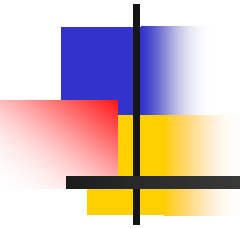
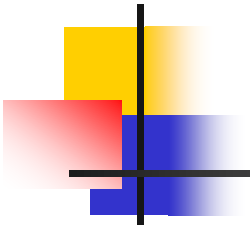
Data Types



Lecture 10

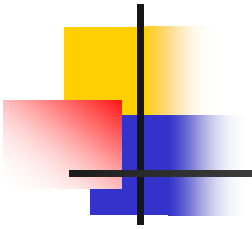
Instructor: C. Pu (Ph.D., Assistant Professor)

puc@marshall.edu



Introduction

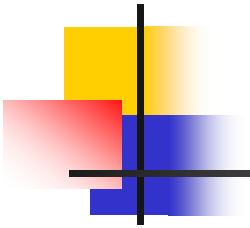
- A ***data type*** defines a ***collection of data values*** and a ***set of predefined operations*** on those values.
- Computer programs produce results by manipulating data.
- An important factor in determining the ease with which they can perform this task is how well the data types available in the language being used match the objects in the real-world of the problem being addressed.
- Therefore, it is crucial that a language supports an appropriate collection of data types and structures.



Primitive Data Types

- Data types that are not defined in terms of other types are called ***primitive data types***.

- Nearly all programming languages provide a set of primitive data types.
  - Some of the primitive types are merely *reflections* of the hardware—for example, most integer types.



Integer

- The most common primitive numeric data type is ***integer***.

- Many computers now support several sizes of integers.
- These sizes of integers, and often a few others, are supported by some programming languages.
- For example,
  - Java includes four signed integer sizes: ***byte***, ***short***, ***int***, and ***long***.

- Some languages, for example, C++ and C#, include unsigned integer types, which are simply types for integer values without signs.
  - https://msdn.microsoft.com/en-us/library/s3f49ktz.aspx

Integer

- Most integer types are supported directly by the hardware.
- One example of an integer type that is not supported directly by the hardware is the ***long integer*** type of Python.

```
# A Python program to demonstrate that we can store
# large numbers in Python

x = 1000000000000000000000000000000000000000000000;
x = x + 1
print (x)
```

Output :

```
1000000000000000000000000000000000000000000001
```

Complex

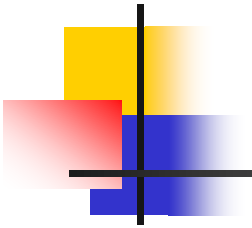
- Some programming languages support a complex data type—for example, Fortran and Python.
- Complex values are represented as ordered pairs of floating-point values.
- For example, in Python

```
# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

Decimal

- Most larger computers that are designed to support business systems applications have hardware support for decimal data types.
- Decimal data types store a fixed number of decimal digits, with the decimal point at a fixed position in the value.
- These are the primary data types for business data processing and are therefore essential to C# also have decimal data types.
  - For example in C#
    - The ***decimal*** keyword indicates a 128-bit data type.
    - Compared to other floating-point types, the decimal type has more precision, which makes it appropriate for financial and monetary calculations.

Type	Approximate Range	Precision
`decimal`	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / (10^{0}$ to $10^{28})$	28-29 significant digits



Boolean Types

- ***Boolean*** types are perhaps the simplest of all types.
- Their ***range*** of values has only two elements: one for ***true*** and one for ***false***.
- They were introduced in ALGOL 60 and have been included in most general-purpose languages designed since 1960.

Boolean Types

- One popular exception is C89, in which numeric expressions are used as conditionals.
  - In such expressions, all operands with ***nonzero*** values are considered ***true***, and ***zero*** is considered ***false***.

