    } while (choice=='y');
}
```

Nøsting av løkker

```
void doubleLoop(){
    for (char letter = 'a'; letter <'d' ; letter++) {
        for (int I = 0 ; I<3U; I++) {
            cout << letter << I<<endl;
            usleep(250000);
        }
    }
}
```

Litt om scope

- Ein variabel som er deklarert inne i ein funksjon (også i parameterlista) er *lokal* for funksjonen. Den eksisterer ikkje utanfor funksjonen.
- Navna til dei lokale variablane har ingen samanheng med variabelnavn på utsida av funksjonen
- Kun returverdien vert sendt vidare når funksjonen er ferdig.
- NB! Det går an å endre på parametrane til ein funksjon ved å bruke call-by-reference eller peikarar

Å kalle funksjonar

- Å kalle ≈ å bruke
- Alle funksjonar kan kalle på andre funksjonar, både eigendefinerte og innebygde
- To «typar» funksjonar, ein som returneren ein verdi og ein som ikkje gjer det

Kva skjer ved eit funksjonskall?

- Argumenta vert kopierte (og deretter behandla som lokale variablar)
- Kontrollflyten vert overført til funksjonen
- Funksjonen kjører
- Returverdien (viss den fins) vert returnert til kallande funksjon
- Dei lokale variablane «forsvinn»

Funksjonar som returnerer noko

- Returverdien vert returnert til *kallande funksjon*
- Evaluerer til ein verdi, bestemt av returtypen
- Kan dermed brukast som argument til andre funksjonar, i tilordningar (på høgre side, i første omgong), som betingelse i if-setningar osv…

Void funksjonar

- Enkeltståande statement
- Returnerer ikkje ein verdi, kan dermed ikkje brukast der programmet forventar ein verdi
- Kan altså ikkje brukast i tilordningar osv.
- Betyr ikkje at ingenting kan endrast!

Funksjonskall, illustrasjon

```
bool isEven(int num);
int abs(int num);
std::string getNameFromUser();

void printFormattedName();
void swap(int* a, int* b);

int testfunksjon(){

    int a;
    std::string name;

    if (isEven(a)) {
        int b = abs(a);
        name = getNameFromUser();
    }


}
```

Kvifor lagre prosjektet i fleire filer

- Headerfiler og Sourcefiler
- Som regel fleire filer i kvart prosjekt.
- Ryddegare
- Slepp å tenke på kva rekkefølgje ting kjem i

Funksjonsprototypen

- Det er ikkje mykje som skal til får programmet veit at ein funksjon eksisterar:
- type funksjonsnamn(type argumentnamn);
- Treng tilgong til denne for å kalle funksjonar i andre filer

Syntaks

- ..cpp er sourcefile og .h eller .hpp er headerfil
- #include <iostream>
- #include "dittFilnamn.h"
- Funksjonsdeklarasjon i headerfila

Døme med fleire filer

Peikarar

int antal = 8;

- Variabelen antal blir tildelt ei adresse i minnet, t.d. 0xFF000000
- Det blir allokert 4 byte
- Den binære verdien til 8 blir lagra i minneadressa til «antal»

Kva skjer?

- PC'en gjer plass til ein **int** i minnet
- Variabelnamnet «antal» blir bunde til ei minneadresse
- Så blir *verdien* av minnet sett til 8.

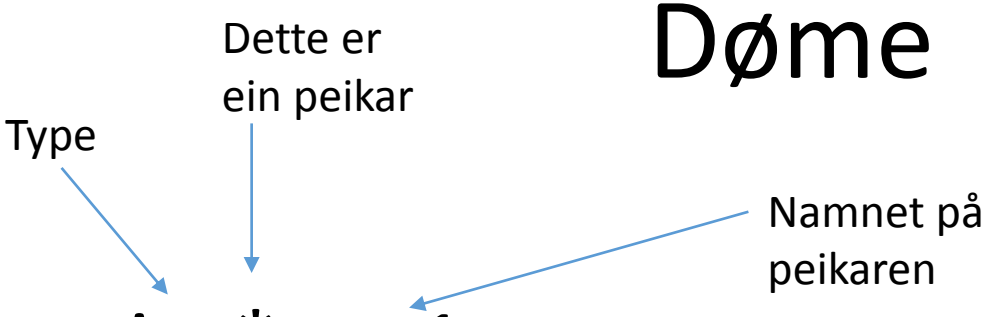
Peikaren er ein spesiell type variabel

- Peikaren held ingen verdi sjølv, han berre peikar vidare på ein annan variable, og *denne* variabelen held ein verdi.
- Vi brukar peikaren til indirekte å få tilgong til variabelen som blir peika på.

Korleis gjer ein det i praksis?

- Vi lagrar minneadressa til variabelen som vi peikar på.
- For å aksessere den «påpeika» variabelen, vandrar vi langs pila. Dette blir kalla å *dereferere* peikaren.

Døme



- int* ptr1;
- int *ptr2;
- double* ptr2;
- int* p=NULL;

Peikarar kan m.a. brukast til å...

- «returnere» fleire varaiblar frå éin funksjon
- Oppdaterere verdiar slik at dei forblir endra etter at funskjonen er ferdig
- Mm (kjem meir seinare i faget)

Korleis finne adressa til ein peikar?

&

Dereferering

- Vi kan lese verdien til ein peikar ved å bruke *
- *ptr=5;

Call-by-value VS Call-by-pointer

- Kva er forskjellen?

```
void increment_by_value( int value){
        value++;
}

void increment_by_pointer( int* value){
        (*value)++;
}
```

Call-by-value VS Call-by-pointer

```
int main() {
int val=0;
cout<<val<<endl;
increment_by_value( val );
cout<<val<<endl;
increment_by_pointer( &val );
cout<<val<<endl;
}
```

Døme med «retur» av fleire variablar v.h.a peikarar

```
void changeValuesOfPointers(double* a,
double*b) {

*a = 1.618033989;

*b = 3.141592658;

}
```

For å teste funksjonen

```
Void main() {
double a = 5;
double b = 8.6;
changeValuesOfPointers(&a, &b);
std::cout << "Etter:   A= " << a << " ,    B=
" << b << std::endl;
}
```