

Types

CS 208 - Bucknell University

Spring 2012

Outline

Primitive Types

Type Coercion

Strongly/Weakly Typed

Static/Dynamic Type

What is a type ?

What is a type ?

Definition: A type is a set of values, operations on the set, a storage configuration.

What is a type ?

Definition: A type is a set of values, operations on the set, a storage configuration.

Example: booleans in Java are true and false. Their operators include `&&` , `||` and `!` . They occupy one bit in memory (according to Java O'Reilly 2.0) and are thus stored as 1 and 0.

Primitive types

Definition: A primitive type comes with the language.

Definition: A constructed type (no official name, aka non-primitive) is one that you, the programmer, get to create using a mechanism called a type constructor.

Primitive types

Definition: A primitive type comes with the language.

Definition: A constructed type (no official name, aka non-primitive) is one that you, the programmer, get to create using a mechanism called a type constructor.

Question: What primitive types are in Java?

Primitive types

Definition: A primitive type comes with the language.

Definition: A constructed type (no official name, aka non-primitive) is one that you, the programmer, get to create using a mechanism called a type constructor.

Question: What primitive types are in Java?

Answer: int, boolean, float, double, long, char, byte, short.

Languages have different primitive types

C has int, char, short, long int, long long int, float, double, long double. (note no boolean and more float options than Java). In C, use sizeof to get the size of primitive types.

```
// gcc -o CTypeSizes CTypeSizes.c
// CTypeSizes
#include <stdio.h>
int main() {
    printf("On THIS machine...\n");
    printf("char is %d bytes\n", sizeof(char));
    printf("short is %d bytes\n", sizeof(short));
    printf("int is %d bytes\n", sizeof(int));
    printf("long int is %d bytes\n", sizeof(long int));
    printf("long long int is %d bytes\n", sizeof(long long int));
    printf("float is %d bytes\n", sizeof(float));
    printf("double is %d bytes\n", sizeof(double));
    printf("long double is %d bytes\n", sizeof(long double));
}
```

Primitive types in C

On a 32-bit machine, the int is likely to be 32 bits. On a 64-bit machine it could be 64 bits. The only size guarantees are

- $\text{char} \leq \text{short} \leq \text{int} \leq \text{long int} \leq \text{long long int}$
- $\text{float} \leq \text{double} \leq \text{long double}$

For example at Bucknell, linuxremote1 is a 64-bit machine. linuxremote-32 is a 32-bit machine. They will give different answers for some types.

Primitive types in C

On a 32-bit machine, the int is likely to be 32 bits. On a 64-bit machine it could be 64 bits. The only size guarantees are

- $\text{char} \leq \text{short} \leq \text{int} \leq \text{long int} \leq \text{long long int}$
- $\text{float} \leq \text{double} \leq \text{long double}$

For example at Bucknell, linuxremote1 is a 64-bit machine. linuxremote-32 is a 32-bit machine. They will give different answers for some types.

Lesson: Some language guarantee the size of a type, some don't.

Portability

Definition: Portability is the ability to run the same program on many machines and operating systems without tailoring it to each arch/OS. Even better is the ability to compile it once and then run it everywhere.

Portability

Definition: Portability is the ability to run the same program on many machines and operating systems without tailoring it to each arch/OS. Even better is the ability to compile it once and then run it everywhere.

The unguaranteed sizes of C are one of many features that make C code non-portable. What if we really need them to be a certain size? For a given C compiler and machine and OS, figure out the C type that works for you. Give it a nickname....

```
typedef int S32; // S32 now means the same as the type int
S32 x = 12;
```

How does this help us? We can put all the size information in one file called a header file and use S32 everywhere. We only need to change out the one header file for each machine instead of changing all the types in every files. Operating systems use this strategy.

Memory packing algorithm

Definition: A memory packing algorithm defines how declared variables are stored in memory. (Right next to each other or with space inbetween.)

Memory packing algorithm

Definition: A memory packing algorithm defines how declared variables are stored in memory. (Right next to each other or with space inbetween.)

The following program prints the starting address of variables:

```
// gcc -o CPacking CPacking.c
// CPacking
#include <stdio.h>
int main() {
    // These pack nicely on a 32 bit boundary
    int a;
    short b;
    short c;

    printf("%x int a\n", &a); // &a is the address of a
    printf("%x short b\n", &b); // %x prints an address in hexid
    printf("%x short c\n\n", &c);
}
```

Output of the program

On linuxremote-32:

```
bf94a4c0 int a  
bf94a4be short b  
bf94a4bc short c
```

On the MacBook:

```
5fbff978 int a  
5fbff97e short b  
5fbff97c short c
```

Activity

Copy all the programs in the following directory

```
~cs208/2012-spring/student/programs/types/
```

Compile CPacking2.c and run it. Draw the memory.

