CSCI 410
22 – Operating System

Jack Revisited

```java
/** Compute the average of a sequence of integers */
class Main {
    function void main() {
        var Array a;
        var int length;
        var int i, sum;
        let length = Keyboard.readValue("How many numbers? ");
        let a = Array.new(length); // Constructs the array
        let i = 0;
        while (i < length) {
            let a[i] = Keyboard.readValue("Enter the next number: ");
            let sum = sum + a[i];
            let i = i + 1;
        }
        do Output.printString("The average is: ");
        do Output.printInt(sum / length);
        return;
    }
}
```

Typical OS Functions

- Mathematical operations (abs, sqrt, ...)
- Abstract data types (String, Date, ...)
- Output functions (printChar, printString...)
- Input functions (readChar, readLine...)
- Graphics functions (drawPixel, drawCircle, ...)
- And more...

The Jack OS

- Math: Provides basic mathematical operations;
- String: Implements the String type and string-related operations;
- Array: Implements the Array type and array-related operations;
- Output: Handles text output to the screen;
- Screen: Handles graphic output to the screen;
- Keyboard: Handles user input from the keyboard;
- Memory: Handles memory operations;
- Sys: Provides some execution-related services.

Jack OS API

```
class Main {
    class String {
        class Array {
            class Output {
                class Screen {
                    class Memory {
                        class Sys {
                            function void halt():
                            function void error(int errorCode)
                            function void wait(int duration)
                            ```
A Modern OS

- Is modular and scalable
- Empowers programmers (language extensions)
- Empowers users (file system, GUI, ...)
- Closes gaps between software and hardware
- Runs in "protected mode"
- Typically written in some high level language
- Typically grows gradually, assuming more and more functions
- Must be efficient.

### Efficiency

We have to implement various operations on \( n \)-bit binary numbers (\( n = 16, 32, 64, ... \)).

#### Example I: Multiplication

**The algorithm explained**

```plaintext
\( \text{multiply}(y, x) = \{ \text{for } i = 0 \ldots y - 1 \text{ do } \text{sum} = \text{sum} + x \} \)
```

- Run-time: proportional to \( y \)
- Can be implemented in SW or HW
- Division: similar idea.

#### Example II: Square Root

The square root function has two convenient properties:
- Its inverse function is computed easily
- It's monotonically increasing

Functions that have these two properties can be computed by binary search:

```java
\text{sqrt}(x):\

// Compute the integer part of \( y = \sqrt{x} \). Strategy:
// Find an integer \( y \) such that \( x^2 \leq n < (y+1)^2 \) (for \( 0 \leq x < 2^n \))
// By performing a binary search in the range \( 0 \ldots 2^{n/2} - 1 \)
// \( y = 0 \)
// for \( j = n/2 - 1 \ldots 0 \) do
// if \( (x + 2^j)^2 \leq x \) then \( y = y + 2^j \)
// return \( y \)
```

Number of loop iterations is bounded by \( n/2 \), thus the run-time is \( O(n) \).

### Math Functions

```java
class Math {
    function void init()
    function int abs(int x)

    function int multiply(int x, int y)
    function int divide(int x, int y)
    function int max(int x, int y)
    function int min(int x, int y)
    function int sqrt(int x)
}
```

The remaining functions are simple to implement.

### String Processing

```java
class String {
    constructor String new(int maxLength)
    method void dispose()
    method int length()
    method char charAt(int j)
    method String appendChar(char c)
    method void eraseLastChar()
    method int intValue()
    method void setInt(int j)
    function char backSpace()
    function char doubleQuote()
    function char newLine()
}
```
Single Digit ASCII Conversions

Character | ASCII code
---|---
0 | 48
1 | 49
2 | 50
3 | 51
4 | 52
5 | 53
6 | 54
7 | 55
8 | 56
9 | 57

- asciiCode(digit) == digit + 48
- digit(asciiCode) == asciiCode - 48

Converting Number to String

- SingleDigit-to-character conversions: done
- Number-to-string conversions:

```java
// Convert a non-negative number to a string
int String(int n) {
    intLastDigit = n % 10
    c = character representing intLastDigit
    n = n / 10
    return c (as a string)
    else
        return int2String(n / 10) + c
}
```

Memory Management

