Pointers I

DM2111
C++ Programming

Introduction

Introduction	Break
Problem solving	Array and Strings
Basic elements of C++	Array and Strings
Basic elements of C++	Pointers
Statements	Pointers
Repetition	I/O operations
Functions	Structs
Functions	Others

Agenda

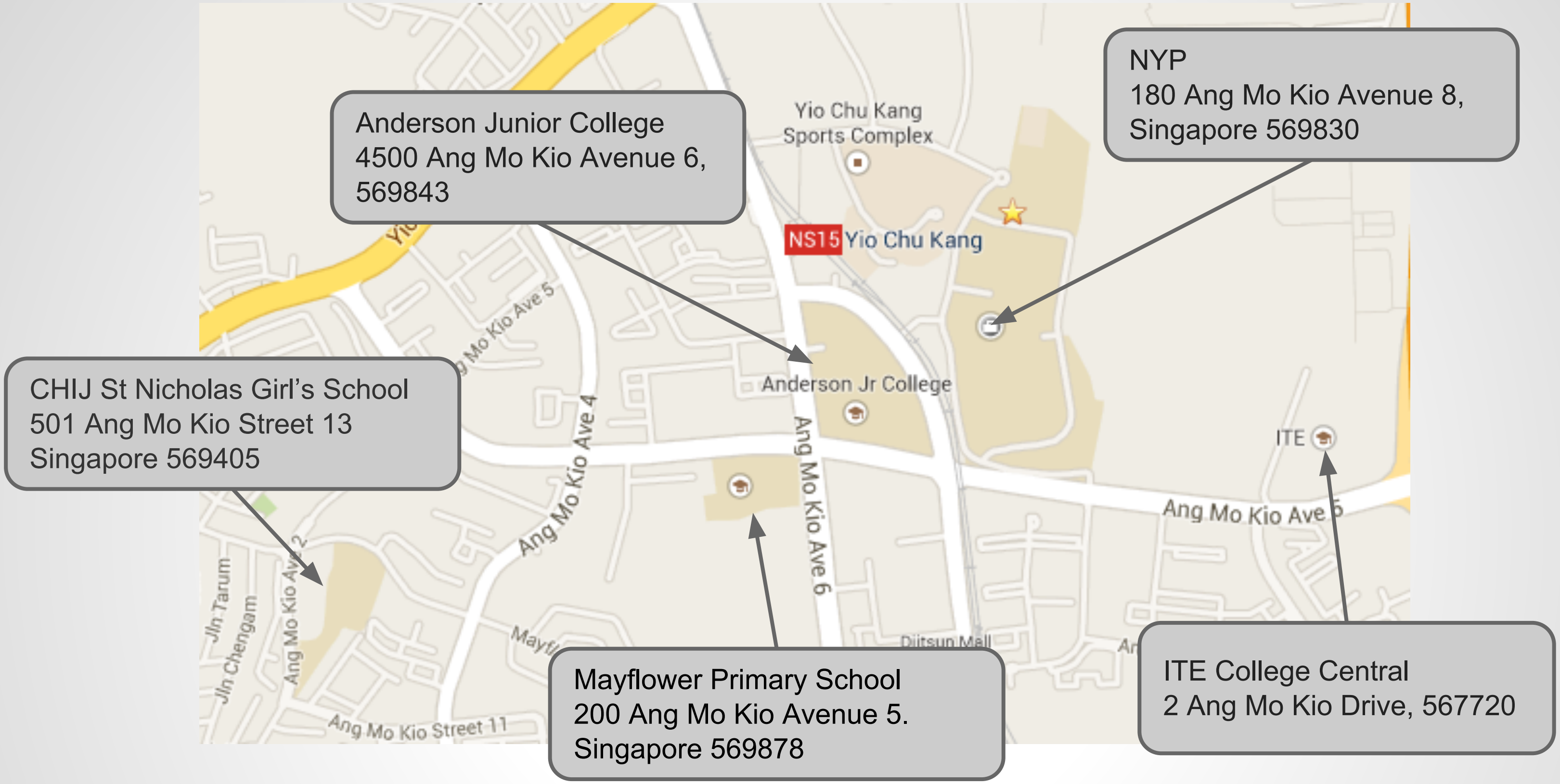
- Pointers
- Dereferencing
- Dynamic Variables
- Dynamic Arrays
- Pointer Parameters

Pointers

A type that “points to” another type

Used to access variables indirectly.