Type Coercion

Type coercion

Definition: A type coercion is when a value is changed from one type to another. There are two kinds of type coercion:

1. Implicit coercion
2. Explicit coercion

The type coercion may change the representation of the value.

Implicit coercion

What happens in Java with

```
double x = 3;  
double y = 4 * 5.2;  
int z = 6.8;
```

Implicit coercion

What happens in Java with

```
double x = 3;  
double y = 4 * 5.2;  
int z = 6.8;
```

Definition: Implicit coercion is when the *programming language* changes a *value* from one type to another. (Not when it changes the type of a variable, that's something else altogether...).

Implicit coercion

What happens in Java with

```
double x = 3;  
double y = 4 * 5.2;  
int z = 6.8;
```

Definition: Implicit coercion is when the *programming language* changes a *value* from one type to another. (Not when it changes the type of a variable, that's something else altogether...).

What implicit coercion is allowed? char to int? boolean to int?

Rule of Thumb: "Don't lose info".

Implicit coercion

Activity: Make a table with all primitives on top and the side. Side is "from". Top is "to". Check the box if the implicit coercion is allowed.

For example:

```
char c = 'a';
short s = 2;
s = c; // try to implicitly coerce a char to a s
```

	boolean	byte	char	short	int	float	long	double
1 boolean								
1 byte								
2 char								
2 short								
4 int								
4 float								
8 long								
8 double								

Implicit coercion

Activity: Make a table with all primitives on top and the side. Side is "from". Top is "to". Check the box if the implicit coercion is allowed.

For example:

```
char c = 'a';
short s = 2;
s = c; // try to implicitly coerce a char to a s
```

	boolean	byte	char	short	int	float	long	double
1 boolean	✓							
1 byte		✓		✓	✓	✓	✓	✓
2 char			✓		✓	✓	✓	✓
2 short				✓	✓	✓	✓	✓
4 int					✓	✓	✓	✓
4 float						✓		✓
8 long						✓	✓	✓
8 double								✓

Conclusion on implicit coercion

Rule of Thumb: “Don’t lose information”.

Java allows to implicitly coerce when it is sure that the conversion will not lose information. Floating point values are inclined to rounding error.

Explicit coercion

What happens in Java with

```
double x = (double) 3;  
int y = (int) 5.2;  
byte b1 = (byte) 5.3e17;
```

Explicit coercion

What happens in Java with

```
double x = (double) 3;  
int y = (int) 5.2;  
byte b1 = (byte) 5.3e17;
```

Definition: Explicit coercion (cast) is when the *programmer* explicitly ask to change a value from one type to another.

Explicit coercion

What happens in Java with

```
double x = (double) 3;  
int y = (int) 5.2;  
byte b1 = (byte) 5.3e17;
```

Definition: Explicit coercion (cast) is when the *programmer* explicitly ask to change a value from one type to another.

What explicit coercion is allowed?

Explicit coercion

Activity: same protocol

For example:

```
char c = 'a';
```

```
short s = 2;
```

```
s = (short) c; // try to explicitly coerce a char to a short
```

	boolean	byte	char	short	int	float	long	double
1 boolean								
1 byte								
2 char								
2 short								
4 int								
4 float								
8 long								
8 double								

Explicit coercion

Activity: same protocol

For example:

```
char c = 'a';
```

```
short s = 2;
```

```
s = (short) c; // try to explicitly coerce a char to a short
```

	boolean	byte	char	short	int	float	long	double
1 boolean	✓							
1 byte		✓	✓	✓	✓	✓	✓	✓
2 char		✓	✓	✓	✓	✓	✓	✓
2 short		✓	✓	✓	✓	✓	✓	✓
4 int		✓	✓	✓	✓	✓	✓	✓
4 float		✓	✓	✓	✓	✓	✓	✓
8 long		✓	✓	✓	✓	✓	✓	✓
8 double		✓	✓	✓	✓	✓	✓	✓

Conclusion on explicit coercion

Java allows to explicitly coerce any primitive type to any other primitive type (except for booleans). The explicit coercion may can a lot the original value. This is a source of **bugs**.

Untyped/Weakly/Strongly Typed

Untyped/Weakly/Strongly Typed

Definition: A programming language is strongly typed when every value is given a type and there is no implicit coercion from one type to another.

One definition strongly typed involves preventing success for an operation on arguments which have the wrong type.

Untyped/Weakly/Strongly Typed

Definition: A programming language is strongly typed when every value is given a type and there is no implicit coercion from one type to another.

One definition strongly typed involves preventing success for an operation on arguments which have the wrong type.

Definition: A programming language is weakly typed when it allows to implicitly convert (or casts) types if necessary.

Untyped/Weakly/Strongly Typed

Definition: A programming language is strongly typed when every value is given a type and there is no implicit coercion from one type to another.

One definition strongly typed involves preventing success for an operation on arguments which have the wrong type.

