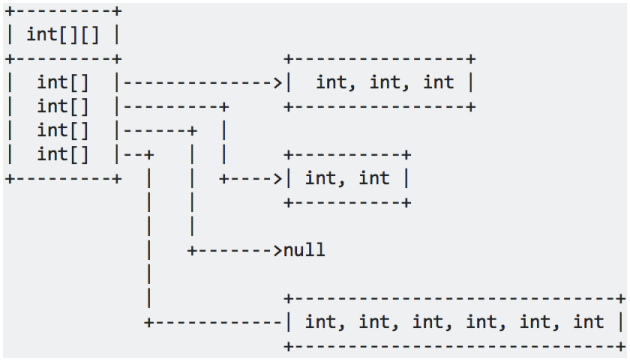
# Types (IV)

## Multidimensional Arrays (cont.)

- Does row-major vs col-major make any difference? If you are searching the array which order should you do it? Why?
  - same as hardware; cache hits increase
- Which way does Java do it?
  - Neither. Advantage of Java way is you can have irregular shaped arrays.
  - Java has arrays of arrays. While a given array's entries are stored in a contiguous block memory, the subordinate arrays those entires point to are object references to complete separate unrelated blocks of memory.




So `a[3][4]` means "Get the array referenced by the entry at index 3 of `a`, then get the entry referenced by index 4 of that subordinate array."

## Type of Data in Arrays

- In statically typed languages, need all entries to be the same type. Why?
  - To avoid having to type cast and so that all slots have same size so can calculate the index.
- Then how do dynamically typed languages handle arrays? Since you can put any size of data in each slot, how can the interpreter or compiler calculate A[i] efficiently?
  - Each slot contains pointers to the data and all pointers are the same size.

## Functions as data types

- Functions can also be a data type that can be passed as parameters and assigned to variables
- Easy with Python, but for statically typed languages, you have to declare the type of the function. How do you specify the function type?

```
def foo (x):
  print 3+x

bar = foo
bar(5) #prints 8

def baz(fn):
  fn(6)

baz(bar) #prints 9
```

- Example - C (show functionPointer.c)

```
#include <stdio.h>

void print_int (int i) {
     printf("%d\n", i);
}


int main() {

     // decalre a function pointer
     void (*func)(int);

     // initialize a function pointer
     func = print_int;

     // call a function pointer
     func(10);

     return 0;
}
```

- Why would you want to pass a function as parameters?
  - convenience; polymorphism in C
- Example - C qsort (show qsortExample.c)
  - `#include <stdlib>`

```
/**
 * Example of qsort: sort parts of a drone in ascending order in terms of price
 * Ying Li
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
        char name[50];
        int price;
} part;


int compare (const void* p1, const void* p2) {
        int m = ((part *)p1)->price;
        int n = ((part *)p2)->price;
        return m-n;
}
```

```c
int main () {
        part drone[6];

        strcpy(drone[0].name, "DJI Flamewheel Kit");
        drone[0].price = 299;

        strcpy(drone[1].name, "Navio2");
        drone[1].price = 117;

        strcpy(drone[2].name, "Power Module");
        drone[2].price = 22;

        strcpy(drone[3].name, "Receiver");
        drone[3].price = 30;

        strcpy(drone[4].name, "Batteries");
        drone[4].price = 60;

        strcpy(drone[5].name, "Sonar");
        drone[5].price = 25;

        qsort (drone, 6, sizeof(part), compare);

        for (int i = 0; i < 6; i++) {
                printf("%s: %d\n\n", drone[i].name, drone[i].price);
        }

        return 0;
}
```

- `void qsort(void *base, size_t nmemb, size_t size, int(*compar) (const void *, const void *));`
- return value of compare function
  - < 0 the first parameter is smaller than the second one (first placed before second)
  - = 0 two parameters are the same
  - > 0 the first parameter is larger than the second one (second placed before first)

