STU FEI

Programovacie techniky

1. Úvod, Bjarne, lokálne premenné, preťažovanie funkcií, referencia &, univerzálna referencia &&, const * const, kompilácia C++ kódu.

Martin Drozda
martin.drozda@stuba.sk
C601

Prednáška: pondelok 15:00-17:00 BC300
Tutoriál: utorok 12:00-13:00

Konzultácie: podľa skupín

Podmienky absolvovania

Test 1: 5.-6. týždeň (10 bodov)

Test 2: 11.-12. týždeň (10 bodov)

Úlohy: počas semestra (10 bodov)

Test 1 + Test 2 + Úlohy: min. 15 bodov

Opravný test (ospravedlnená neúčasť): december

Riadny termín / Opravný termín (70 bodov)

Bonusové body na cvičeniach a na prednáške

Podmienky absolvovania

Predmet je prednáško-centrický

Cvičenie ≠ prednáška

Učebný text

Zbierka príkladov v1.03

Na web stránke predmetu.

Obsah predmetu

Programovacie techniky: dátové štruktúry údajov a techniky práce s nimi.

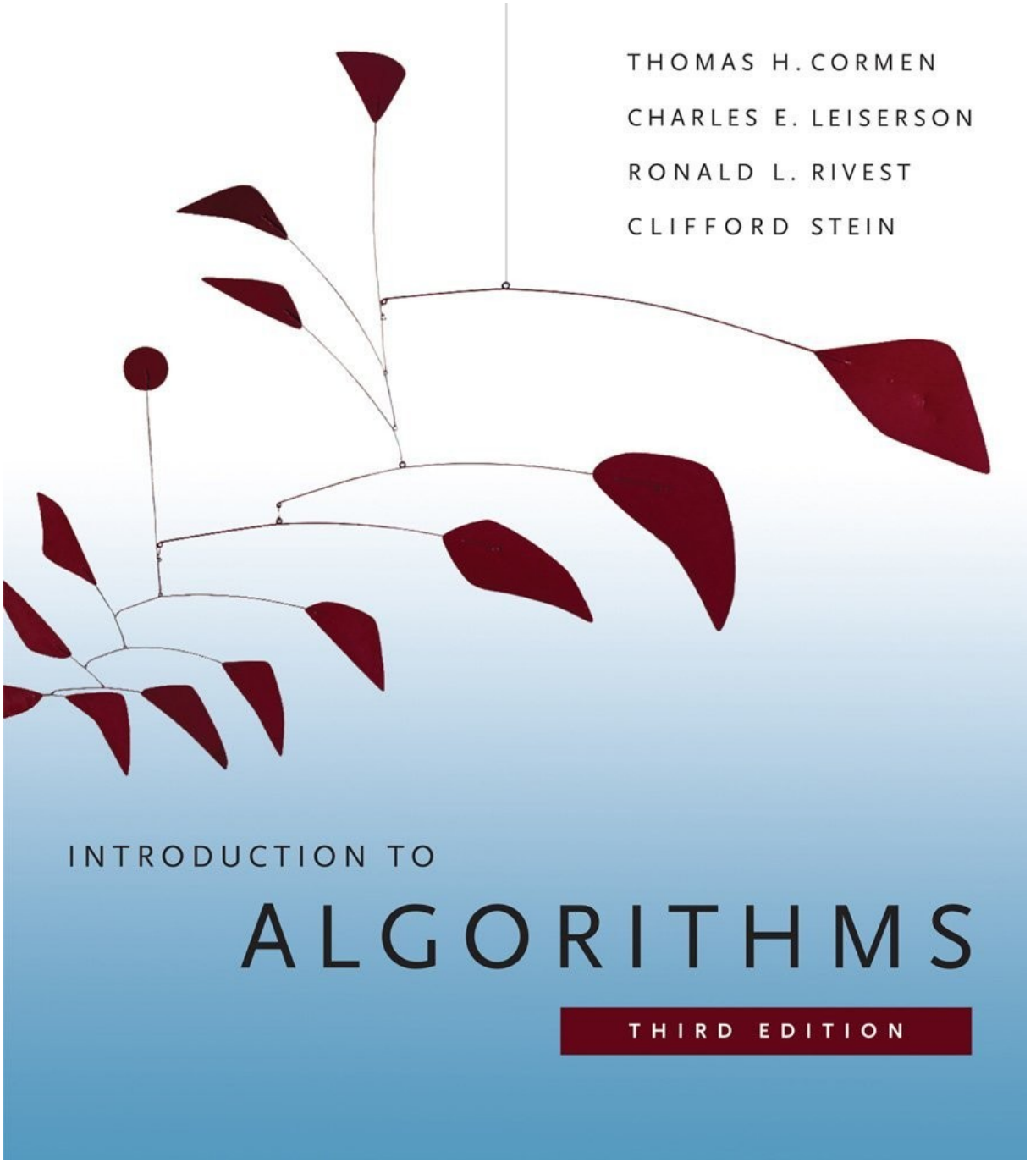
- Hľadanie a triedenie
- Grafové algoritmy

