TDT4102
Prosedyre- og objektorientert programmering, Øvingsforelesning veke 5, 1. februar

Sivert Krøvel
sivertkr@stud.ntnu.no

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver

Tilfeldige tal

- I C++ er det to måtar å generere tilfeldige tal
- Vi bruker «gamlemåten» i denne øvinga, same metode som i C
- I cstdlib finn vi funksjonen rand()
- Returnerer eit (pseudo)tilfeldig tal mellom 0 og RAND_MAX (ein konstant)

Tilfeldige tal

- Ikkje eigentlig tilfeldig, «pseudoramdom»
- Ei rekke med tal, generert
- Basert på eit «frø» (seed)
- Same seed gir same rekke, kvar gong
- Vi må sette eit nytt frø for å få ei ny rekke
- Bør gjerast éin gong per program

Tilfeldige tal

- Funksjonen srand() set seed for rand-funksjonen
- For å få ei ny talrekke treng vi eit nytt seed for kvar gong vi kjører programmet
- Vanlig å bruke time()-funksjonen som seed. Den returnerer antal sekund sidan nyttår 1970 (frå ctime-biblioteket)

Tilfeldige tal

```
void printRandom() {
    std::cout << std::rand()
        << std::endl;
}

int main() {
    srand(time(nullptr));
    for (int i = 0; i < 10; ++i) {
        printRandom();
    }

    return 0;
}
```

Demonstrasjon

Tilfeldige tal

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver

Flyttal, presisjon

- I datamaskiner er alle tal representerte binært (1 og 0)
- Avgrensa nøyaktigheit medfører avrundingsfeil
- Ikkje alle verdiar kan representerast med det antal bits vi har tilgjengelig
- Flyttal i C++: float, double, long double

IEEE Standard 754

- Vanlig måte å representere flyttal i datamaskiner
- Vitenskapelig notasjon:
  $(-1)^{SIGN} x$ MANTISSA x $2^{BIAS+EXPONENT}$
- Forskjellige variantar, ulik presisjon (32 bit, 64 bit)
- Meir informasjon i boka/på internett

0.1+0.2 == 0.3
true or false?

```
int main() {
    cout << "0.1+0.2 == 0.3: " <<
        ((0.1 + 0.2) == 0.3 ? "true" : "false")
        << std::endl;
}
```

0.1+0.2 == 0.3
true or false?

```
int main() {
    cout << "0.1+0.2 == 0.3: " <<
        ((0.1 + 0.2) == 0.3 ? "true" : "false")
        << std::endl;
}
```

```
0.1+0.2 == 0.3: false
Trykk en tast for å fortsette...
```

Korleis samanlikne to flyttal?

Som de har sett, kan vi ikkje alltid sjekke om flyttal er heilt like. Korleis kan vi komme rundt dette? 1-2 minutt tenkepause, diskuter gjerne med naboen

Demonstrasjon

Flyttal

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver

Variablar i minnet

- Ein variabel er lagra ein plass i minnet, og den staden har ei bestemt adresse
- Adressa er eit tal, som regel representert ved ein heksadesimal verdi (t.d. 0x0AD5)
- Minneområdet vert allokert når variabelen vert deklarert, og frigitt når den går ut av scope

Variablar i minnet

Dette er *adressa* til variabelen *a*

...	...
0x0AB3	??
0x0AB4	5
0x0AB5	??
0x0AB6	??
0x0AB7	??
...	...

Her ligg heiltalsvariabelen *a*


```
int a = 5;
```

NB! Addressene i dette dømet er vilkårlig valde, i datamaskina di er dei lenger

Kva er ein peikar?

- Ein variabel som «peikar til» ein plass i minnet
- Inneheld ei minneadresse

Peikarvariablar

```
int a = 5;
```

```
int* a_ptr = &a;
```

Variabelen *a*, med adresse

a

...	...
0x0AB3	??
0x0AB4	5
0x0AB5	??
0x0AB6	??
0x0AB7	??
...	...

...	...
0x0C06	??
0x0C07	0x0AB4
0x0C08	??
0x0C09	??
0x0C0A	??
...	...

a_ptr

Denne variabelen, *a_ptr*, inneheld *adressa* til *a*

Legg merke til at denne kan ligge ein heilt annan stad i minnet

Peikarsyntaks

	*	&
Deklarasjon	Deklarerer ein variabel av peikartype	Deklarerer ein referanse
Bruk/elles (operator)	Dereferer ein peikar (aksesserer minnet den peikar til)	Tek addressa til ein variabel (lager ein peikar til variabelen

Referanse vs peikar

- Litt forenkla:
- Ein **peikar** «peikar til» variabelen
- Ein **referanse** *er* variabelen

```
int main() {
    int a = 5;
    int& a_ref = a;
    int* a_ptr = &a;

    cout << "a: " << a << "\na_ref: " << a_ref
        << "\na_ptr: " << a_ptr << "\n&a: " << &a
        << "\n&a_ref: " << &a_ref << "\n*a_ptr: "
        << *a_ptr << endl;

    return 0;
}
```

```
a: 5
a_ref: 5
a_ptr: 0036F7C8
&a: 0036F7C8
&a_ref: 0036F7C8
*a_ptr: 5
```

Å bruke peikarar

- Peikarar har fleire nyttige bruksområde
- Lage tabellar med variabel storleik, gi funksjonar tilgang til å endre på variablar mm.
- Vi kjem tilbake til dei fleire gonger gjennom semesteret

Å bruke peikarar

- I denne øvinga brukar vi peikarar til å lagre «returverdiar»
- Vi gir funksjonar anledning til å *endre på* variablar utan å direkte ta dei inn som argument

Å bruke peikarar

- Funksjonen tek inn addressa gjennom ein peikar, og endrar så på variabelen ved å *derefere* peikaren
- Nyttig for å «returnere» fleire verdiar frå ein funksjon (jf *getVelocityVector*-funksjonen i øving 3)

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver

Tabellar

- Ein tabell er eit samanhengande minneområde
- Storleiken må vere bestemt når programmet *kompilerer*, dvs før det køyrer (iallefall nokre veker til)

Tabellar

```
int tabell[4];
```

…	…
0x0A3C	??
0x0A3D	tabell[0]
0x0A3E	tabell[1]
0x0A3F	tabell[2]
0x0A40	tabell[3]
…	…

Tabellar

- Ein tabellvariabel er eigentlig ein referanse til det første elementet
- Indekseringsoperatoren, [], legg deretter til indeksen til adressa for å finne rektig element
- Derfor gir det meining å starte med indeks 0

C-strengar

- Ein c-streng er ein tabell med *char*
- Sluttar med ‘\0’
- Kan brukast direkte i std::cout
- Mange funksjonar som manipulerer c-strings i biblioteket <cstring>

Agenda

- Tilfeldige tal
- Flyttal (double...)
- Kort oppgåve
- Meir om peikarar
- Tabellar og c-strings
- Oppgåver