```java
class Memory {
    function int peek(int address)
    function void poke(int address, int value)
    function Array alloc(int size)
    function void deAlloc(Array o)
}
```

- The data structure that this algorithm manages is a single pointer, `free`.

Naïve Memory Management

- When a program constructs (deletes) an object, the OS has to allocate (deallocate) a RAM block on the heap:
  - `alloc(size)`: returns a reference to a free RAM block of size `size`
  - `deAlloc(object)`: recycles the RAM block that `object` refers to

```
Initialization: free = heapBase

// Allocate a memory block of size words:
alloc(size)
    pointer = free
    free = free + size
    return pointer

// Deallocate the memory space of a given object:
deAlloc(object)
    do nothing
```

Improved Memory Management

```java
class Memory {
    int FreeList = heapBase
    int FreeListLength = heapLength
    int FreeListnext = null

    void dealloc(Area obj) {
        FreeListLength = FreeListLength - 1
        FreeListnext = FreeListnext - obj
    }

    function int peek(int address)
    function void poke(int address, int value)
    function Array alloc(int size)
    function void deAlloc(Array o)
}
```

- Implementation: based on our ability to exploit exotic casting in Jack:

```
// To create a data-level "proxy" of the RAM using Array memory:
let memory = 0;
// From this point on we can use code like:
let x = memory[2] // Where j is any RAM address
let memory[j] = y // Where j is any RAM address
```
**Graphics Primitives**

```java
class Screen {
    function void clearScreen()
    function void setColor(boolean b)
    function void drawPixel(int x, int y)
    function void drawLine(int x1, int y1, int x2, int y2)
    function void drawRectangle(int x1, int y1, int x2, int y2)
    function void drawCircle(int x, int y, int r)
}
```

**Memory-Mapped Screen**

Memory-Mapped Screen

- **Pixel Drawing**
  - **drawPixel(x, y):**
    - // Hardware-specific.
    - If accessing a memory-mapped screen.
    - Write a predetermined value in the RAM location corresponding to screen location (x, y).
  - Implementation: using `poke(address, value)`

**How to Draw a Line**

- **Basic idea:** `drawLine` is implemented through a sequence of `drawPixel` operations.
- **Challenge 1:** Which pixels should be drawn?
- **Challenge 2:** How to draw the line fast?
- **Simplifying assumption:** the line that we are asked to draw goes north-east.

**Line Drawing**

- **Given:** `drawLine(x1, y1, x2, y2)`
- **Notation:** `x=x1, y=y1, dx=x2-x1, dy=y2-y1`
- **Using the new notation:** We are asked to draw a line between `(x, y)` and `(x+dx, y+dy)`.

**Line Drawing Algorithm**

- Set `(a, b) = (0, 0)`
- While there is more work to do:
  - `drawPixel(x+a, y+b)`
  - decide if you want to go right, or up
    - if you decide to go right, set `a++`;
    - if you decide to go up, set `b++`.
  - `set (a, b) = (0, 0)`
  - While `(a ≤ dx)` and `(b ≤ dy)`:
    - `drawPixel(x+a, y+b)`
    - decide if you want to go right, or up
      - if you decide to go right, set `a++`;
      - if you decide to go up, set `b++`.

**Line Drawing**

- Given: `drawLine(x1, y1, x2, y2)`
- Notation: `x=x1, y=y1, dx=x2-x1, dy=y2-y1`
- Using the new notation:
  - We are asked to draw a line between `(x, y)` and `(x+dx, y+dy)`.

**Line Drawing Algorithm**

- `set (a, b) = (0, 0)`
- While there is more work to do:
  - `drawPixel(x+a, y+b)`
  - decide if you want to go right, or up
    - if you decide to go right, set `a++`;
    - if you decide to go up, set `b++`.
  - `set (a, b) = (0, 0)`
  - While `(a ≤ dx)` and `(b ≤ dy)`:
    - `drawPixel(x+a, y+b)`
    - decide if you want to go right, or up
      - if you decide to go right, set `a++`;
      - if you decide to go up, set `b++`.
**Line Drawing Algorithm**

Motivation:
- When you draw polygons, e.g., in animation or video, you need to draw millions of lines.
- Therefore, drawing must be ultra fast.
- Division is a very slow operation.
- Addition is ultra fast (hardware based).

- \( \frac{a}{dx} < \frac{b}{dy} \) is the same as \( a*dy < b*dx \)
- Define \( \text{diff} = a*dy - b*dx \)
- Let's take a close look at this \( \text{diff} \):
  1. \( \frac{a}{dx} < \frac{b}{dy} \) is the same as \( \text{diff} < 0 \)
  2. When we set \((a,b) = (0,0)\), \( \text{diff} = 0 \)
  3. When we set \(a = a+1\), \( \text{diff} \) goes up by \( dy \)
  4. When we set \(b = b+1\), \( \text{diff} \) goes down by \( dx \)

```plaintext
drawLine(x,y,x+dx,y+dy)
set (a,b) = (0,0)
while (a ≤ dx) and (b ≤ dy)
drawPixel(x+a,y+b)
if \( \frac{a}{dx} < \frac{b}{dy} \)
  set a = a+1
else
  set b = b+1
```

**Circle Drawing**

The screen origin \((0,0)\) is at the top left.