You know address?



Anderson Junior College
4500 Ang Mo Kio Avenue 6,
569843

NYP
180 Ang Mo Kio Avenue 8,
Singapore 569830

CHIJ St Nicholas Girl's School
501 Ang Mo Kio Street 13
Singapore 569405

Mayflower Primary School
200 Ang Mo Kio Avenue 5.
Singapore 569878

ITE College Central
2 Ang Mo Kio Drive, 567720

What is in a name?

How would you prefer to refer to this place?

Nanyang Polytechnic?

NYP?

180 Ang Mo Kio Avenue 8, Singapore 569830?

Poly?

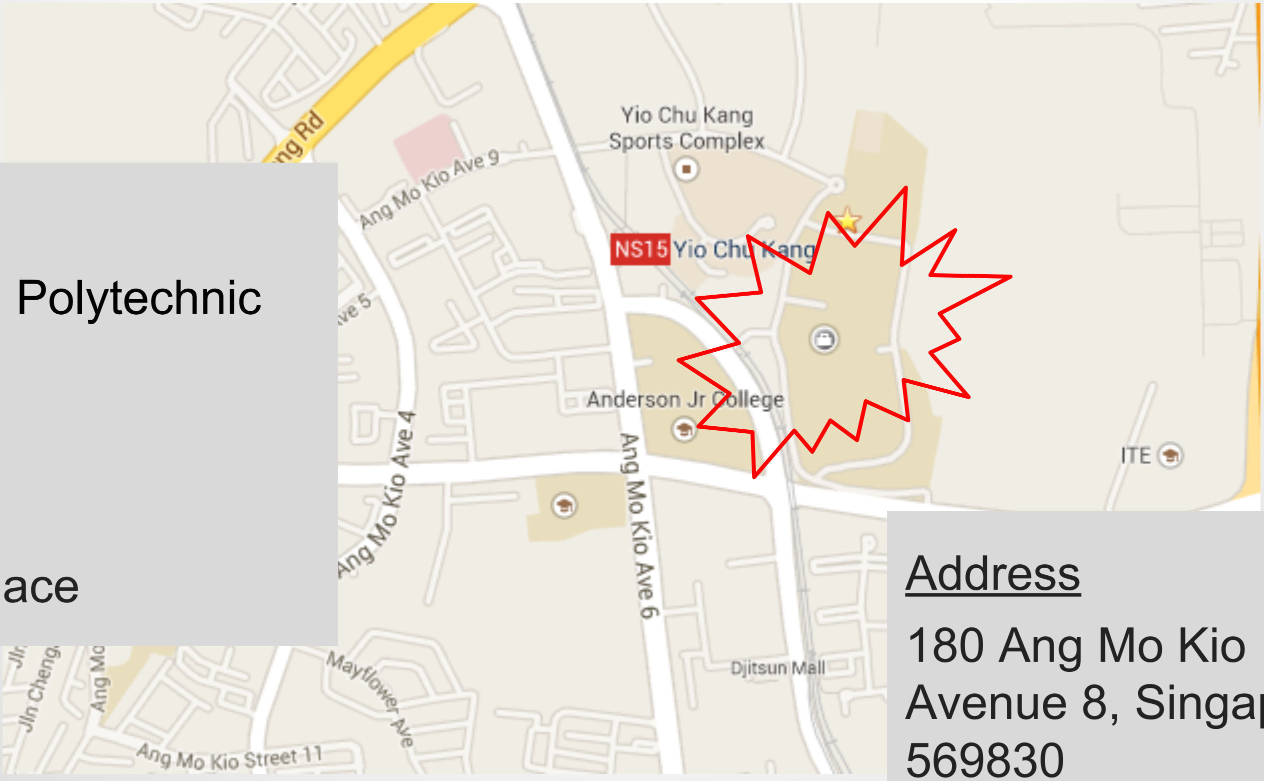
School?