---

## Polymorphism

- A function is polymorphic if ???
  - It can be applied to several different types of data
- Why function polymorphism?
  - Avoid code duplication
- How is it implemented?
  - Depends
    - C uses the `void *` data type and function pointers to provide polymorphism (qsort is an example)

- Java uses generic type (Stack.java)
- How do you implement a generic stack in Python? Python is a dynamically typed language, it supports different data types naturally (Stack.py).
  - List in Python can be used as stack.
  - Its append() method can be used as push, adding to the top of a stack
  - Its pop() method is the same as stack pop, removing elements in a LIFO order.

```
"""
Generic Stack
Ying Li
"""

class Stack:

    def __init__ (self):
        self.stack = []

    def push (self, item):
        self.stack.append(item)

    def pop (self):
        if not self.isEmpty():
            return self.stack.pop()
        else:
            return None

    def peek (self):
        if not self.isEmpty():
            return self.stack[len(self.stack)-1]
        else:
            return None

    def isEmpty(self):
        return len(self.stack) == 0


def main ():
    stk = Stack()

    print(stk.pop())

    for i in range (0, 5):
        stk.push(i)

    print("top is: ", stk.peek())

    for i in range (0, 5):
        print("pop: ", stk.pop())

    print(stk.peek())


if __name__ == '__main__':
    main()
```

```java
/**
 * Generic Stack
 *
 * Ying Li
 */
class Stack<T>{
    Node<T> top;

    T pop () {
      if (top != null) {
            T item = top.data;
            top = top.next;
            return item;
      }
      return null;
    }

    void push (T item) {
      Node<T> t = new Node<T> (item);
      t.next = top;
      top = t;
    }

    T peek () {
      return top.data;
    }

    boolean isEmpty () {
      if (top == null) {
            return true;
      } else {
            return false;
      }
    }

    public static void main (String[] args) {
      Stack<Integer> s = new Stack<Integer>();
      s.push(3);
      s.push(4);
      s.push(5);
      while (!s.isEmpty()) {
            System.out.println(s.pop());
      }
    }
}


class Node<T> {
    Node<T> next = null;
    T data;

    public Node (T d) {
      data = d;
    }
}
```

- Polymorphism in OO language is not just function polymorphism.
  - Inheritance
    - A class inherits characteristics from parent classes
  - Override
    - A subclass redefine the method inherited from a super class
    - Overriding vs overloading
      - Overriding: same method name, same parameters in different class (run-time concept)
      - Overloading: same method name, different parameters in one class (compile-time concept); Java supports overloading but Python doesn't.
  - Abstract class
    - Can be inherited only, cannot have instances
    - Provide default function prototype for the subclasses.
    - Java: declared by using the keyword, abstract
      - Comparing with Interfaces, abstract classes have some implementation.
    - Python: a class with at least one abstract method
      - abstract method: a method that has declaration but not implementation
  - Multiple inheritance:
    - Inherit more than one class (Python class can inherit from multiple classes)
    - Java can allow one subclass extend from one super class, but it allows a class implement multiple interfaces.

```java
/**
 * File: ConversionII.java
 * Author: Ying Li
 */

public class ConversionII {

      public static void main (String[] argv) {
            Parent p = new Parent(100);
            Child c = new Child(101);
            Human h1 = p;
            Human h2 = c;

            h1.print();
            h2.print();
      }
}

abstract class Human {
    protected int id;

    public void setID (int id) { this.id = id; }
    public abstract boolean isHappy ();
    public void print () { System.out.println(id + ", " + isHappy()); }
}

class Parent extends Human {

    public Parent (int id) {
        setID (id);
    }

      public boolean isHappy () { return false; }

      // overloading
      public boolean isHappy (int age) {
          if (age > 68) { return true; }
          else { return false; }
      }

      // overriding
      public void print () { System.out.printf("ID %d is %s\n", id, isHappy()); }
}

class Child extends Human {

    public Child (int id) {
        setID (id);
    }

      public boolean isHappy () { return true; }
}
```