```
int a = 10, b = 20;

if (a)
    if (b)
        printf("value of a + b is %d\n", a + b);
```

- Although C99 and C++ have a Boolean type, they also allow numeric expressions to be used as if they were Boolean.

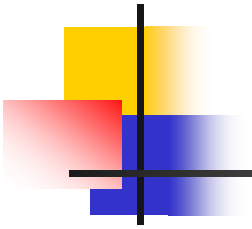
- This is not the case in the subsequent languages, Java and C#.

Character Types

- Character data are stored in computers as ***numeric coding***.
- Traditionally, the most commonly used coding was the 8-bit code ASCII (American Standard Code for Information Interchange), which uses the values 0 to 127 to code 128 different characters.
  - https://www.ascii-code.com/
- ISO 8859-1 is another 8-bit character code, but it allows 256 different characters. Ada 95+ uses ISO 8859-1.
  - https://cs.stanford.edu/people/miles/iso8859.html

Character Types

- Because of the globalization of business and the need for computers to communicate with other computers around the world, the ASCII character set became inadequate.
- In response, in 1991, the Unicode Consortium published the UCS-2 standard, a 16-bit character set.
  - This character code is often called ***Unicode***.
- After 1991, the Unicode Consortium, in cooperation with the International Standards Organization (ISO), developed a 4-byte character code named UCS-4, or UTF-32, which is described in the ISO/IEC 10646 Standard, published in 2000.



Character String Types

- A ***character string type*** is one in which the values consist of sequences of characters.
- Character string constants are used to label output, and the input and output of all kinds of data are often done in terms of strings.
- Of course, character strings also are an essential type for all programs that do character manipulation.

Character String Types: Strings and Their Operations

- The most common string operations are *assignment*, *catenation*, *substring reference*, *comparison*, and *pattern matching*.

- If strings are not defined as a primitive type, string data is usually stored in arrays of single characters and referenced as such in the language.
  - This is the approach taken by C and C++.

Character String Types: Strings and Their Operations

- C and C++ use ***char arrays*** to store character strings.
  - These languages provide a collection of string operations through standard libraries.
- Many uses of strings and many of the library functions use the convention that character strings are terminated with a special character, ***null***, which is represented with zero.
- The library operations simply carry out their operations until the ***null*** character appears in the string being operated on.
- Library functions that produce strings often supply the ***null*** character.

Character String Types: Strings and Their Operations

- The character string literals that are built by the compiler also have the ***null*** character.
- For example, consider the following declaration:

```
char str[] = "apples";
```

  - In this example, str is an array of char elements, specifically apples**0**, where **0** is the **null** character.

Character String Types: Strings and Their Operations

- Some of the most commonly used library functions for character strings in C and C++ are
  - ***strcpy***, which moves strings;

```
/* Create an example variable capable of holding 50 characters */
char example[50];

/* Copy the string "TechOnTheNet.com knows strcpy!" into the example variable */
strcpy (example, "TechOnTheNet.com knows strcpy!");
```

Character String Types: Strings and Their Operations

- Some of the most commonly used library functions for character strings in C and C++ are
  - ***strcat***, which contenates one given string onto another;

```
/* Define a temporary variable */
char example[100];

/* Copy the first string into the variable */
strcpy(example, "TechOnTheNet.com ");

/* Concatenate the following two strings to the end of the first one */
strcat(example, "is over 10 ");
strcat(example, "years old.");
```

Character String Types: Strings and Their Operations

- Some of the most commonly used library functions for character strings in C and C++ are
  - ***strcmp***, which lexicographically compares (by the order of their character codes) two given strings;

```
/* Create two arrays to hold our data */
char example1[50];
char example2[50];

/* Copy two strings into our data arrays */
strcpy(example1, "C programming at TechOnTheNet.com");
strcpy(example2, "C programming is fun");

/* Compare the two strings provided */
result = strcmp(example1, example2);
```

Character String Types: Strings and Their Operations

- Some of the most commonly used library functions for character strings in C and C++ are
  - ***strlen***, which returns the number of characters, not counting the null, in the given string.

```
char str[50];
int len;

strcpy(str, "This is tutorialspoint.com");

len = strlen(str);
```

Character String Types: String Length Options

- There are several design choices regarding the length of string values.
- First, the length can be static and set when the string is created. Such a string is called a ***static length string***.
  - In Java,
    - The String class is ***immutable***, so that once it is created a String object cannot be changed.
    - The String class has a number of methods, some of them appear to modify strings.
    - Since strings are immutable, what these methods really do is create and return a new string that contains the result of the operation.