Yio Chu Kang Rd
Ang Mo Kio Ave 9
Yio Chu Kang Sports Complex
NS15 Yio Chu Kang
Ang Mo Kio Ave 5
Anderson Jr College
Thomson Hill
Ang Mo Kio Ave 4
Ang Mo Kio Ave 6
ITE
Ang Mo Kio Ave 5
Jln Tarum
Jln Chengam
Ang Mo Kio Ave 2
Mayflower Ave
Djitsun Mall
Ang Mo Kio Street 52
Ang Mo Kio Street 11

What is in a name?

“A rose by any other name would smell as sweet”
Juliet

spoiler - she died in the end



Names

Nanyang Polytechnic

NYP

Poly

School

My fav place

Address

180 Ang Mo Kio
Avenue 8, Singapore
569830

What is in a name?

Variable Name -> Nanyang Polytechnic, NYP

Address -> 180 Ang Mo Kio Avenue 8, Singapore 569830

```
int ivar;      // integer variable
float fvar;    // float variable

ivar = 35;
fvar = 3.14f;

cout << &ivar << endl;   // 00664AA0
cout << &fvar << endl;   // 00664AA4
```

Name	Addr	Content
ivar	0x00664AA0	35
fvar	0x00664AA4	3.14f
NYP	180 Ang Mo Kio Avenue 8, Singapore 569830	Students, buildings, trees.

Contents of pointers are addresses

`iptr` is a pointer
`fptr` is a pointer

Name	Addr	Content
iptr	0x00AA4FF0	0x00664AA0
fptr	0x00AA4FF4	0x00664AA4
ivar	0x00664AA0	35
fvar	0x00664AA4	3.14f
School	some neurons in your head	180 Ang Mo Kio Avenue 8, Singapore 569830
NYP	180 Ang Mo Kio Avenue 8, Singapore 569830	Students, buildings, trees.

`iptr` contains the address of a variable
`fptr` contains the address of a variable

Pointers contain addresses that match the type

`iptr` is an **integer** pointer
`fptr` is a **float** pointer

Name	Addr	Content
iptr	0x00AA4FF0	0x00664AA0
fptr	0x00AA4FF4	0x00664AA4
ivar	0x00664AA0	35
fvar	0x00664AA4	3.14f
School	some neurons in your head	180 Ang Mo Kio Avenue 8, Singapore 569830
NYP	180 Ang Mo Kio Avenue 8, Singapore 569830	Students, buildings, trees.

`iptr` contains the address of an **integer** variable
`fptr` contains the address of a **float** variable

Value in pointers should be in one of these 4 states

1. Point to an object.
2. Point to a location immediately past the end of an object.
3. Null pointer, indicating that it is not bound to any object.
4. Invalid, not any of the above.

Pointer declaration

Declaration

```
int val;      // integer variable
int *ptr;     // pointer to integer

char ch1;     // character variable
char *ptr2;   // pointer to character

double dval;  // double variable
double *dptr; // double pointer
```

Pointers should be initialized to 0 or a valid location

```
int *a;
a = NULL;
```

↔

```
int *a;
a = 0;
```

Otherwise, you wouldn’t know it if is valid or not.

Pointer declaration

Multiple Declaration

```
int *a, b;
```

⇔

```
int *a;
int b;
```

```
int *a, *b;
int *c;
int *d
```

Don’t be lazy

Pointer assignment

Remember that pointers store addresses. You need to assign addresses to the pointer.

