

Types (II)

Implicit conversions (cont.)

```
/**
 * conversion loses precision
 * conversion2.c
 */
#include <stdio.h>

int main () {
    float f = 4.3;
    int i = f;
    char c = i;

    printf("i = %d, c = %d\n", i, c);

    return 0;
}
```

- Java has much more complicate rules, since it needs to define the rules not only for primitive types but also non-primitive types, for example, the conversion between super type and subtype.
 - Show the ConversionI.java, ask what is the output of the program. [Run-time error at downcasting. Parent cannot be cast to Child. How do you avoid? Use if (p1 instanceof Child)]

```
/**
 * File: ConversionI.java
 * Author: Ying Li
 */
public class ConversionI {

    public static void main (String[] argv) {
        Parent p1 = new Parent(100);
        System.out.println("p1 is " + p1);
        System.out.println("p1 is happy " + p1.isHappy());

        Child c1 = new Child(101);
        System.out.println("c1 is " + c1);
        System.out.println("c1 is happy " + c1.isHappy());

        Parent p2 = new Child(102); // upcasting
        System.out.println("p2 is " + p2);
        System.out.println("p2 is happy " + p2.isHappy());

        Child c3 = (Child) p2;
        System.out.println("c3 is " + c3);
        System.out.println("c3 is happy " + c3.isHappy());

        Child c2 = (Child) p1; // downcasting
        System.out.println("c2 is " + c2);
        System.out.println("c2 is happy " + c2.isHappy());
    }
}
```

```
class Parent {  
    protected long id;  
    public Parent (long id) { this.id = id; }  
    public String toString () { return Long.toString(id); }  
    public boolean isHappy () { return false; }  
}  
  
class Child extends Parent {  
    public Child (Integer id) { super(id); }  
    public boolean isHappy () { return true; }  
}
```

- If change Child c2 = p1, what is the output? [Compiler-time error]
- Downcasting and upcasting in Java
 - Downcasting and upcasting are applied on objects.
 - **Upcasting**, cast **subtypes to super types**, **implicit**. Parent p2 = (Parent) new Child(102); Parent p2 = new Child(102)
 - **Downcasting**, cast **super types to subtypes**, **explicit**. May cause java.lang.ClassCastException. Child c2 = (Child) p1
- Can use **instanceof** to decide the type of instance before downcasting

```
if (p1 instanceof Child) {  
    Child c2 = (Child) p1;  
    System.out.println("c2 is " + c2);  
    System.out.println("c2 is happy " + c2.isHappy());  
}
```

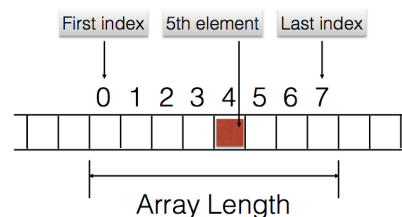
Pointers

- What is a pointer?
 - Pointers are a **data type** that enables explicit memory addressing.
 - A pointer **holds the address** of a variable of a certain type.
 - A pointer **does not hold data**, although the address itself can be treated as data in some situations.
- Dereference operator
 - The dereference operator **applies only to pointers**
 - The dereference operator **provides the value of the variable at the address** specified by the pointer
- Strongly typed languages vs. weakly typed languages

- In a strongly typed language, the pointer must have the same base type as the variable it addresses
- In a weakly typed language, the pointer does not have to be the same type as the variable it addresses
- Do all languages have pointers?
 - All languages have pointers.
 - Not all languages call them pointers or give the user explicit operators for managing them.
 - C/C++ give users the explicit operators for managing pointer. Star (*) is used to define a pointer, and ampersand (&) is used to dereference.
 - Java does not provide explicit operators. But all variables, except the variables of the primitive type are pointers. Instead of using the word pointer, Java uses reference, and references are strongly typed. Dereferencing in Java is automatic.

Arrays

- What is an array?
 - Arrays are indexed sequences of elements stored in chunks of continuous memory.
 - An array can be thought as a virtual grid of a particular type size laid over memory.



- Why use arrays? What are the advantages?
 - Efficient use of space and random access in constant time
- How can it access any location $A[i]$ in constant time?
 - On the hardware level, just need to find the address in memory of $A[i]$.
 - $\text{address of } A[i] = \text{address-of-}A + i * \text{slot-size}$
- Array in C and Pointer Arithmetic
 - Arrays and Pointers are the same thing in C [show pointers.c]
 - You allocate some memory and assign it to a pointer and that memory is automatically an array.
 - You can use array syntax or pointer syntax to access the slots in the array.

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

int main (int argc, char *argv[]) {
    int i;
    int *x = malloc(sizeof(int) * 6);

    x[0] = 1;
    *(x + 1) = 2;

    i = 2;
    x[i] = 3;

    i++;
    i[x] = 4;

    i++;
    *(x + i) = 5;

    //printf("addr(x): %p\n", x);
    x += 5;
    //printf("addr(x): %p\n", x);
    *x = 6;
    x -= 5;

    printf("%d %d %d %d %d %d\n", x[0], x[1], x[2], x[3], x[4], x[5]);

    free(x);

    return (0);
}
```

*The pointer's value is incremented/
decremented by the units of the type size*

*the size of an int is 4 bytes, then the
expression `x += 5` increment x by 20*