```
String greeting = "Hello world!";
```

Character String Types: String Length Options

- The second option is to allow strings to have varying length up to a declared and fixed maximum set by the variable's definition, as exemplified by the strings in C and the C-style strings of C++.
  - These are called ***limited dynamic length strings***.
  - Such string variables can store any number of characters between zero and the maximum.

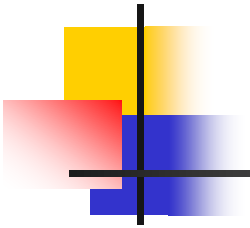
```
char str[14] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};
```

Character String Types: String Length Options

- The third option is to allow strings to have varying length with no maximum, as in JavaScript, Perl, and the standard C++ library.
  - These are called ***dynamic length strings***.

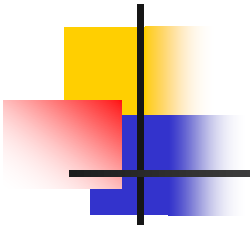
```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
var sln = txt.length;
```

```
var str = "Please locate where 'locate' occurs!";
var pos = str.indexOf("locate");
```



Array Types

- An ***array*** is a homogeneous aggregate of data elements in which an individual element is identified by its position in the aggregate, relative to the first element.
- The individual data elements of an array are of the same type.
- References to individual array elements are specified using subscript expressions.
- If any of the subscript expressions in a reference include variables, then the reference will require an additional run-time calculation to determine the address of the memory location being referenced.



Array Types

- In many languages, such as C, C++, Java, Ada, and C#, all of the elements of an array are required to be of the same type.
  - In these languages, pointers and references are restricted to point to or reference a single type.
  - So the objects or data values being pointed to or referenced are also of a single type.

- In some other languages, such as JavaScript, Python, and Ruby, variables are typeless references to objects or data values.
  - In these cases, arrays still consist of elements of a single type, but the elements can reference objects or data values of different types.
  - Such arrays are still homogeneous, because the array elements are of the same type.

Array Types: Arrays and Indices

- Specific elements of an array are referenced by means of a two-level syntactic mechanism
  - The *first part* is the aggregate name.
  - The *second part* is a possibly dynamic selector consisting of one or more items known as subscripts or indices.
  - If all of the subscripts in a reference are constants, the selector is static; otherwise, it is dynamic.
- The selection operation can be thought of as a mapping from the array name and the set of subscript values to an element in the aggregate.
- Indeed, arrays are sometimes called finite mappings. Symbolically, this mapping can be shown as

array_name(subscript_value_list) → element

Array Types: Arrays and Indices

- The syntax of array references is fairly universal:
  - The array name is followed by the list of subscripts, which is surrounded by either parentheses or brackets.
- In some languages that provide multi-dimensioned arrays as arrays of arrays, each subscript appears in its own brackets.
- A problem with using parentheses to enclose subscript expressions is that they often are also used to enclose the parameters in subprogram calls; this use makes references to arrays appear exactly like those calls.
- For example, consider the following Ada assignment statement:

```
Sum := Sum + B(I);
```

Array Types: Array Initialization

- Some languages provide the means to initialize arrays at the time their storage is allocated.
- In Fortran 95+, an array can be initialized by assigning it an array aggregate in its declaration.
- An array aggregate for a single-dimensioned array is a list of literals delimited by parentheses and slashes.
- For example, we could have

```
Integer, Dimension (3) :: List = (/0, 5, 5/)
```

- C, C++, Java, and C# also allow initialization of their arrays, but with one new twist: In the C declaration

```
int list [] = {4, 5, 7, 83};
```

Array Types: Array Initialization

- Character strings in C and C++ are implemented as arrays of char.
  - These arrays can be initialized to string constants, as in

```
char name [] = "freddie";
```

- Arrays of strings in C and C++ can also be initialized with string literals.
- In this case, the array is one of pointers to characters.
  - For example,

```
char *names [] = {"Bob", "Jake", "Darcie"};
```

Array Types: Array Initialization

- In Java, similar syntax is used to define and initialize an array of references to String objects.
- For example,

```
String[] names = ["Bob", "Jake", "Darcie"];
```

- Ada provides two mechanisms for initializing arrays in the declaration statement:
  - by listing them in the order in which they are to be stored
  - by directly assigning them to an index position using the => operator, which in Ada is called an arrow.
- For example, consider the following:

```
List : array (1..5) of Integer := (1, 3, 5, 7, 9);
Bunch : array (1..5) of Integer := (1 => 17, 3 => 34,
                                    others => 0);
```