The address-of (&) operator refers to the address of its operand

& = What is your address? Where do you store your value?

```
int *p = 0;
int *p2 = 0;
int ival;
int ival2 = 8;

ival = 0xF2;    // ival holds the value 0xF2
p = &ival;      // p is assigned address of ival
p = &ival2;     // p is assigned address of ival2
p = ival;       // illegal, int to address
p2 = p;         // assign addresses, can do
ival2 = p;      // illegal, address to int
```

Pointer assignment must be of same type

```
int ival;
int *iptr = 0;
char cval;
char *cptr = 0;

iptr = &ival;     ✓// same integer type, compiler (*v*)
cptr = &cval;      // same char type, compiler (*v*)

iptr = &cval;     ✗// not same type, compiler (p_q)
cptr = &ival;      // not same type, compiler (p_q)
```

Dereferencing pointers

The dereferencing (*) operator is used to access that object.
* = Go to the address, tell me what is there

```
int ival;
int *iptr = 0;

ival = 242;     // ival is assigned value of 242
iptr = &ival;   // iptr is assigned to “point at” ival

cout << ival;   // 242
cout << iptr;   // 0x0029F774 i.e. address of ival
cout << *iptr;  // 242
```

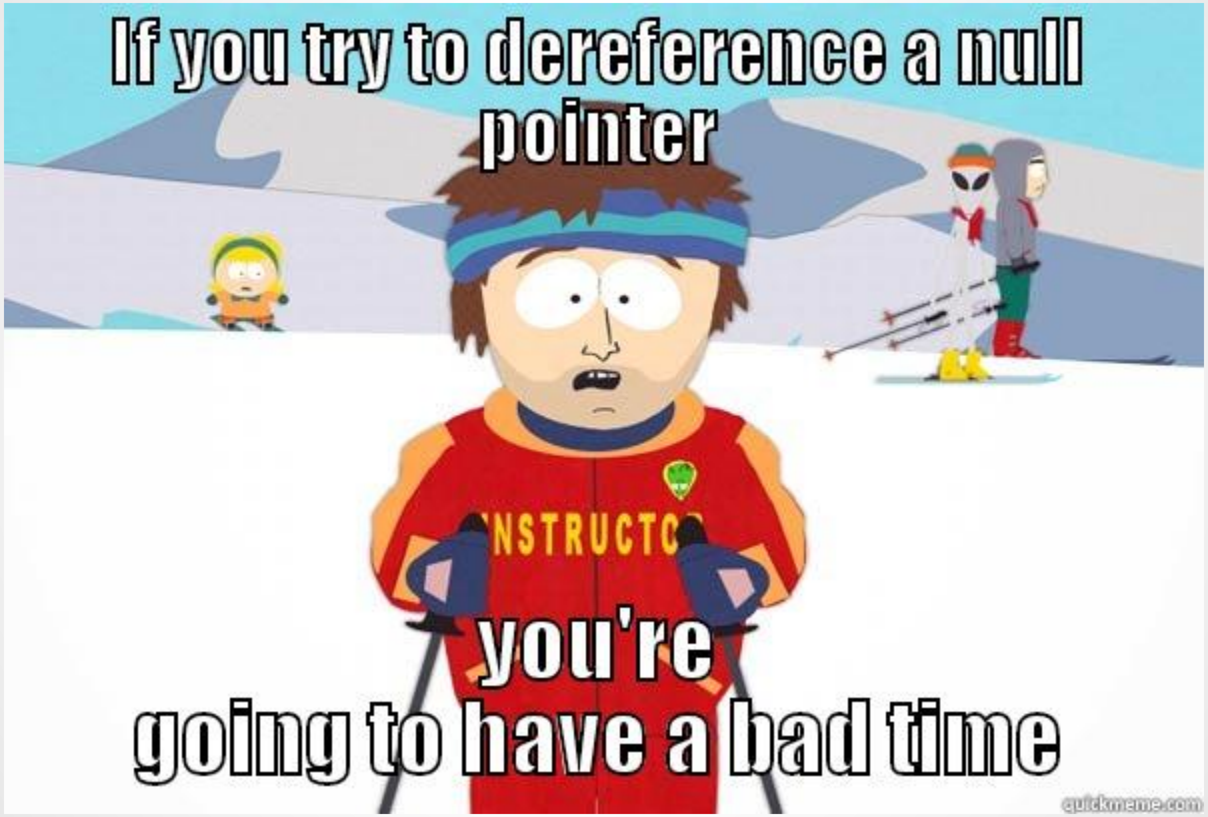
Dereferenced object is the object

```cpp
int ival;
int *iptr = 0;  // integer pointer initialized to 0

ival = 242;     // ival is assigned value of 242
iptr = &ival;   // iptr is assigned to “point at” ival

cout << ival;   // 242
cout << iptr;   // 0x0029F774
cout << *iptr;  // 242

ival = 10;
cout << *iptr;  // 10, iptr points to ival

*iptr = 123;
cout << *iptr;  // 123, dereferenced object was changed
cout << ival;   // 123, iptr points to ival
```