Definition: A programming language is weakly typed when it allows to implicitly convert (or casts) types if necessary.

Definition: A programming language is Untyped when there is no type checking. There can still be types but the rules are not enforced. It is up to the programmer to ensure that data given to functions is of the appropriate type. Any type conversion required is explicit.

Untyped/Weakly/Strongly Typed

Example:

```
var x := 5;  
var y := "37";  
print(x+y);
```

- In a strongly typed language, this does not make sense.
- In a weakly typed language, we could get some interesting results. Visual Basic would notice that the string only contains digits, implicitly coerce it to an int and then add to get 42. It is possible that we might want to implicitly coerce 5 to a string and use string concatenation as in Java.
- In an untyped language, there really are no types and every operation on every value will make something happen. You just may not like the results.

Untyped/Weakly/Strongly Typed

Untyped	Weakly Typed		Strongly Typed
Forth	C	Java	Ada
Assembly		Pascal	Smalltalk
		C#	Ruby
			Python
			LISP
			Haskell

Statically/Dynamically Typed

Statically Typed or Dynamically Typed Languages

According to Wikipedia (http://en.wikipedia.org/wiki/Type_system):

Statically Typed or Dynamically Typed Languages

According to Wikipedia (http://en.wikipedia.org/wiki/Type_system):

Definition: “A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Statically typed languages include Ada, AS3, C, C++, C#, Eiffel, F#, Go, JADE, Java, Fortran, Haskell, ML, OCaml, Objective-C, Pascal, Perl (with respect to distinguishing scalars, arrays, hashes and subroutines) and Scala.

Statically Typed or Dynamically Typed Languages

According to Wikipedia (http://en.wikipedia.org/wiki/Type_system):

Definition: “A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Statically typed languages include Ada, AS3, C, C++, C#, Eiffel, F#, Go, JADE, Java, Fortran, Haskell, ML, OCaml, Objective-C, Pascal, Perl (with respect to distinguishing scalars, arrays, hashes and subroutines) and Scala.

Static typing is a limited form of program verification (type safety): accordingly, it allows many type errors to be caught early in the development cycle. Static type checkers evaluate only the type information that can be determined at compile time, but are able to verify that the checked conditions hold for all possible executions of the program, which eliminates the need to repeat type checks every time the program is executed. Program execution may also be made more efficient (i.e. faster or taking reduced memory) by omitting runtime type checks and enabling other optimizations.”

Statically Typed or Dynamically Typed Languages

According to Wikipedia (http://en.wikipedia.org/wiki/Type_system):

Statically Typed or Dynamically Typed Languages

According to Wikipedia (http://en.wikipedia.org/wiki/Type_system):

Definition: “A programming language is said to be dynamically typed when the majority of its type checking is performed at run-time as opposed to at compile-time.

In dynamic typing, values have types but variables do not; that is, a variable can refer to a value of any type.

Dynamically typed languages include Erlang, Groovy, JavaScript, Lisp, Lua, Objective-C, Perl (with respect to user-defined (constructed) types but not built-in types), PHP, Prolog, Python, Ruby, Smalltalk and Tcl.”

Statically Typed or Dynamically Typed Languages

Example:

- Java is statically typed so variables carry the type

```
int x = 3;
```

```
x = "hi";
```

x has type int. This means x cannot be given the value "hi" so this code is bad.

Statically Typed or Dynamically Typed Languages

Example:

- Java is statically typed so variables carry the type

```
int x = 3;
```

```
x = "hi";
```

x has type int. This means x cannot be given the value "hi" so this code is bad.

- Smalltalk is dynamically typed so the values carry the type

```
x := 3.
```

```
x := 'hi'.
```

3 and 'hi' have types int and string. x has no type so this code is good.

Statically Typed or Dynamically Typed Languages

What are the pros and cons of static typing and dynamic typing ?

Statically Typed or Dynamically Typed Languages

What are the pros and cons of static typing and dynamic typing ?

- Pro Statically Typed: Type checking is run only once at the compilation. Whereas with dynamic typing the program do the type checking while it executes, it is less efficient.

Statically Typed or Dynamically Typed Languages

What are the pros and cons of static typing and dynamic typing ?

- Pro Statically Typed: Type checking is run only once at the compilation. Whereas with dynamic typing the program do the type checking while it executes, it is less efficient.
- Pro Statically Typed: Type checking ensures the type correctness of the program. When the program is dynamically typed, a type error will be signaled at run-time.

Statically Typed or Dynamically Typed Languages

What are the pros and cons of static typing and dynamic typing ?

- Pro Statically Typed: Type checking is run only once at the compilation. Whereas with dynamic typing the program do the type checking while it executes, it is less efficient.
- Pro Statically Typed: Type checking ensures the type correctness of the program. When the program is dynamically typed, a type error will be signaled at run-time.
- Pro Dynamically Typed: More flexible, some valid programs could be rejected at compile time for no reason.