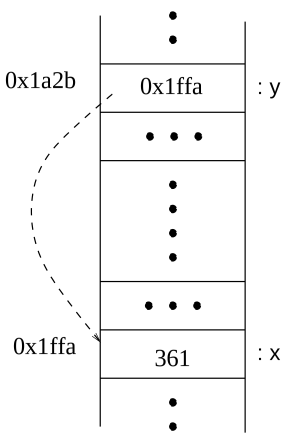
Comp151

Pointers

Pointers: Introduction


```
int x = 361;
int* y = &x;
```

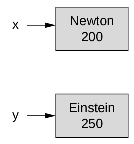


- Related to the concept of "objects" (c.f. objects in OOP and "links" in HTML documents/webpages).
- Two main uses:
  1. indirect addressing (as in assembly language programming)
  2. dynamic memory allocation

Use of Pointers: Dynamic Memory Allocation


```
#ifndef IQ1_H
#define IQ1_H

#include <iostream.h>

class IQ
{
  private:
    char name[20];
    int score;

  public:
    IQ(const char* s, int k)
    {
        strcpy(name, s);
        score = k;
    }

    void smarter(int k) { score += k; }
    void print() const
    {
        cout << "( " << name << " , "
             << score << " )" << endl;
    }
};

#endif // IQ1_H
```

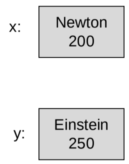
```
#include <iostream.h>
#include "iq1.h"

int main()
{
    IQ* x = new IQ("Newton", 200);
    IQ* y = new IQ("Einstein", 250);

    x->print();
    y->print();

    return 0;
}
```



Use of Pointers: Indirect Addressing


```
#include <iostream.h>
#include "iq1.h"

int main()
{
    int choice;
    IQ x("Newton", 200);
    IQ y("Einstein", 250);

    cin >> choice;
    if (choice == 1)
    {
        x.smarter(50);
        x.print();
    }
    else
    {
        y.smarter(50);
        y.print();
    }

    return 0;
}
```

```
#include <iostream.h>
#include "iq1.h"

int main()
{
    int choice;
    IQ* iq_ptr;
    IQ x("Newton", 200);
    IQ y("Einstein", 250);

    cin >> choice;
    if (choice == 1)
        iq_ptr = &x;
    else
        iq_ptr = &y;

    iq_ptr->smarter(50);
    iq_ptr->print();

    return 0;
}
```



Pointers: Operations

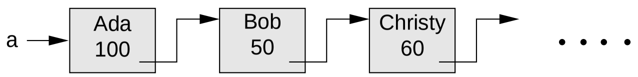
Two basic unary operations:

- *Referencing, &*: $\&x \Rightarrow$ address of variable x.
- *Dereferencing/Indirection*, $*$: $*x \Rightarrow$ value of the memory location stored in variable x.

Other operations: $+, -, ++, --$, **new**, **delete**

```
int x = 1, y = 2, *w;
w = &x;                          // w now points to x
y = *w;                          // c.f. y = x
*w = 3;                          // c.f. x = 3

IQ* x = new IQ("Newton", 200);
x->print();                      // record fields accessed thru pointers
(*x).print();                    // record fields accessed thru non-pointers
```

Pointers and Records: Linked List Example




```
#ifndef IQ2_H
#define IQ2_H

#include <iostream.h>

class IQ
{
  private:
    char name[20];
    int score;
    IQ* next;

  public:
    IQ(const char* s, int k) { strcpy(name, s); score = k; next = NULL; }

    IQ* get_next() { return next; }
    void add(IQ* p) { next = p; }
    void smarter(int k) { score += k; }
    void print() const { cout << "( " << name << " , " << score << " )" << endl; }

};
```

Pointers and Records: Dynamic Data

Dynamic structures (e.g. linked list, binary tree) that grow/shrink during execution can be implemented using records and pointers.

- new : to create a new record
- delete : to destroy a previously allocated record

e.g. To add Angela between Ada and Bob:

```
IQ* p = new IQ(``Angela'', 70); // allocation of *p
p->add(a->get_next());          // p->next = a->next;
a->add(p);                      // a->next = p;
```

e.g. To "fire" Bob:

```
IQ* b = a->get_next();        // b = a->next;
a->add(b->get_next());        // a->next = b->next;
delete b;                     // deallocation of *b
```

Pointers: Common Bugs! (1)

e.g. To add Angela between Ada and Bob:

```
IQ* p = new IQ(``Angela'', 70); // allocation of *p
a->add(p);         // a->next = p; BUG! Bob is lost!
p->add(a->get_next());  // p->next = a->next;
```

Memory Leak: when there is *garbage* — allocated storage/structure which becomes inaccessible.

Pointers: Common Bugs! (2)

e.g. To "fire" Bob:

```
IQ* b = a->get_next();        // b = a->next;
a->add(b->get_next());        // a->next = b->next;
delete b;                     // deallocation of *b
cout << b->name;              // BUG! What is c?
```

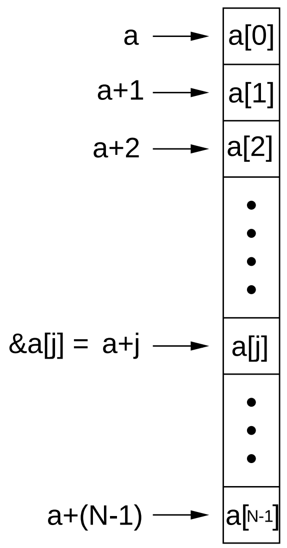
Dangling Pointer: points to storage that is being used for another purpose; typically, the storage has been freed.

Pointers and Arrays

In C++, the name of an array is also treated as a pointer to its first element.

- $a = \&a[0]$.
- $a + j$ is the same as $\&a[j]$.
- $*(a + j)$ is the same as $a[j]$
- Thus, any operation that can be achieved by array subscripting can also be done with pointers.
- The meaning of $(a + j)$ is:

new address
||
base address (a)
+
sizeof(object) $*j$.



Pointers and Arrays: Examples

```
char* strcpy(char* destination, const char* source)
{
    char* d = destination;
    while (*source != '\0')
    {
        *destination = *source;
        ++destination, ++source;
    }

    *destination = '\0';
    return d;
}

char* strcpy2(char* destination, const char* source)
{
    char* d = destination;
    while (*source != '\0')
        *destination++ = *source++;

    *destination = '\0';
    return d;
}
```


Dynamic Allocation of Multi-Dimensional Arrays

To dynamically allocate a 2-D $(N+1) \times (M+1)$ array of IQ during run-time:

```
IQ** a = new (IQ*) [N+1];
for(int i = 0; i < N+1; i++)
    a[i] = new IQ [M+1];
```