Funktsioonid

Viit funktsioonile

Kood laetakse arvuti mällu – järelikult igal käsul on oma mäluaadress

Kuna koodil, sh funktsioonil on aadress, saame

- Koodi käivitada teades aadressi
- Deklareerida muutuja, mis sisaldab funktsiooni aadressi
- Funktsiooni parameetriks määrata funktsiooniviida
- Funktsioonist tagastada funktsiooni aadressi

Kasutusvaldkonnad

- Programmi töö ajal dünaamiliselt sobiliku funktsiooni välja kutsumine
- Abstraktsioon - nt andmetüübist sõltumatud funktsioonid (täna Quicksort)
- Tagasihelistamise funktsioonid (*callback*), nt GUI sündmuspõhises programmeerimises nupulevajutus
- „Võlts klasside loomine C-s“

Muutuja deklaratsioon

Järgnevad on **muutujate deklaratsioonid**, mis sisaldavad viita funktsioonile

```
void (*printer)(int);
```

Parameetriks täisarv

Muutuja nimi

Tagastus puudub

```
double (*compute)(float, float);
```

Parameetriteks kaks reaalarvu

Muutuja nimi

Tagastuse tüüp double

Näide 1: Funktsioonikutse

```c
#include <stdio.h>

void PrintNum(int num);

int main(void)
{
    // Declare function pointer, assign address of PrintNum
    void (*printer)(int) = PrintNum;

    // Call a function using function pointer
    printer(-5);
    printer(15);
    return 0;
}

void PrintNum(int num)
{
    printf("Num: %d\n", num);
}
```

Näide 2: Funktsiooniviit parameetrina

```c
#include <stdio.h>

void Foo(void);
void Bar(void);
void FunctionCaller(void (*func)(void));

int main(void)
{
    FunctionCaller(Foo);
    FunctionCaller(Bar);
    return 0;
}
```

```c
void Foo(void)
{
    printf("Called function foo!\n");
}

void Bar(void)
{
    printf("Called function bar!\n");
}

void FunctionCaller(void (*func)(void))
{
    func();
}
```

Näide 3: Arvutused

```c
#include <stdio.h>

int Add(int a, int b);
int Sub(int a, int b);
int Calc(int valA, int valB,
         int (*op)(int, int));

int main(void)
{
    printf("5 + 2 = %d\n", Calc(5, 2, Add));
    printf("5 - 2 = %d\n", Calc(5, 2, Sub));
    return 0;
}
```

```c
int Add(int a, int b)
{
    return a + b;
}

int Sub(int a, int b)
{
    return a - b;
}

int Calc(int valA, int valB,
         int (*op)(int, int))
{
    return op(valA, valB);
}
```

Näide 4: tagasikutse funktsioon

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

void ExitHandler(void)
{
    printf("Bye-bye!\n");
}

int main(void)
{
    if (atexit(ExitHandler))
    {
        fprintf(stderr, "Registering exit handler failed!\n");
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Parameetri andmetüüp
`void func(void)`

`atexit()` lubab registreerida funktsioone, mis kutsutakse välja programmi sulgemise eel

Tagastus `main()` funktsioonist käivitab `ExitHandler()` funktsiooni

Qsort

Meeldetuletuseks (PR1)

	i7-7600u	i5-8600k	i5-8600k gcc -O3
Piiramata tsüklitega mullsort	3.27 s	2.96 s	1.38 s
Elementaarse optimeeringuga mullsort	2.52 s	2.28 s	0.97 s
Mullsort, mis veendub, et ega massiiv juba sorteeritud pole	2.45 s	2.22 s	1.10 s
Qsort	0.00313 s	0.00284 s	0.00213 s

Testitud probleemi suurus: 30 000 juhuslikku positiivset täisarvu
Keskmistatud üle 5 iteratsiooni

