**Competencies**

**Section 7.1**

What is a type system?

Informally, a *type system* consists of (1) a mechanism to define types and associate

them with certain language constructs and (2) a set of rules for *type equivalence*,

*type compatibility*, and *type inference*.

What is explicit parametric polymorphism?

also provide *explicit parametric polymorphism (generics)*, which allow the programmer

to define classes with type parameters.

**(Q)**What the

- **composite types arrays,**

are the most commonly used composite types. An array can be thought

of as a function that maps members of an *index* type to members of a *component*

type. Arrays of characters are often referred to as *strings*, and are often

supported by special purpose operations not available for other arrays.

- **records**,

A record consists of a collection of *fields*, each of whichbelongs

to a (potentially different) simpler type. Records are akin to mathematical *tuples*;

a record type corresponds to the Cartesian product of the types of the

fields.

**- variant records (unions),**

differ from “normal” records in that only one of a variant

record’s fields (or collections of fields) is valid at any given time. A variant

record type is the *union* of its field types, rather than their Cartesian product.

-pointers

are l-values. A pointer value is a *reference* to an object of the pointer’s

base type. Pointers are often but not always implemented as addresses. They

are most often used to implement *recursive* data types. A type *T* is recursive

if an object of type *T* may contain one or more references to other objects of

type *T*.

**Section 7.2**

What are the differences between structural equivalence and name equivalence?

Name equivalence is the most straightforward: two types are equal if, and only if, they have the same name. Thus, for example, in the code (using C syntax).

Structural equivalence,  two types are equal if, and only if, they have the same "structure",

**What is type casting?**

Typecasting enables data type conversion. C++ supports Implicit conversions and Explicit conversion. Implicit conversions automatically performed when a value is copied to a compatible type. If there is an operation between an int and a float, the int is promoted to float before performing operation.

You can cast types explicitly as follows.

**(Q)**What is type coercion?

Whenever a language allows a value of one type to be used in a context that expects

another, the language implementationmust performan automatic, implicit conversion to the expected type. This conversion is called a *type coercion*

**(Q)**What are universal reference types?

To facilitate the writing of general-purpose container (collection) objects(list,stacks ect) that hold references to other objects

**Section 7.3**

What is a record field?

The fields of a record are usually stored in adjacent locations in memory. In its

symbol table, the compiler keeps track of the offset of each field within each

record type. When it needs to access a field, the compiler typically generates a

load or store instruction with displacement addressing. For a local object, the

base register is the frame pointer; for a global object, the base register is the globals

pointer. In either case, the displacement is the sum of the record’s offset from

the register and the field’s offset within the record.

**(Q)**What is a record hole?

What is a word or double word alignment?

A word is the amount of data which each register in the CPU is able to hold.

The size of a double word is just the size of a word \* 2.

What is a Union?

How are variants (unions) mapped?

**(Q)**Page 325, Problems 22 and 25.

What are two purposes for using unions (Page 325, Problem 26)?

**Section 7.4**

What is an array index?

Is a collection that uses to localize the data stored

Is the distance from the base address

**(Q)**Section 7.4.2: static versus dynamic allocation

**(Q)**Page 342, Problem 31.

What are row major order and column major order?

In row-

Row-major v.

column-major array layout

major order, consecutive locations in memory hold elements that differ by one

in the final subscript (except at the ends of rows). A[2, 4], for example, is followed

by A[2, 5].

In column-major order, consecutive locations hold elements

that differ by one in the *initial* subscript: A[2, 4] is followed by A[3, 4]. These

options are illustrated for two-dimensional arrays in Figure 7.9.

How are array elements addressed in contiguous allocation?

Elements of data are logically stored sequentially in blocks within the array.Why do many languages use 0 based lower bounds?

**(Q)**Compare contiguous allocation (as in C) to row-pointer layout (as in Java) (Page 342, Problem 33).

**(Q)**Compare rectangular arrays to ragged arrays.

**Section 7.5**

**(Q)**Statically allocated strings: fixed length, sentinel

 the length of fixed-length strings is determined at compile-time, not run-time.  In addition, unlike dynamic strings, fixed-length strings do not use handles.  When you pass a fixed-length string to a [procedure](javascript:void(0);) as a parameter, you are actually passing a [pointer](javascript:void(0);) to the string data.

**(Q)**Dynamically allocated strings (reference model)

**(Q)**Immutable strings - the string object does not change.  Any operation create a new string object, it does not modify the operand object.

**Section 7.7**

**(Q)**How are pointers and references the same?

They point a location in memory

**(Q)**How are pointers and references different? Why are references safer? Why are pointers useful?

1. A pointer can be re-assigned any number of times while a reference can not be re-seated after binding.

What are the operators "address of" and "dereference"?

The **dereference operator** or **indirection operator**, denoted by "\*" (i.e. an [asterisk](http://en.wikipedia.org/wiki/Asterisk)), is a [unary operator](http://en.wikipedia.org/wiki/Unary_operator) found in [C](http://en.wikipedia.org/wiki/C_(programming_language))-like languages that include [pointer](http://en.wikipedia.org/wiki/Pointer_(computer_programming)) variables.

## - **Definition - What does *Address-of Operator (&)* mean?**

An address-of operator is a mechanism within C++ that returns the [memory](http://www.techopedia.com/definition/25580/address-of-operator--c)address of a variable. These addresses returned by the address-of operator are known as pointers, because they "point" to the variable in memory.  
  
The address-of operator is a unary operator represented by an ampersand (&). It is also known as an address operator.

**(Q)**What is pointer arithmetic?  Why is it dangerous? Why is it useful?

Is uses a arithmetic operation to go to the desired array location. Because data can differ insize and also in can point to a location outside of the array . It gives

**(Q) What is a dangling reference? What causes a dangling reference? (Page 370, Problem 44).**

**(Q) What is garbage? How is it created and why is it a problem?**

**How do garbage collection and explicit deallocation differ?**

**(Q) Page 370, Problem 49.**

**(Q) Page 370, Problem 51.  Consider the problems with having a pointer to an automatic variable such as a local variable.  What can go wrong?**

**Topics**

**This is a comprehensive, but not complete, list of topics.**

**Section 7.1**

Type systems

Section 7.1.1: Strongly typed, static typing, statically typed, dynamic typing.

Polymorphism: parametric, subtype

Types: denotational, constructive, abstraction-based

Classification of types: numeric, enumeration, subrange, character

Composite types: structures, variant records, arrays, sets, pointers, lists, files

**Section 7.2**

Type equivalence, compatability, inference

Name equivalence, structure equivalence

Type conversion, casting, coercion

Converting types casts, nonconverting type casts

**Section 7.3**

Records and variants: syntax, operations, memory layout, memory alignment

With statements

**Section 7.4**

Arrays: index type, zero-based, syntax, memory layout, arrays of arrays, two-dimensional, object-oriented arrays

Slices, array operations

Array allocation: contiguous and row-pointer; what are the advantages and disadvantages?

**Section 7.5**

Strings: memory layout, sizing

Operations

Immutable objects (C# and Java)

**Section 7.6**

Sets: operations, implementation, bit-vectors, arrays, lists, has tables, trees

**Section 7.7**

Pointers: syntax, operations, versus references

Pointers and C arrays

Dangling references and garbage

Explicit and implicit deallocation

**Section 7.8**

Lists: car and cdr versus head and tail, append

**Section 7.10**

Equality testing

Assignment, reference types, structured types