```plaintext
drawCircle(x,y,r)
for each \( \phi \in \{...,-1\} \)
drawLine((x - r cos(\phi), y + r sin(\phi)) to \((x + r cos(\phi), y + r sin(\phi))\)
```

**Character Output Primitives**

```plaintext
class Output {
  function void moveCursor(int i, int j)
  function void printChar(char c)
  function void printString(String s)
  function void printInt(int i)
  function void println()
  function void backSpace()
}
```

**Character Output**

- Given display: a physical screen, say 256 rows by 512 columns
- We can allocate an 11 by 8 grid for each character
- Hence, our output package should manage a 23 lines by 64 characters screen
- Font: each displayable character must have an agreed-upon bitmap
- In addition, we have to manage a "cursor".

```plaintext
class Output {
  static Array CharMaps;
  function void initMap()
  {
    let CharMaps = Array.new(127);
    // Assign a bitmap for each character
    do Output.create(32,0,0,0,0,0,0,0,0,0,0,0);          // space
    do Output.create(33,12,30,30,30,12,12,0,12,12,0,0);  // !
    do Output.create(34,54,54,20,0,0,0,0,0,0,0,0);       // "
    do Output.create(35,0,18,18,63,18,18,63,18,18,0,0);  // #
    ... do Output.create(65,0,0,0,0,0,0,0,0,0,0,0);          // A ** TO BE FILLED **
    do Output.create(66,31,51,51,51,31,51,51,51,31,0,0); // B
    do Output.create(67,28,54,35,3,3,3,35,54,28,0,0);    // C
    ... return;
  }
}
```

**Font Implementation**

```plaintext
class Output {
  static CharMaps;
  function void initMap()
  {
    let CharMaps = Array.new(127);
    // Assign a bitmap for each character
    do Output.create(32,0,0,0,0,0,0,0,0,0,0,0);          // space
    do Output.create(33,12,30,30,30,12,12,0,12,12,0,0);  // !
    do Output.create(34,54,54,20,0,0,0,0,0,0,0,0);       // "
    do Output.create(35,0,18,18,63,18,18,63,18,18,0,0);  // #
    ... do Output.create(65,0,0,0,0,0,0,0,0,0,0,0);          // A ** TO BE FILLED **
    do Output.create(66,31,51,51,51,31,51,51,51,31,0,0); // B
    do Output.create(67,28,54,35,3,3,3,35,54,28,0,0);    // C
    ... return;
  }
}
```

**Keyboard Primitives**

```plaintext
class Keyboard {
  function char keyPressed()
  function char readChar()
  function String readLine(String message)
  function int readInt(String message)
}
```
Keyboard Input

```c
keyPressed()
{
    // Depends on the specifics of the keyboard interface
does this key is presently pressed or on the keyboard
    returns the ASCII value of the key.
    else
    return 0;
}
```

- If the RAM address of the keyboard’s memory map is known, the above logic can be implemented using a peek function.
- Problem I: the elapsed time between a “key press” and key release” events is unpredictable.
- Problem II: when pressing a key, the user should get some visible feedback (cursor, echo, ...).

ReadLine()

```c
readLine()
{
    // Read and echo a single character
    display the name;
    while no key is pressed on the keyboard
        do nothing; // wait till the user presses a key
    e = code of currently pressed key
    while a key is pressed
    do nothing; // wait for the user to let go
    print e at the current cursor location
    move the cursor one position to the right.
    return e;
}
```

Perspective

- What we presented can be described as:
  - A mini OS
  - A standard library
  - Many classical OS functions are missing
  - No separation between user mode and OS mode
  - Some algorithms (e.g., multiplication and division) are standard
  - Other algorithms (e.g., line- and circle-drawing) can be accelerated with special hardware
  - And, by the way, we’ve just finished building the computer.