

**COM 1101 Fundamentals of Computer Science**  
**Midterm Exam: February 10, 1998**  
**Rasala Sections**

Name \_\_\_\_\_ ID Number \_\_\_\_\_

1: 20 Pts	2: 20 Pts	3: 25 Pts	4: 15 Pts	5: 10 Pts	6: 10 Pts	Total

**Problem 1: Parameter Passage (20 Points: 4/4/4/4/4)**

Assume the declarations:

<pre>struct Junk {     int x;     int y; };</pre>	<pre>void Foo1(Junk J) {     J.x += 1;     J.y += 1; }</pre>
<pre>void Foo2(Junk&amp; J) {     J.x += 2;     J.y += 2; }</pre>	<pre>void Foo3(const Junk&amp; J) {     J.x += 3;     J.y += 3; }</pre>
<pre>void Foo4(Junk* P) {     P-&gt;x += 4;     P-&gt;y += 4; }</pre>	<pre>void Foo5(const Junk* P) {     P-&gt;x += 5;     P-&gt;y += 5; }</pre>

(A) Explain the concept of *passage by value*.

In passage by value, a copy is made of the data passed and the function parameter name refers to this copy rather than to the original data. This has two results. Any changes to the function parameter will not change the original data. In addition, it is permissible to pass an expression by value since the expression will be evaluated and stored in the new location associated with the parameter name.

Which of the Foo functions use passage by value?: Foo 1

(B) Explain the concept of *passage by reference*.

In passage by reference, the function parameter name is made into an alias (alternate name) for the original data. Thus no copy is done. Instead, all uses of the function parameter name refer to the original data location. In particular, all changes to the function parameter automatically change the original data since there is only one location in memory that is being referenced.

Passage by reference uses the mechanism of a hidden pointer to accomplish the alias. By using a hidden pointer, the compiler assumes the responsibility for the pointer operations & and \* as needed. Programmer code becomes less cluttered.

Passage by reference is fast for large data structures. To achieve the speed of passage by reference but prevent change to the original, passage by const reference is used.

If `T` is a typename, then passage by reference is `T&` and passage by const reference is `const T&`.

Which of the `Foo` functions use passage by reference?: `Foo2` and `Foo3`

(C) Explain the concept of *passage by pointer*.

In passage by pointer, a pointer to some data type is passed rather than passing the data type itself directly (either by value or by reference).

To pass an ordinary variable `v` as an argument which expects a pointer, you must pass the address of `v`, that is, `&v`.

If `P` is the name of a pointer argument, then to access the associated data you must either use `*P` or `P->fieldname`.

In other words, passage by pointer and usage of the pointer require manual coding of the `&`, `*`, and `->` operators.

There are four common forms of passage by pointer (where `T` is a typename):

(1) `T* P`

In this case the pointer value is copied and the data may be accessed and modified using either `*P` or `P->fieldname`.

(2) `const T* P`

In this case the pointer value is copied and the data may be accessed but *not* modified using either `*P` or `P->fieldname`.

(3) `T*& P`

*In this case the pointer is passed by reference so that changes to the pointer itself will be reflected in the original pointer. The data may be accessed and modified using either `*P` or `P->fieldname`.*

(4) `const T*& P`

*In this case the pointer is passed by reference so that changes to the pointer itself will be reflected in the original pointer. The data may be accessed but *not* modified using either `*P` or `P->fieldname`.*

Which of the `Foo` functions use passage by pointer?: `Foo4` and `Foo5`

<pre>struct Junk {     int x;     int y; };</pre>	<pre>void Foo1(Junk J) {     J.x += 1;     J.y += 1; }</pre>
<pre>void Foo2(Junk&amp; J) {     J.x += 2;     J.y += 2; }</pre>	<pre>void Foo3(const Junk&amp; J) {     J.x += 3;     J.y += 3; }</pre>
<pre>void Foo4(Junk* P) {     P-&gt;x += 4;     P-&gt;y += 4; }</pre>	<pre>void Foo5(const Junk* P) {     P-&gt;x += 5;     P-&gt;y += 5; }</pre>

(D) Some of the above functions produce an error at compile time. Which ones?

Foo3 and Foo5

Explain why these function definitions produce an error.

Foo3 changes the data fields in the `const` parameter J.

Foo5 changes the data fields of the object pointed to by the `const` pointer P.

(E) For the code sequence below, show the printed output for each `cout` statement and explain the output.