Pointers can point at different variables

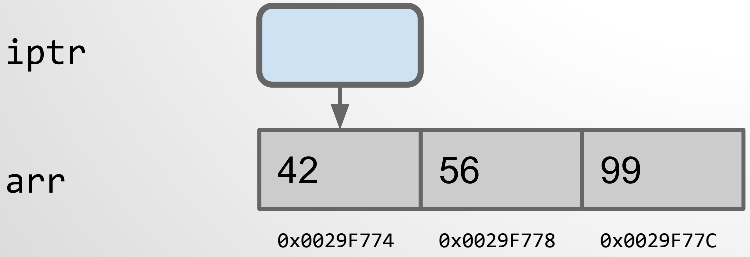
```
int cow;
int duck;
int *aptr = 0;  // animal pointer initialized to 0

cow = 246;      // cow is assigned value of 246
duck = 123;     // duck is assigned value of 123
aptr = &cow;    // aptr is assigned to “point at” cow

cout << *aptr;  // 246

cow = 10;
cout << *aptr;  // 10, aptr points to cow

aptr = &duck;
cout << *aptr;  // 123, dereferenced object is duck
cout << cow;    // 10, cow has 10

*aptr = 50;     // aptr points to duck
cout << duck;   // 50, duck value was changed
```

Dereferencing


If you try to dereference a null pointer
you're going to have a bad time
INSTRUCTO
quickmeme.com

Pointer and arrays, maths

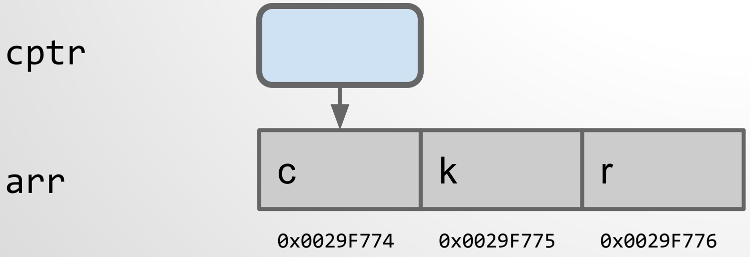
```
int arr[3] = {};   // declare array of 3 int
int *iptr = 0;     // integer pointer

iptr = arr;        // arr holds the address of 1st element
*iptr = 42;        // assign 42 to 1st element
cout << arr[0];    // 42

*(iptr + 1) = 56; // assign 56 to the 2nd element
cout << arr[1];    // 56

arr[2] = 99;       // assign 99 to 3rd element
cout << *(iptr + 2); // 99

*iptr + 1 = 56;    // illegal, pointer is dereferenced then add 1
```



each element is 4 bytes

Pointer and arrays, with array syntax

```
char arr[3] = {}; // declare array of 3 char
char *cptr = 0;   // char pointer

cptr = arr;       // arr holds the address of 1st element
cptr[0] = 'c';    // assign c to 1st element
cout << arr[0];   // c

cptr[1] = 'k';    // assign k to the 2nd element
cout << arr[1];   // k

arr[2] = r;       // assign r to 3rd element
cout << cptr[2];  // r

cptr[0] + 1 = 56; // illegal, hope you know why by now
```



each element is 1 byte

Pointer and maths

address + offset = address (all the time)

offset + offset = offset (what’s the point?)