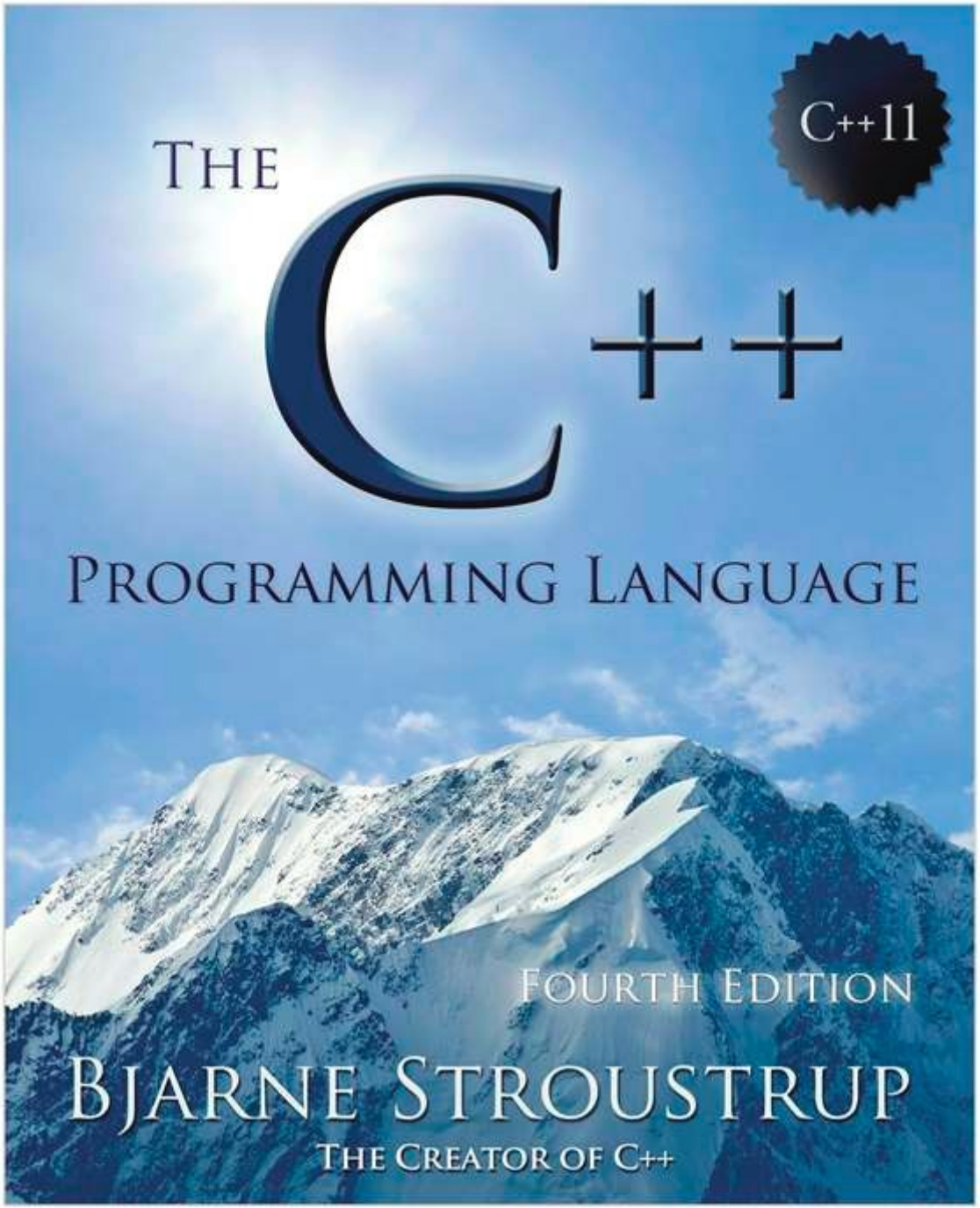
Programovací jazyk C++ (základy)