Statements	Output	Output Explanation
<pre>Junk S; Junk* Q = &amp;S;</pre>		Data just assigned so output is same as data
<pre>S.x = 1; S.y = 3; cout &lt;&lt; S.x &lt;&lt; " " &lt;&lt; S.y &lt;&lt; "\n";</pre>	1 3	
<pre>Foo1(S); cout &lt;&lt; S.x &lt;&lt; " " &lt;&lt; S.y &lt;&lt; "\n";</pre>	1 3	Foo1 operates on copy of data so original is unchanged
<pre>Foo2(S); cout &lt;&lt; S.x &lt;&lt; " " &lt;&lt; S.y &lt;&lt; "\n";</pre>	3 5	Foo2 operates on same location as original so data is changed
<pre>Foo4(Q); cout &lt;&lt; S.x &lt;&lt; " " &lt;&lt; S.y &lt;&lt; "\n";</pre>	7 9	Foo4 operates on same location as original via -> operator so data is changed

## Problem 2: Arrays and Pointers (20 Points: 3/3/2/5/5/2)

Assume the declarations:

```
long A[100];
long* B;
long* P;
long size;
```

(A) Write the code to fill A with the sequence 1, 3, 5, 7, 9, ... .

```
for (long i = 0; i < 100; i++)
    A[i] = 2 * i + 1;
```

(B) Write the code to make P point to a dynamically allocated long and set the value of that long to 22.

```
P = new long(22);           // notice initialization uses ( )
// or
P = new long;              // allocate but do not initialize
*P = 22;                   // initialize afterwards
```

(C) Write the code to deallocate the long to which P points.

```
delete P;                  // delete single item
```

(D) Write the code to:

- (1) read size from the user
- (2) make B point to a dynamically allocated array of long with size elements.

```
size = RequestLong("Enter array size: ");
B = new long[size];       // notice array allocation uses [ ]
```

(E) Using pointer traversal (with P as the pointer), write the code to traverse the array to which B is pointing and fill that array with the sequence 1, 4, 7, 10, 13, ... .

```
P = B;                    // make P point to cell 0 of B
*P = 1;                   // initialize cell 0 to value 1
++P;                      // make P point to cell 1 of B

while (P < (B + size)) {
    *P = *(P - 1) + 3;     // increase previous cell by 3
    ++P;                  // make P point to next cell
}
```

(F) Write the code to deallocate the array to which B points.

```
delete [ ] B;             // delete whole array
```

### Problem 3: Clocks (25 Points: 8/4/3/3/4)

Assume the following class declaration to which you will add functions.

```
class ClockTime {
    long hours;           // assume 0 <= hours < 24
    long minutes;        // assume 0 <= minutes < 60
    long seconds;        // assume 0 <= seconds < 60

public:
    // member functions follow ...
}
```

(A) Write a member function `Set` of class `ClockTime` with the header:

```
ClockTime& Set(long Hours, long Minutes, long Seconds);
```

This function must examine its parameters carefully and modify them in an appropriate manner so that the stored values of `hours`, `minutes`, and `seconds` satisfy the standard conditions given above. In addition, excess seconds should carry into the minutes and excess minutes into the hours with appropriate wraparound. The function should return a reference to the `ClockTime` object itself.

For example, if `C` is a `ClockTime` object, then:

```
C. Set(7, 5, 13);           produces 7 hours, 5 minutes, 13 seconds
C. Set(17, 59, 195);       produces 18 hours, 2 minutes, 15 seconds
C. Set(0, 0, -1);          produces 23 hours, 59 minutes, 59 seconds
```

Direct solution:

```
{
    // First fix parameters to wrap values into standard range

    Minutes += Seconds / 60;           // add excess Seconds to Minutes
    Seconds %= 60;                     // reduce Seconds modulo 60

    if (Seconds < 0) {                 // handle negative Seconds
        Minutes--;                     // get one minute back
        Seconds += 60;                 // wrap Seconds to positive
    }

    Hours += Minutes / 60;             // add excess Minutes to Hours
    Minutes %= 60;                     // reduce Minutes modulo 60

    if (Minutes < 0) {                 // handle negative Minutes
        Hours--;                       // get one hour back
        Minutes += 60;                 // wrap Minutes to positive
    }

    Hours %= 24;                       // reduce Hours modulo 24

    if (Hours < 0) {                   // handle negative Hours
        Hours += 24;                   // wrap Hours to positive
    }
}
```

```

// now assign parameters to member variables

seconds = Seconds;
mi nutes = Mi nutes;
hours    = Hours;

// finally return the current object

return *this;
}