offset + address = address (no one does this)

address + address = some number (no use)

address - offset = address (yet to see a use)

offset - offset = offset (what’s the point?)

offset - address = negative address? (Stahp!)

address - address = offset (useful)

Dynamic memory allocation during runtime

`new`

- To create memory for variables
- Returns the address of the created variable

`delete`

- To free memory used by the variable
- Takes the variable address and frees it up

Dynamic Variables allows us to use variables as required.

```
int *p = 0;        // 1 declare int pointer

p = new int;       // 2 dynamically allocate memory (heap)
*p = 22;           // 3 assign value
delete p;          // 4 release memory at the address
```

1

Variable	Address	Content
p	0042FC54	0

2

Variable	Address	Content
p	0042FC54	0042FC58
?	0042FC58	??

3

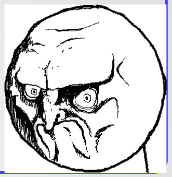
Variable	Address	Content
p	0042FC54	0042FC58
?	0042FC58	22

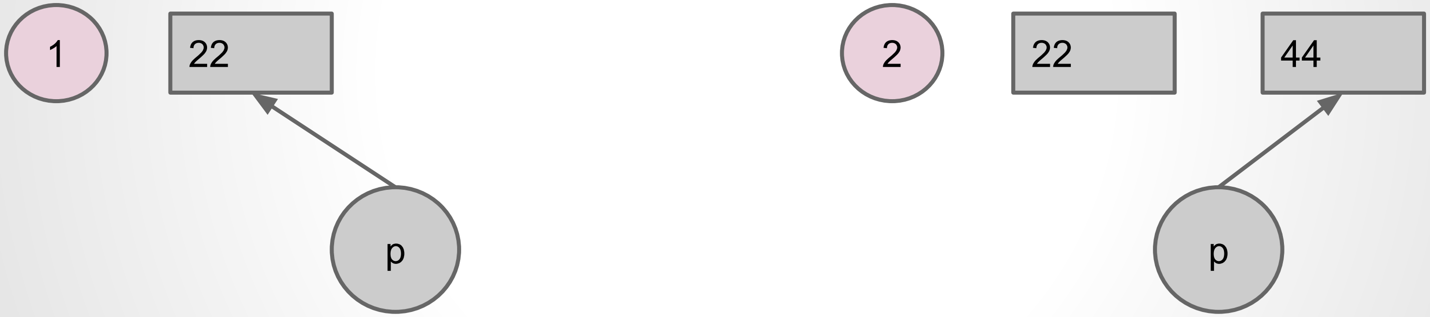
4

Variable	Address	Content
p	0042FC54	0042FC58
?	0042FC58	??

Memory leaks are a real danger

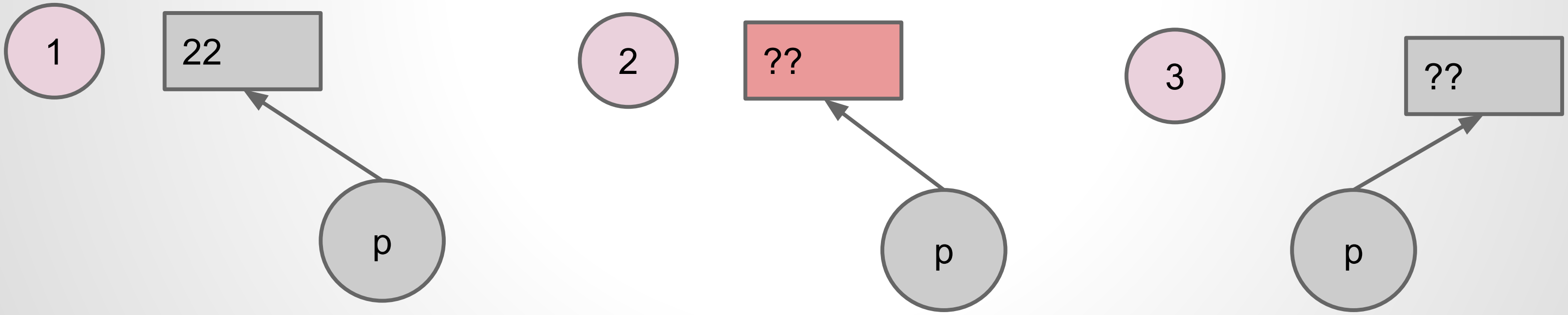
```
int * p;

p = new int;
*p = 22;        // 1 allocates one space

p = new int;    // 2 lost the address of the first memory, leak!
*p = 44;
```





