

# Minni og bendar

## Gagnaskipan 2015

Hjalti Magnússon (hjaltim@ru.is)



HÁSKÓLINN Í REYKJAVÍK  
REYKJAVÍK UNIVERSITY

11. janúar 2015

- 1 Minnismódelið
- 2 Bendar
- 3 Tilvísunarbreytur
- 4 Færíbreytur
- 5 Minnisleki

- 1 Minnismódelið
- 2 Bendar
- 3 Tilvísunarbreytur
- 4 Færíbreytur
- 5 Minnisleki

# Minnismódelið í C++

Minnið sem C++ notar má skipta í þrjá hluta

- Static minni
- Stafli (e. stack)
- Hróga, kös (e. heap)

# Static minni

- Geymir „global“ breytur
- Stærð ákvörðuð á þýðingartíma
- Skoðum það ekki nánar

# Staflí

- Geymir gögn og lýsigögn (e. metadata) allra falla sem búið er að kalla á, t.d.
  - Breytur
  - Færibreytur
  - Skilagildi
- Gögn sem tengjast hverju falli kallast kvaðningarfærsla (e. activation record)
- Stærð kvaðningarfærsla er ákvörðuð á þýðingartíma

# Hrúga

- Stórt minnissvæði þar sem minni er úthlutað á kviklegan hátt (e. dynamically)
- Minni úthlutað á keyrslutíma
- Stýrikerfi heldur utan um hvaða hluta minnisins við erum að nota
- Við verðum að sjálf að segja stýrikerfinu að við séum hætt að nota tiltekna hluta minnisins

# Yfirlit

1 Minnismódelið

2 **Bendar**

3 Tilvísunarbreytur

4 Færíbreytur

5 Minnisleki



# Bendar

- Allar breytur (þar með talið bendar) fá úthlutað minni á staflanum
- Breytur innihalda **gildi** en bendar innihalda **minnisföng**
- Allir bendar eru jafn stórir
- Ef búinn er til bendir með kalli á [new](#), er það minni sem bendirinn bendir á tekið frá á hrúgunni
  - Bendirinn sjálfur er geymdur á staflanum

# Dæmi

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

int main() {
    int *a = new int(3);
    cout << *a << endl;
    return 0;
}
```

pointer1.cpp

# Týpur benda

- `new int`, `new int[30]`, `new string`, `new int*` (og allt sem kemur út úr `new`) skila sama fyrirbærinu: bendi
- Týpur benda segja C++ stærðina á þeim minnisblökkum sem við erum að vinna með
- `int a* = new int[30]` tekur frá  $30 \cdot 4$  bæti í minni (og skilar bendi á fremsta minnishólfið)
- `a[4]` er breytt í `*(a + 4)` og skilar okkur fimmta stakinu í `a`
  - Hvað gerir `a + 4`?
  - Ef `a` hefði verið `double`-bendir, hefði `a + 4` skilað sama minnisfangi?

# Dæmi

```
int *a = new int[10];
for(int i = 0; i < 10; i++) a[i] = i;
for(int i = 0; i < 10; i++)
    cout << a[i] << endl;

void* x = a;
for(int i = 0; i < 5; i++)
    cout << ((long*)x)[i] << endl;
for(int i = 0; i < 20; i++)
    cout << ((short*)x)[i] << endl;
for(int i = 0; i < 40; i++)
    cout << ((int)((unsigned char*)x)[i]) << endl;

double *y = (double*)a;
for(int i = 0; i < 5; i++)
    cout << y[i] << endl;
```

pointer2.cpp

# Yfirlit

- 1 Minnismódelið
- 2 Bendar
- 3 Tilvísunarbreytur**
- 4 Færíbreytur
- 5 Minnisleki

# Tilvísunarbreytur

- Tilvísanir búa til *alias* á breytur
  - Tvö nöfn á **sömu** breytunni
- Notum tilvísanir aðallega í færribreytum

# Dæmi

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

int main() {
    int b = 3;
    int &a = b;
    cout << a << " " << b << endl;
    a = 4;
    cout << a << " " << b << endl;
    b = 12;
    cout << a << " " << b << endl;
    return 0;
}
```

reference.cpp

# Yfirlit

- 1 Minnismódelið
- 2 Bendar
- 3 Tilvísunarbreytur
- 4 Færíbreytur**
- 5 Minnisleki



