ch 7, 8, 9

Stroustrup
ch. 17 pp 569-610
1

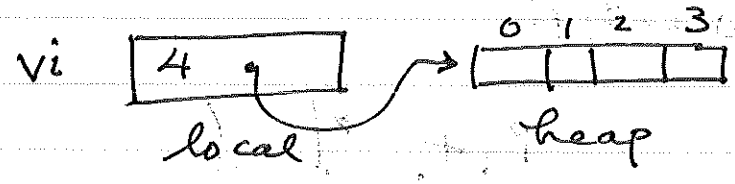
17. Vector & free store

vector: op[] push_back
size() pop_back

string
list
map

17.2 Basics

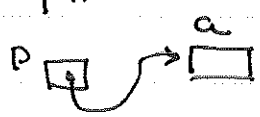
```
vector<int> vi(4)
```

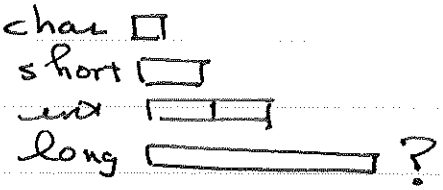



```
class vector {
    int -size
    int *data
}
```

17.3 mem, addr = ptr

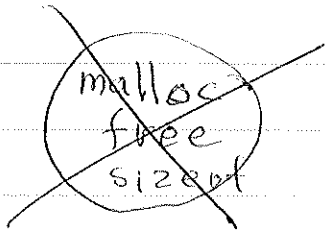
```
int a
int *p = &a
```



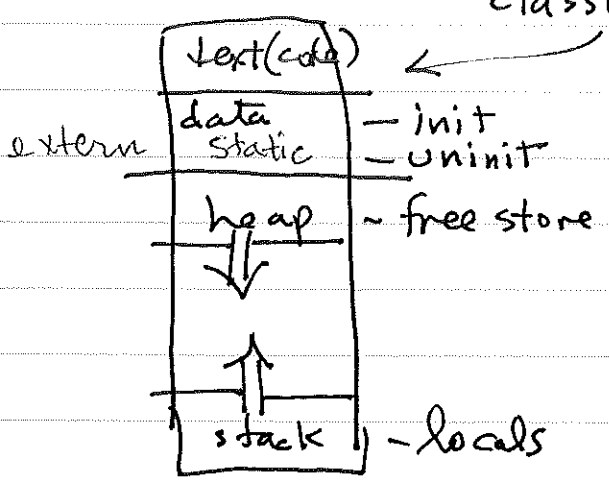


alignment char* ≠ int*

sizeof operator



17.4 memory

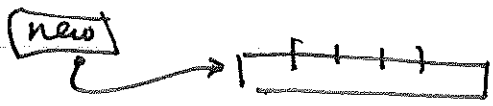


Stroustrup 17

2

new new[]
delete delete[]

~~malloc free~~



does not know #elts.

double *p = new double
p = new double[4]

foo *p = new foo();
p = new foo[4];

p[i] ≡ *(p+i)
≡ *(i+p)
≡ i[p]

use p[i] for array
*p for unit

∴ p[0] ≡ *p

p→f ≡ (*p).f

note ∅ ≡ NULL

p = new T(); ⟶ delete p;
p = new T[n]; ⟶ delete[] p;

Problem : p = new T[4];
p[-1] ?
p[6] ?
> does not know size

17.4.4 Initialization

double *p∅
double *p1 = new double
double *p2 = new double(1.6)
double *p3 = new double[5]

p∅ uninit
p1 init, *p1 uninit
p2 init, *p2 init
p3 init, array uninit

— no init for array of objects
— always default ctor for objs

17.4.5 NULL

∅ (digit) also null ptr

p≠0 ⟶ true
p==0 ⟶ false

if(p) ≡ if(p != ∅)

Stroustrup 17

3

17.5 Destructors

```
class vector {
explicit vector(int s) : size(s), data(new int[s]) { ? }
                                                   ↑ if <0 ???
    ~vector() { delete [] data }

int f() {
    vector v;
}  ← implicit delete [] call.
```

Beware:

~~vector v();~~

17.5.1 Auto Destruct

- destruct obj → ∀ field destruct obj
  - prim → ϕ
  - obj → calldest
  - ptr → ϕ

Life cycle: ctor acquire resource
lifetime acq & rel
end: release res.

```
ex: f() {
        foo *p = new foo
        delete p
    }
```

```
{ foo p
} - implicit delete
```

17.6 access to element

```
int get(int n) const { ret data[n] }
void set(int i, int v) { data[i]=v }
```

a = v[i] a = v.at(i)

v[i] = a v.at(i) = a

17.6 Ptrs to obj

vec *p = new vec
delete p ⟶ call dtor ~vec

new :– alloc mem
– call ctor

delete :– call dtor each ~~elem~~ field
– free mem

general: avoid new except ctor
delete except dtor

~~to & to void~~ ptr to *

17.7 Pointers

vector *p = new vector(20)
delete p;

1. call dtor
2. free obj

1. alloc obj
2. call ctor

real: vector<vector<double>> *p
= new vector<vector<double>>();

space

typedef to avoid stutter

17.8 void* avoid

(cast) = avoid ⟶ (int)x
use conv instead: int(x)

compile → static_cast <T> (v) related ptr types & void*
run ⟶ dynamic_cast

const_cast – assign to const
reinterpret_cast – move bits

ex: int *p = reinterpret_cast<int *>(0xFF);

abs addr

17.9 Ptrs & Refs

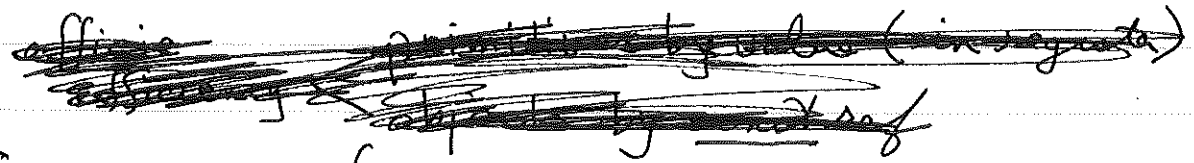
- p= change ptr, not pointee
- get a ptr: new or &x
- access via ptr: *p or p[i]
- ref = changes obj not ref
- — can't change ref.
- ref never null ∴ can't use for ∅
- $r_1 = r_2$ does deep copy (via ctors)
- $p_1 = p_2$ does shallow copy
- Beware of null ptrs

ex: `int *p1 = x;` `int &r1 = x`

`*p1 = 8` `r1 = 8`

17.9.1 Params

```
int incrV(int x);      x = incrV(x)
void incrp(int *p)     incrp(&x)
void incrR(int &r)     incrR(x)
```



Efficiency prefs

primitive register objs → by value

objects which may be null → by pointer

guaranteed non null objects → by const reference

else by reference if need to change