**Data Types– Part 3**

**Slide 1**

In the third and final part of this presentation on data types, we will discuss strings, records and pointers.

**Slide 2**

We begin with strings.

In Ada, the string type is defined as an array of characters. One benefit of that approach is that the notation for selecting an array subscript can be used to access individual characters of an array. In Ada, parentheses are used to enclose subscripts rather than more customary square brackets.

Ada also provides a syntax for array slices, which when used with strings, is a technique for extracting a substring.

In C, strings are also arrays of characters although in C, there is no associated type name. Such strings are terminated by a null character, which makes traversing all characters possible without knowing the size of array.

In both C++ and Java, strings are objects of a predefined class. In Java string objects are immutable, which is not true in C++.

C++ allows operator overloading. So the C++ string class is able to overload all of the relational operators. In addition, brackets are considered an operator, and are overloaded to allow strings to be treated as though they were arrays. In Java, a method charAt must be used to access individual characters inside a string.

**Slide 3**

This Ada program consists of a function that accepts two strings as parameters and returns whether first string is a substring of the second.

The Ada String type is an unconstrained array type, so when an array variable is declared, its bounds must be specified.

Here is an example of the use of a slice, which is specified by the first and last subscripts separated by two dots. Because strings are arrays, all of the array attributes are available, in this case, the attribute for determine the length.

In Ada, the ampersand is the operator used for string concatenation.

**Slide 4**

Languages that predate the move to object-orientation had records or structures.

That applies to both Pascal and Ada. Although a later version of Ada, Ada 95, is object-oriented, it does not have classes, like the ones in C++ or Java.

Records are compound data types, meaning that they contain multiple values. The fields as they are often called, are accessed by name.

C structure types were originally like records and could contain only data.

Although C++ retained structure types, the only difference between a struct and a class in C++ is that in a struct the members are public by default and in a class they are private by default. Such a difference is insignificant and not an ideal language design. For that reason Java dispensed with structures altogether.

By contrast C# retained the structure type, but in C# there is a significant difference between a struct and a class. Objects of a structure type are allocated on the stack whereas, objects of a class are allocated on the heap. Inheritance is not permitted with struct types, which avoids a kind of undesirable behavior that can happen with objects in C++, which is object slicing—a topic we will discuss in conjunction with inheritance.

**Slide 5**

Like records and structures, languages that predated the inclusion of inheritance needed a mechanism to define data types that had several variants. Both unions and variant records accomplish that goal.

C and C++ have free unions. By contrast, neither Java nor C# included them.

In a free union, the fields overlay one another in memory.

Using unions can be risky because they allow data of one type to be accessed as another type. By nesting a union inside a struct, it is possible to duplicate a variant record, but without the type checking that can be performed with variant records.

Both Pascal and Ada have variant records. Because Ada 83 did not support inheritance, variant records were needed.

The variant part of a variant record contains a CASE structure that allows a collection of variant fields to be selected.

Whenever a new case variant is added to such a record, the program requires significant modifications because wherever these records are accessed, a change is needed.

Inheritance obviates the need for variant records. Moreover it has the advantage that a new subclass can be added, which is akin to adding a new case variant, with far fewer changes.

**Slide 6**

To better understand the purpose of a variant record, let’s consider an Ada example.

Every variant record must have a field referred to as the tag. It is what is used to select a variant. In this example the type of the tag is Gender\_Type.

All records of this type contain the first two fields, Name and Age.

The variant part begins with the reserved word case. This record has two variants, either male or female. The male variant has the additional field Beard, whereas the female variant has no additional fields.

This is an example of an assignment to variant record that uses named association, which matches the field name with its value.

**Slide 7**

Finally we discuss pointer and reference types.

C only has pointers.

To access the value that a pointer points to, a pointer must be explicitly dereferenced using the unary asterisk symbol. That symbol is also used as a part of a pointer type.

C++ has both pointers and explicit references.

A reference is declared with an ampersand. Reference variables, unlike pointer variables, are explicitly dereferenced.

Java does not have pointers, only implicit references. Every object and array name is understood to contain a reference.

Like C++, the dereferencing is implicit whenever an array or object name is used.

Ada has pointers but they are called access types. When a pointer to record field is used, the dereferencing is implicit. Otherwise .all must be used.

**Slide 8**

C and C++ allow pointer arithmetic. The two functions below that have pointer parameters make use of it. Both functions return the sum of the elements in an integer array. Recall that in C and C++ arrays and pointers are interchangeable.

In this first function, pointer addition is being used. The benefit of this approach is that a multiplication is avoided that would be performed in the code generated to compute the array subscript. The increment operation does not add just one to the pointer but the size of one integer is added.

C and C++ allow two kinds of pointer subtraction, subtracting an integer from a pointer and subtracting one pointer from another. In the next function, subtraction of two pointers is used. It does not just subtract the address but it divides the difference by the size of an integer.

**Slide 9**

Finally we examine two versions of a swap function written in C++ to compare using pointer parameters with reference parameters. Both make it possible to preserve changes made to the values that the parameters point to, upon return to the caller.

The first version, swap1, uses pointers. Notice that the parameters must be declared as pointers, and each time one of the parameters is referenced in the body of the function, it must be dereferenced.

The second version, swap2, uses references. The parameters still must be declared as references, but no dereferencing is needed in the body of the function. For that reason, using references appears simpler.

In the call to swap1, the addresses of the two variables must be explicitly passed.

In the call to swap2, by contrast, does not require explicitly passing the addresses. Using references, it is implicit.