Week 6 - Topic 1: Data, References, Arrays and Searching

Topic Overview

- Character data types
- Arrays, lists and searching
- Using references in Ruby
- Designing the Text Music Player
- Tests

ASCII

- ASCII (American Standard Code for Information Interchange).
- Used since 1960.
- It represents the 128 characters as found on standard keyboards.
- It uses only 7 of the 8 bits (the first bit is always zero).

ASCII Codes

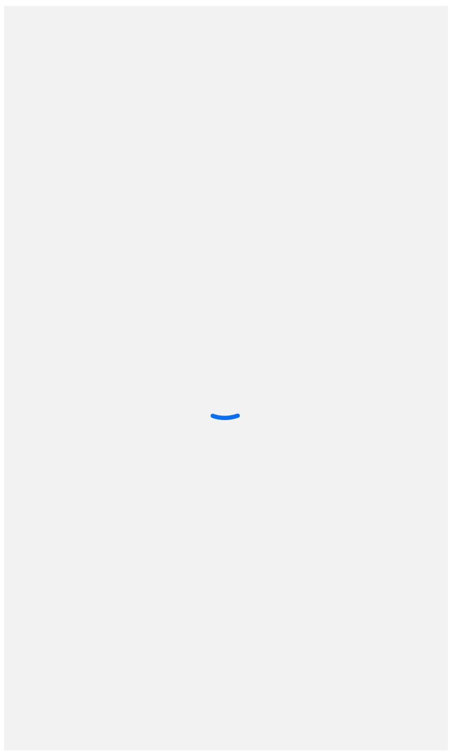
Run RUBY

```ruby
# Type a key and press "enter" to see the ASCII code:
s = gets.chomp
s_ascii = s.unpack("c*")
puts s_ascii
```

UTF-8

- But there are many more characters to be represented if we want to represent a wider set of characters (eg: Greek, Arabic, Chinese, Korean etc).
- To represent a more complete character set the Unicode standard was developed (in 1991).
- UTF-8 is one implementation of the Unicode standard. It is backwards compatible with ASCII (for non-ASCII the first bit is non-zero).
- There are also standards UTF-16, UTF-32 and UTF-64 - in these cases the first byte encodes the size (number of bytes) used to represent the character.

Fun Fact: binary data may stored in two orders - with most significant digit on the left, or on the right:



Source: https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html

UTF Codes

Run RUBY

```ruby
#source: https://www.charbase.com

checkmark = "\u2713"
puts checkmark.encode('utf-8')

delta = "\u0394"
puts delta.encode('utf-8')

letter = 110 # n
puts letter.chr()

puts 'L'.ord
```

Dynamic Arrays Revisited

Remember we looked at creating dynamic arrays (arrays that grow):

Run RUBY

```ruby
def main
    name_array = Array.new()

    puts("How many names: ")
    count = gets.to_i
    index = 0
    while (index < count)
        puts("Enter next name: ")
        name_array << gets.chomp
        index += 1 # Increment index by one
    end

    index = 0
    while (index < count)
```

Static Arrays (C)

In C we can create an array of **static, stack** elements as follows:

```c
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>

typedef struct Date {
    int day;
    int month;
    int year;
} Date;


bool check_if_finished() {
    char answer[256];
    bool result;
```

Dynamic Memory (C)

Or in this case we create dynamic, heap elements as follows:

```c
#include <stdio.h>
#include <stdbool.h>
#include <ctype.h>
#include <stdlib.h>

typedef struct Date {
    int day;
    int month;
    int year;
} Date;


bool check_if_finished() {
    char answer[256];
```

Dynamic Memory (Ruby)

Run RUBY

```ruby

class Date
    attr_accessor :day, :month, :year
end

def read_date()
    d = Date.new()
    puts "Enter day: "
    d.day = gets.chomp.to_i
    puts "Enter month: "
    d.month = gets.chomp.to_i
    puts "Enter year: "
    d.year = gets.chomp.to_i

```

Lists

Lists are another type of data structure. We look at lists as having a **head** and a **tail**.

eg: ["apple", "pear", "orange"]

the head is the element "apple, the tail is the list ["pair", "orange"]

In Ruby Arrays can be treated as lists.

Fun Fact

In some languages (Lisp, Scheme) a different terminology is used: **car** and **cdr**.

car = “Contents of the Address part of the Register”
cdr = “Contents of the Decrement part of the Register” (“could-er”)

From the 1950s implementation of LISP on an IBM 704.

What do we do with lists?

What sort of of things might you want to do to a list:

- [ ] Add something to the end of list
- [ ] Take something off the list
- [ ] Move through the list doing something for each item

List Operations

An array is a data structure - it simply is an organisation of memory. But a list generally comes with a set of additional operations that are useful for using data as a list.

A list is a way of viewing and using data.

List data structures typically allow the following operations:

1. items can added to them,
2. you can take the head (the first item)
3. you can ask if a list includes an item
4. you can apply an operation to each item in the list
5. You can insert items at a particular point

These are not things that languages (such as C) provide automatically with arrays.

Basic List Operations

Here are some of the simple operations you might do with lists:

```ruby

mylist = ["apple", "pear", "orange"]

puts "Original list: "
puts mylist

mylist += ["banana", "plum"]

puts "List with another list ['banana', 'plum'] added: "
puts mylist

mylist.insert(3, 'cherry')
puts "List with an item 'cherry' inserted after position 3: "
puts mylist
```

Complex List Operations - reject

Here are some more complex operations - the first creates a new list missing items if they match a condition:

Run RUBY

```ruby

list = [2, 4, 6, 8]
puts "Original list: "
puts list

newlist = list.reject do |i|
    if i > 2
        true
    else
        false
    end
end

puts("New list is: ")
```

Complex List Operations - each

This second one applies an operation to each item in the list if the item matches a condition - it:

1. Adds one to each item in the list and returns the changed items as a new list
2. Goes through the list printing out each element and its position

```ruby
list =[2, 4, 6, 8]

newlist = list.collect do |i|
    i + 1
end

j = 0
newlist.each do |i|
    j += 1
    puts "Item #{j} is #{i}"
end
```

Searching an Array

So we have seen some complex list operations performed on arrays.

One thing we often want to do is to see if an array contains an item.

To do this we need to search through the array and check each item to see if it matches the item we want.

If the item is there, we then want to be able to get it, so we might need its position.

We can do this as follows using a while loop:

Run RUBY

```ruby
# this returns the index of the item or -1 if it is not found
def search_array(a, search_item)
    index = 0
    found_index = -1
    while (index < a.length)
        if (a[index] == search_item)
            found_index = index
        end
        index += 1
    end
    return found_index
end

def main
```

Change the code above so that the item searched for is not in the list. Does the function search_array() work as expected?

Search question

Instead of returning the index of the item we could just return the item itself?

- True
- False

Designing and building the Text Music player

Lets us look at:

1. Bottom up design for the Text Music player task
2. The structure chart for that task
3. Writing the code and testing it for that task.

Designing the Music Player video.

Music Player Structure Chart

7.1T_Structure (2).pptx

Tests

We look at the two tests:

Test 1

This will be run online and offered once in Week 8. You need to submit whatever you have done in Ed after 90 minutes, but you get to redo the test if it is not finished, or close to finished. The times for the test will be in the workshop time.

Test 2

This will also be offered online in Week 10 in the workshop time.