Grunnatriði og Ad Hoc

Bergur Snorrason

14. janúar 2021

Grunntög og takmarkanir þeirra

- Í grunninn snýst forritun um gögn.

Grunntög og takmarkanir þeirra

- Í grunninn snýst forritun um gögn.
- Þegar við forritum flokkum við gögnin okkar með *tögum*.

Grunntög og takmarkanir þeirra

- Í grunninn snýst forritun um gögn.
- Þegar við forritum flokkum við gögnin okkar með *tögum*.
- Dæmi um tög í C/C++ eru `int` og `double`.

Grunntög og takmarkanir þeirra

- Í grunninn snýst forritun um gögn.
- Þegar við forritum flokkum við gögnin okkar með *tögum*.
- Dæmi um tög í `C/C++` eru `int` og `double`.
- Helstu tögin í `C/C++` eru (yfirleitt):

Heiti	Lýsing	Skorður
`int`	Heiltala	Á bilinu $[-2^{31}, 2^{31}-1]$
`unsigned int`	Heiltala	Á bilinu $[0, 2^{32}-1]$
`long long`	Heiltala	Á bilinu $[-2^{63}, 2^{63}-1]$
`unsigned long long`	Heiltala	Á bilinu $[0, 2^{64}-1]$
`double`	Fleytitala	Takmörkuð nákvæmni
`char`	Heiltala	Á bilinu $[-128, 127]$

Hvað með tölur utan þessa bila?

- Einn helsti kostur `Python` í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

Hvað með tölur utan þessa bila?

- Einn helsti kostur `Python` í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial
print(factorial(100))
```

Hvað með tölur utan þessa bila?

- Einn helsti kostur `Python` í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial
print(factorial(100))
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

Hvað með tölur utan þessa bila?

- Einn helsti kostur `Python` í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial
print(factorial(100))
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

- Það er einnig hægt að nota `fractions` pakkann í `Python` til að vinna með fleytitölur án þess að tapa nákvæmni.

Hvað með tölur utan þessa bila?

- Sumir C/C++ þýðendur bjóða upp á gagnatagið `__int128` (til dæmis gcc).

Hvað með tölur utan þessa bila?

- Sumir `C/C++` þýðendur bjóða upp á gagnatagið `__int128` (til dæmis `gcc`).
- Þetta tag býður upp á að nota tölur á bilinu $[-2^{127}, 2^{127} - 1]$.

Hvað með tölur utan þessa bila?

- Sumir C/C++ þýðendur bjóða upp á gagnatagið `__int128` (til dæmis gcc).
- Þetta tag býður upp á að nota tölur á bilinu $[-2^{127}, 2^{127} - 1]$.
- Þetta þarf ekki að nota oft.

Röðun

- Við munum reglulega þurfa að raða gögnum í einhverja röð.

Röðun

- Við munum reglulega þurfa að raða gögnum í einhverja röð.

Forritunarmál	Röðun
`C`	`qsort(...)`
`C++`	`sort()`
`Python`	`this.sort()` eða `sorted(...)`

Röðun

- Við munum reglulega þurfa að raða gögnum í einhverja röð.
- Skoðum nú hvert forritunarmál til að sjá nánar hvernig föllin eru notuð.

Forritunarmál	Röðun
`C`	`qsort(...)`
`C++`	`sort()`
`Python`	`this.sort()` eða `sorted(...)`

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.
- Við getum raða nær öllum ílátum með `sort`.

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.
- Við getum raða nær öllum ílátum með `sort`.
- Ef við erum með eitthva ílát (til dæmis `vector`) `a` má raða með `sort(a.begin(), a.end())`.

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.
- Við getum raða nær öllum ílátum með `sort`.
- Ef við erum með eitthva ílát (til dæmis `vector`) `a` má raða með `sort(a.begin(), a.end())`.
- Við getum líka bætt við okkar eigin samanburðarfalli sem þriðja inntak.

Röðun í C++

- Í grunninn tekur `sort(...)` við tveimur gildum.
- Fyrra gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.
- Við getum raða nær öllum ílátum með `sort`.
- Ef við erum með eitthva ílát (til dæmis `vector`) `a` má raða með `sort(a.begin(), a.end())`.
- Við getum líka bætt við okkar eigin samanburðarfalli sem þriðja inntak.
- Það kemur þá í stað "minna eða samasem" samanburðarins sem er sjálfgefinn.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.

Röðun í Python

- Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.
- Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.
- Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.
- Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- Til að raða `a` á þennan hátt þarf `a = sorted(a)`.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.
- Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- Til að raða `a` á þennan hátt þarf `a = sorted(a)`.
- Nota má inntakið `key` til að raða eftir öðrum samanburðum.

Röðun í Python

- Til að raða lista í `Python` þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- Gerum ráð fyrir að listinn okkar heiti `a`.
- Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- Til að raða `a` á þennan hátt þarf `a = sorted(a)`.
- Nota má inntakið `key` til að raða eftir öðrum samanburðum.
- Það er einnig inntak sem heitir `reverse` sem er Boole gildi sem leyfir auðveldlega að raða öfugt.

Röðun í C

- Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.
  - `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.
  - `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
  - `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.
  - `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
  - `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
  - `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.
  - `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
  - `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
  - `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.
- Síðasta inntakið er kannski flókið við fyrstu sýn en er einfalt fyrir okkur að nota.

Röðun í C

- Í `C` er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- Til röðunar notum við fallið `qsort(...)`.
- Fallið tekur fjögur viðföng:
  - `void* a`. Þetta er fylkið sem við viljum raða.
  - `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
  - `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
  - `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.
- Síðasta inntakið er kannski flókið við fyrstu sýn en er einfalt fyrir okkur að nota.
- Þetta er *fallabendir* (e. *function pointer*) ef þið viljið kynna ykkur það frekar.

Röðun í C

```c
#include <stdio.h>
#include <stdlib.h>

int cmp(const void* p1, const void* p2)
{
    return *(int*)p1 - *(int*)p2;
}

int rcmp(const void* p1, const void* p2)
{
    return *(int*)p2 - *(int*)p1;
}

int main()
{
    int n, i;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &a[i]);

    qsort(a, n, sizeof(a[0]), cmp);

    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");

    qsort(a, n, sizeof(a[0]), rcmp);

    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```

Test