Evaluácia 2017

"Zaujímavé teoretické poznatky z programovania a osvojovanie metód, techník, nie len bezhlavé kódovanie."

Tento predmet nie je základný kurz C++. Cieľ je efektívne programovanie.


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION


C++11
THE
C++
PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++

B. Stroustrup

Smerníky

Ak ešte stále nerozumiete smerníky, ste v nesprávnom čase na nesprávnom mieste. :(

C++ = veľa smerníkov, referencií a podobných zaujímavostí.

C++

C++98, C++03, C++11, C++14, C++17, C++20

Najnovší štandard je C++17

Kompilátor nemusí implementovať všetky časti štandardu

Premenná

```
int a = 1;
```

a je l-hodnota

1 je r-hodnota

1 je konštanta, literál

Adresa l-hodnoty je známa.

r-hodnota je všetko, čo nie je l-hodnota (do C++11).

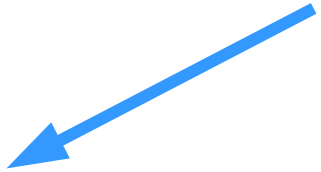
Pretečenie premennej

```
int main() {
  int x = 1000000;
  int y = 1000000;

  long long z = x * y;
  //long long z = 1LL * x * y;
  return 0; }
```

Môže nastať pretečenie premennej.

sizeof(int), sizeof(long long) ??

Poradie evaluácie



```
double x = (3,14);
printf("%lf", x); //14
```

```
warning: left-hand operand of comma expression
   has no effect [-Wunused-value]
   double x = (3,14);
```

Lokálna premenná

```
int main() {
  int x;


  return 0;
}
```

Lokálna premenná zanikne

Lokálna premenná

```
void funkcia(int x) { //int x = $1

  ...

}
```

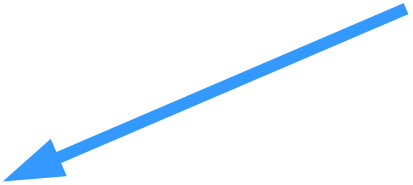
Lokálna premenná x zanikne

Zmena hodnoty

```
int main(){
  int p[10];
  int x = 10;
  p[0] = x;
  x = 20;


  return 0;
}
```

Do p[0] sa vloží hodnota x.
Neskoršia zmena hodnoty x na 20 nemá vplyv na p[0].