# Pass-by-value

- Allar færibreytur í C++ eru pass-by-value
- Fallið vinnur með afrit af breytunni okkar

# Dæmi

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

void fun(int x) { // x is a pass-by-value parameter
    x = 15;
    cout << "fun: " << x << endl;
}

int main() {
    int x = 3;
    cout << "main: " << x << endl;
    fun(x);
    cout << "main: " << x << endl;
    return 0;
}
```

passbyvalue1.cpp

# Pass-by-reference

- Hægt er að gera færibreytu pass-by-reference (tilvísunarfæribreyta)
  - Fallið fær alias
  - Fallið vinnur með nákvæmlega þá breytu sem við sendum inn

# Dæmi

```
void fun(int& a) {  
    a = 15;  
    cout << "fun: " << a << endl;  
}  
  
int main() {  
    int x = 3;  
    cout << "main: " << x << endl;  
    fun(x);  
    cout << "main: " << x << endl;  
  
    return 0;  
}
```

passbyref1.cpp

# Pass-by-value bendar

```
void fun(int *x) {  
    *x = 15;  
    cout << "fun: " << *x << endl;  
}  
  
int main() {  
    int *x = new int(3);  
    cout << "main: " << *x << endl;  
    fun(x);  
    cout << "main: " << *x << endl;  
  
    return 0;  
}
```

passbyvalue2.cpp

# Pass-by-value bendar

```
void fun(int *x) {  
    *x = 15;  
    cout << "fun: " << *x << endl;  
    x = NULL;  
}  
  
int main() {  
    int *x = new int(3);  
    cout << "main: " << *x << endl;  
    fun(x);  
    cout << "main: " << *x << endl;  
  
    return 0;  
}
```

passbyvalue3.cpp

# Pass-by-reference bendar

```
void fun(int *ampx) {  
    *x = 15;  
    cout << "fun: " << *x << endl;  
    x = NULL;  
}  
  
int main() {  
    int *x = new int(3);  
    cout << "main: " << *x << endl;  
    fun(x);  
    cout << "main: " << *x << endl;  
  
    return 0;  
}
```

passbyvalue3.cpp

# Yfirlit

- 1 Minnismódelið
- 2 Bendar
- 3 Tilvísunarbreytur
- 4 Færibreytur
- 5 Minnisleki**



# Minnismeðhöndlun

- C++ sér um allt minni á staflanum (þ.m.t. tiltekt)
- C++ veit bara hvort að minni sér frátekið eða ekki á staflanum
  - Við verðum að sjá um að skila minni sem við erum hætt að nota
  - Það þýðir að við þurfum að halda vel utan um bendana okkar og passa að týna þeim ekki

# Minnisleki

```
void fun() {  
    int *x = new int[10];  
    for(int i = 0; i < 10; i++) {  
        x[i] = i;  
    }  
}  
  
int main() {  
    fun();  
    return 0;  
}
```

memoryleak.cpp

# Ábyrgð

- Við hönnun forrita þurfum við að íhuga vel hver ber ábyrgð á því að skila því minni sem við tökum frá
- Það þarf alls ekki að vera að minni þurfi að skila á sama stað (sama falli) og því var úthlutað

# Hvar á delete að vera?

```
int special(int n) {  
    int *arr = new int[n];  
    int sum = 0;  
    arr[0] = 1;  
    for(int i = 1; i < n; i++) {  
        arr[i] = 0;  
        for(int j = 0; j < i; j++)  
            arr[i] += arr[j];  
        sum += arr[i];  
    }  
    return sum;  
}  
  
int main() {  
    cout << special(11) << endl;  
    return 0;  
}
```

delete1.cpp

# Hvar á delete að vera?

```
int sum(int *arr, int n) {  
    int s = 0;  
    for(int i = 0; i < n; i++)  
        s += arr[i];  
    return s;  
}  
  
int main() {  
    int *x = new int[10];  
    for(int i = 0; i < 10; i++)  
        x[i] = i * i;  
    cout << sum(x, 10) << endl;  
    return 0;  
}
```

delete2.cpp

# Hvar á delete að vera?

```
int* init_arr(int n) {  
    int *arr = new int[n];  
    for(int i = 0; i < n; i++) {  
        arr[i] = 0;  
    }  
    return arr;  
}  
  
int main() {  
    int *x = init_arr(10);  
    for(int i = 0; i < 10; i++) {  
        cout << x[i] << " ";  
    }  
    return 0;  
}
```

delete3.cpp