Memory leaks are a real danger

```
int * p;

p = new int;
*p = 22;       // 1 allocates one space
delete p;      // 2 frees up the memory

p = new int;   // 3 gets another memory space, or the same
*p = 44;
delete p;      // remember to clean up
```

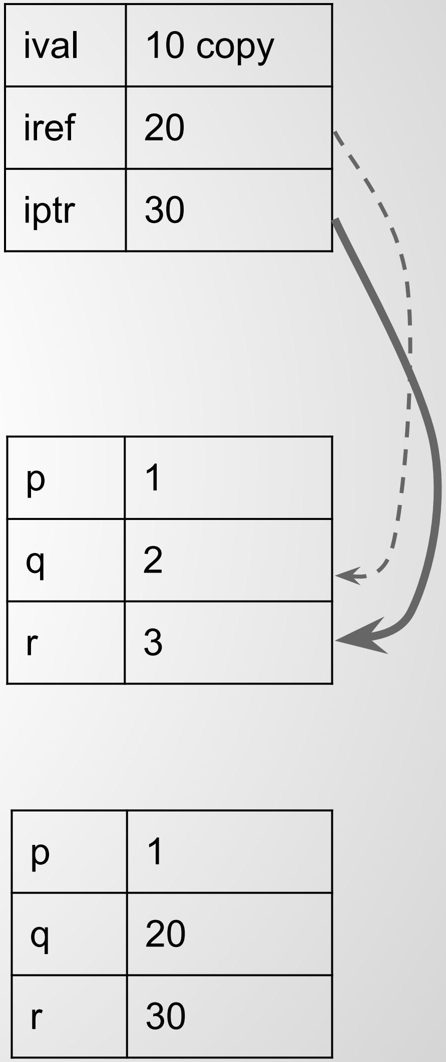


Dynamic Arrays need dynamic cleanup

```cpp
int p[3] = {11, 22, 33}; // array allocated on stack

cout << p;     // address of int array
cout << &p;    // address of 1st element in int array
cout << &p[0]; // address of 1st element in int array

cout << *p;        // 11, 1st element
cout << *(p + 1);  // 22, 2nd element


int *r = new int[3]; // dynamically allocate memory, heap
*r = 11;             // assign 1st element
*(r + 1) = 22;       // assign 2nd element
r[2] = 33;           // assign 3rd element

cout << r;         // address of int array
cout << &r;        // address of int pointer
cout << &r[0];     // address of 1st element in int array
cout << *r;        // 11, 1st element
cout << *(r + 1);  // 22, 2nd element

delete[] r;        // frees up memory from int array
```