Kopírovanie



Do x sa prekopíruje hodnota x z main()
x vo foo() je iná premenná ako x v main(); majú rozdielne adresy

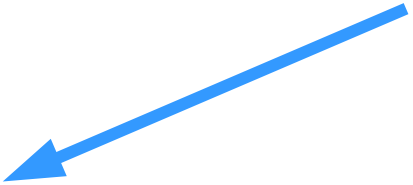
```
int foo(int x) {
  return x+1;
}
```

Lokálna premenná x zanikne

```
int main() {
  int x = 10;
  printf("%d", foo(x));

  return 0;
}
```

Lokálna premenná x zanikne

Kopírovanie



Do x sa prekopíruje adresa y z main()

```
int foo(int* x) {
  return *x;
}
```

Lokálna premenná x zanikne

```
int main() {
  int x = 10;
  int* y = &x;
  printf("%d", foo(y));

  return 0;
}
```

Lokálna premenná x zanikne

V C je všetko “by copy”

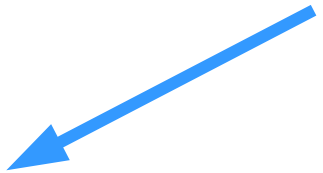
Kopírovanie

```
struct s {
    int a;
    int b;
    int c;
};
int foo(struct s s0) {
  return s0.a;
}
int main() {
  struct s s;
  s.a = 10;
  printf("%d", foo(s));
  return 0;
}
```

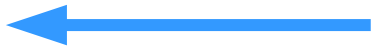
Čo ak by mala struct s 1000 položiek? Položky a0...a999.

Do s0 sa prekopíruje hodnota struct s z main(), a to položka po položke. T.j. je potrebné prekopírovať hodnoty a, b, c, čo je náročnejšie ako prekopírovať len adresu.

Lokálna premenná



```
for(int x=0; x<10; ++x) {
  ...

}
```



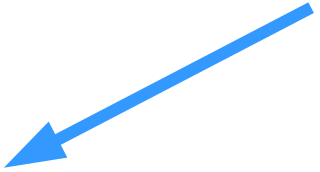
Lokálna premenná zanikne

```
printf("%d", x); //x v tomto scope neexistuje
                 //neskompiluje
```

for cyklus

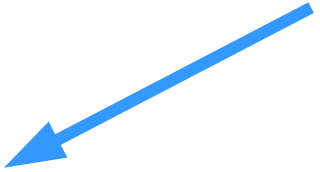
```
int x = 0;
for(; x<10; ++x) {
 ...
 }

printf("%d", x); //10
```

for cyklus



```
int x = 0;
for(; x<10; ++x, printf("%d", x)) {
 ...
 }
```

for cyklus



```
int x = 0;
for(; x<10; printf("%d", x), ++x) {
 ...
 }
```

for cyklus

```
int x = 0;
for(int x = 0; x<10; ++x) {
 ...
 }
printf("%d", x); //0
```

Lokálna premenná x cyklu for zanikne

Lokálna premenná

```
int* funkcia() {
 int x = 10;
 return &x;
}
```

Lokálna premenná zanikne

warning: function returns address of local variable [-Wreturn-local-addr]

return &x;

malloc

```
int* f() {
  int* x = (int*) malloc(sizeof(int));
  *x = 10;
  return x;
}
int main() {
  int* x = f();
  printf("%d", *x); //10
  return 0;
}
```

Lokálna premenná x zanikne

V čom je rozdiel? malloc() alokuje na heape, nie na zásobníku (stacku).

Chyby

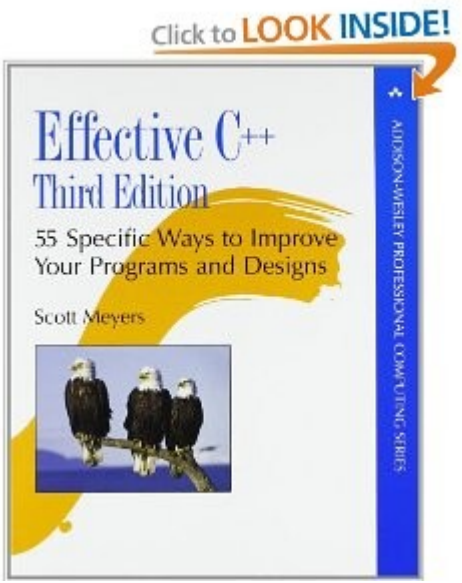
50% neviem, čo je to lokálna premenná

30% neviem ako vzniká objekt (C++)

20% logické chyby (neskúsenosť)