Me võime optimeerida
Me võime lisada arvutusvõimsust
Kuid kiirema algoritmi vastu ei saa

qsort

Lihtne ja (enamasti) kiire meetod sorteerimiseks

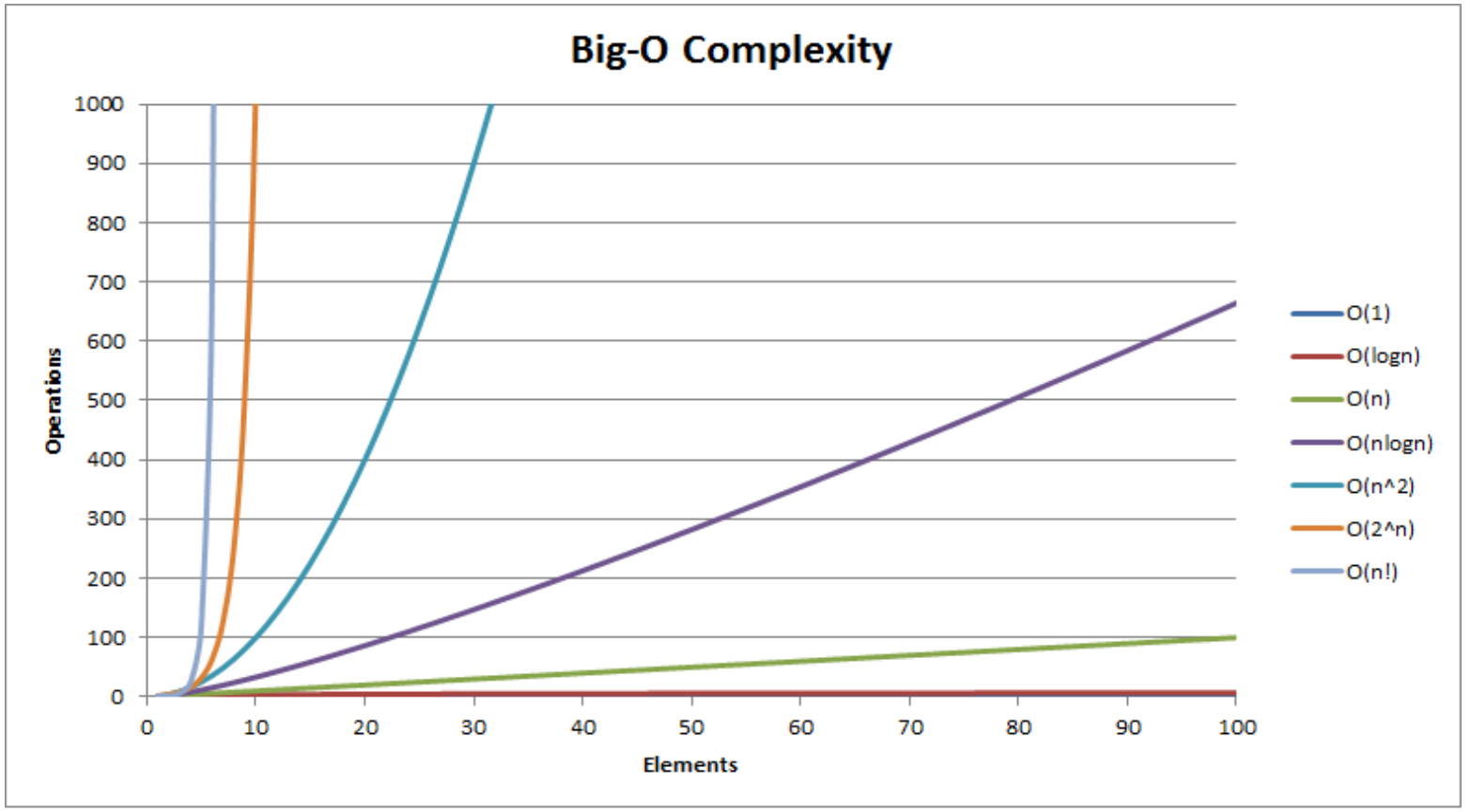
Qsort funktsioon on kirjutatud kõrge abstraktsioonitasemega

- Sisaldab vaid algoritmi
- Töötab ükskõik milliste andmetega, sh enda loodud struktuuridega
- Andmed peavad paiknema **massiivis**
- Väärtuste võrdlemise funktsiooni peame looma ise