Pointers as Parameters

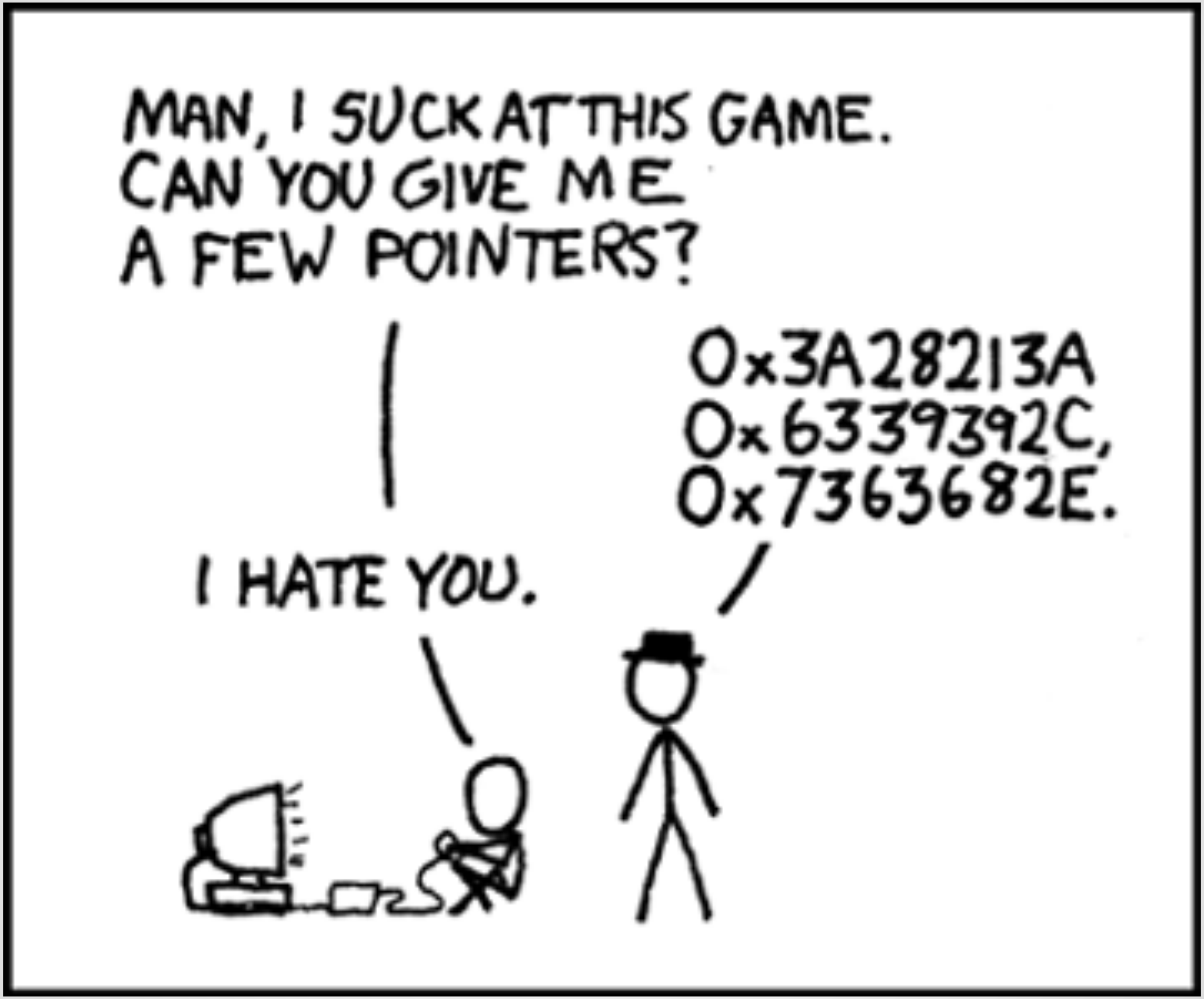
```
void func (int ival, int &iref, int *iptr)
{
    ival = 10;  // passed by value
    iref = 20;  // passed by reference
    *iptr = 30; // passed by pointer value (address)
}

void main (void)
{
    int p = 1, q = 2, r = 3;

    cout << p;    // 1
    cout << q;    // 2
    cout << r;    // 3

    func (p, q, &r);

    cout << p;    // 1
    cout << q;    // 20
    cout << r;    // 30
}
```



ival	10 copy
iref	20
iptr	30

p	1
q	2
r	3

p	1
q	20
r	30


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.