Meyers: Effective C++

Odporúčaná ďalšia literatúra:



Effective C++: item 21

Non-const pointer, non-const data:

```
char *p = "Hello";
```

Non-const pointer, const data:

```
const char *p = "Hello";
char const *p = "Hello";
```

Const pointer, non-const data:

```
char * const p = "Hello";
```

Const pointer, const data:

```
const char * const p = "Hello";
char const * const p = "Hello";
```

printf()

```
int printf(const char *format, ...);
printf("%d", d);
```

Prosím nemanipuluj s obsahom premennej format! Preto som ju deklaroval ako const.

Jasné vyjadrenie zmyslu premennej format

Ak kódujú viacerí programátori, tak je upresnenie zámeru dôležité

++i, i++

```
int i = 1;
++i;
i++; //to isté, pretože nevraciam žiadnu hodnotu

int a = ++i + ++i; //prefixová inkrementácia
int a = i++ + i++; //postfixová inkrementácia
```

Pri postfixovej inkrementácii je potrebné uložiť zoznam premenných, ktoré budú neskôr inkrementované.

C++: pass by reference

```
void f() {
        int i = 1;
        int& r = i;
}
```

r je alias premennej i

r je “to isté” ako i

C++: pass by reference

```
void f() {
        int i = 1;
        int& r = i;
        int x = r;  //x = 1
        r = 2;      //aj i = 2, x = 1
}
```

printf() je C funkcia a r je C++ referencia

printf() nevie vytlačiť/spracovať r

C++: r-hodnota

```
void f(int& x) {
   ++x; //???, ako inkrementovať r-hodnotu?
}

int main() {
   f(3);

   return 0;
}
```

C++: r-hodnota

```
void f(const int& x) {

}

int main() {
   f(3);

   return 0;
}
```

Referencia: inicializácia?

```
int main() {
  int& a = 1; //chyba
  const int& b = 1; //OK

  return 0;
}
```

error: cannot bind non-const lvalue reference of type ‘int&’ to an rvalue of type ‘int’

Jazyk C++: pass by reference

```
void f(int& p) { //p nie je kópia, ani smerník
    p = 10;
}

int main() {
      int a = 6;
      f(a);  //a = 10
}
```

Referencie sú implementované ako smerníky...Bjarne spravil prácu za nás ;)

Jazyk C++: pass by reference

a je referencia na existujúcu premennú

```
void f(int& a) {
    a = 10; //zmení sa aj hodnota inej
      premennej
}
```

a obsahuje prekopírovanú hodnotu

```
void f(int a) {
    a = 10; //a je lokálna premenná
}
```

Referencia &&

```
void foo(int&& b) { //referencia na r-hodnotu
  ++b; //OK

}

int main () {
  foo(3);
}
```

Preťažovanie funkcií

```
void foo(int&& b) { //referencia na r-hodnotu
  ++b;
}
void foo(int& b) {
}

int main () {
  foo(3); //zavolá foo(int&&)
  int a = 10; //zavolá foo(int&)
  foo(a);
  return 0; }
```

Preťažená funkcia musí mať rozdielny vstupný parameter

gcc/g++ (Linux)

g++ file.C -o file //skompiluje file.c do file (bez prepínača -o bude skompilované do súboru a.out)

g++ file.C -Wall -Wextra -o file //(skoro) všetky warnings

g++ file.C -pedantic -o file //ešte viacej warnings, zvýšená kompilovateľnosť rôznymi kompilátormi

g++ file.C -Wall -O2 -o file //optimalizácia kódu

g++ file.C -Wall -O3 -o file //viac optimalizácie

g++ file.C -Wall -Os -o file //minimalizácia veľkosti kódu

g++ file.C -ggdb -o file //kód pre debugger gdb

Referencia na lokálnu premennú

```
int& foo(int b) {
  return b;
}

int main () {
  int a = 10;
  foo(a);
  return 0;
}
```

warning: reference to local variable ‘b’ returned [-Wreturn-local-addr]