Kiirus

- Halvimal juhul sama-aeglane kui mullsorteerimine (n^2)
- Keskmiselt oluliselt kiirem ($n*\log_2(n)$)
- Näiteks 1000 arvu sorteerimine:
  1000^2 = **1 000 000 operatsiooni**
  $1000 * \log_2(1000) = 1000 * 9.97$ = **9 970 operatsiooni**

Keerukus



Graafik: http://bigocheatsheet.com/

qsort funktsioon

Funktsiooni prototüüp:

```
void qsort (void *base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

Parameetrid:

- base – Viit massiivi esimesele liikmele
- num – massiivi liikmete arv
- size – ühe liikme suurus baitides
- compar – funktsiooniviit võrdlusfunktsioonile

Võrdlusfunktsiooni kirjeldus

Qsort kasutamiseks peame looma nõuetele vastava võrdlusfunktsiooni.

```
int compar(const void *var1, const void *var2);
```

Viit ühele massiivi elemendile
nt int, float, struct, …

Parameeter: `const void *var`

- `const` – võrdlusfunktsioon ei tohi andmeid muuta
- `void` – abstraktsioon, töötab ükskõik mis andmetüübiga, **vaja teha tüübiteisendus**!
- `*var` – viit võrreldavale massiivi liikmele

Tagastatav väärtus: täisarv

- < 0 `var1` peab paiknema enne `var2`
- == 0 `var1` ja `var2` on samad (järjestus ei ole oluline)
- > 0 `var1` peab paiknema `var2` järel

QSort pakub vaid algoritmi sorteerimiseks. Võrdlusfunktsiooni eesmärk on qsort funktsioonile öelda kumb kahest võrreldavast elemendist peab olema eespool.

Näide 1: Üldine

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

int CompFunc(const void *pA, const void *pB);

int main(void)
{
    int nums[] = {40, 10, 100, 90, 20, 25};
    int numCnt = sizeof(nums) / sizeof(int);

    qsort(nums, (size_t)numCnt, sizeof(int), CompFunc);

    for (int i = 0; i < numCount; i++)
    {
        printf ("%3d ", nums[i])
    }
    return 0;
}
```

```c
int CompFunc(const void *pA, const void *pB)
{
    if (*(int *)pA > *(int *)pB)
        return 1;
    else if (*(int *)pA < *(int *)pB)
        return -1;
    else
        return 0;
}
```

Parameetriks funktsiooniviit meie võrdlusfunktsioonile. Qsort kutsub meie võrdlusfunktsiooni välja kasutades seda viita.

Näide 2: Üldine, vahemuutujatega

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

int CompFunc(const void *pA, const void *pB);

int main(void)
{
    int nums[] = {40, 10, 100, 90, 20, 25};
    int numCnt = sizeof(nums) / sizeof(int);

    qsort(nums, (size_t)numCnt, sizeof(int), CompFunc);

    for (int i = 0; i < numCount; i++)
    {
        printf ("%3d ", nums[i])
    }
    return 0;
}
```

```c
int CompFunc(const void *pA, const void *pB)
{
    int valA = *(int *)pA;
    int valB = *(int *)pB;

    if (valA > valB)
        return 1;
    else if (valA < valB)
        return -1;
    else
        return 0;
}
```

Mõtle! Miks see versioon ei tööta float ega double andmetüüpidega?

Näide 3: Erijuht täisarvudele

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

int CompFunc(const void *pA, const void *pB);

int main(void)
{
    int nums[] = {40, 10, 100, 90, 20, 25};
    int numCnt = sizeof(nums) / sizeof(int);

    qsort(nums, (size_t)numCnt, sizeof(int), CompFunc);

    for (int i = 0; i < numCount; i++)
    {
        printf ("%3d ", nums[i])
    }
    return 0;
}
```

```c
int CompFunc(const void *pA, const void *pB)
{
    return  *(int *)pA - *(int *)pB;
}
```

Täisarvu ületäitumise oht!