```

Alternate solution uses an auxiliary function:

```

// CarryReduce forces data into range 0 <= data < modulus
// Return carry from this reduction as its function value
// Notice that data is a reference parameter
// CarryReduce assumes modulus > 0

long CarryReduce(long& data, long modulus) {
    long carry;

    carry = data / modulus;    // compute main carry
    data %= modulus;          // wrap data into range

    if (data < 0) {           // handle negative data
        carry--;              // get one carry back
        data += modulus;      // wrap data to positive
    }

    return carry;
}

```

Now give solution using auxiliary function:

```

{
    // First fix parameters to wrap values into standard range

    Mi nutes += CarryReduce(Seconds, 60);
    Hours    += CarryReduce(Mi nutes, 60);
    CarryReduce(Hours, 24);

    // now assign parameters to member variables

    seconds = Seconds;
    mi nutes = Mi nutes;
    hours    = Hours;

    // finally return the current object

    return *this;
}

```

(B) Write a constructor for class `Cl ockTi me` which takes the same parameters as in the `Set` function. The constructor should call `Set` in its body to make the required adjustments to bring the data values into the standard range.

```
Cl ockTi me(l ong Hours, l ong Mi nutes, l ong Seconds) {  
    Set(Hours, Mi nutes, Seconds);  
}
```

(C) Write a member function `Add` of class `Cl ockTi me` with the header:

```
Cl ockTi me& Add(l ong Hours, l ong Mi nutes, l ong Seconds);
```

This function should add the passed parameters to the internal member data and do what is necessary to put the time back into the standard range. The function should return a reference to the `Cl ockTi me` object itself.

```
{  
    seconds += Seconds;  
    mi nutes += Mi nutes;  
    hours   += Hours;  
  
    return Set(hours, mi nutes, seconds);  
}
```

(D) Write a member function `Add` of class `Cl ockTi me` with the header:

```
Cl ockTi me& Add(const Cl ockTi me& X);
```

This function should add the time components in `x` to the member data and do what is necessary to put the time back into the standard range. The function should return a reference to the `Cl ockTi me` object itself.

```
{  
    seconds += X. Seconds;  
    mi nutes += X. Mi nutes;  
    hours   += X. Hours;  
  
    return Set(hours, mi nutes, seconds);  
}
```

(E) Define a member function `TotalSeconds()` that will convert the time in hours, minutes, and seconds into the total number of seconds and return that result. The header should be:

```
long TotalSeconds() const
{
    return (3600 * hours + 60 * minutes + seconds);
}
```

(F) Write a member function `Print()` that will print a `ClockTime` to `cout`. For simplicity, use “24 hour format” which is illustrated below.

Print 7 hours, 5 minutes, 3 seconds as:     07: 05: 03

Print 17 hours, 25 minutes, 33 seconds as:     17: 25: 33

Notice that you must add leading 0's when the hours, minutes, or seconds are less than 10. You may do this by direct code or use standard io manipulators. If you use standard io manipulators, reset them to their default state before you return.

```
void Print() const {
    if (hours < 10) cout << '0';
    cout << hours << ':';

    if (minutes < 10) cout << '0';
    cout << minutes << ':';

    if (seconds < 10) cout << '0';
    cout << seconds;
}
```

// or

```
void Print() const {
    cout
        << setfill('0')
        << setw(2) << hours << ':'
        << setw(2) << minutes << ':'
        << setw(2) << seconds
        << setfill(' ');
}
```

#### Problem 4: Blocks (15 Points: 3/5/5/2)

Assume the following class declaration to which you will add functions.

```
class Block {  
  
    short xsize;    // size of block in x direction  
    short ysize;    // size of block in y direction  
  
    short x;        // x location of topleft corner of block  
    short y;        // y location of topleft corner of block  
  
public:  
    // member functions follow ...  
};
```

(A) Define the following member function `Set` of `Block` to change the `x` and `y` data fields and return a reference to the object.

```
Block& Set(int X, int Y)    // change the fields x and y  
  
{  
    x = X;  
    y = Y;  
  
    return *this;  
}
```

(B) Define the following member function `Draw` of `Block` to draw the block as a painted rectangle in the current foreground color. Assume that `x, y` defines the top left corner of the block and `xsize, ysize` its width and height. Return a reference to the object.

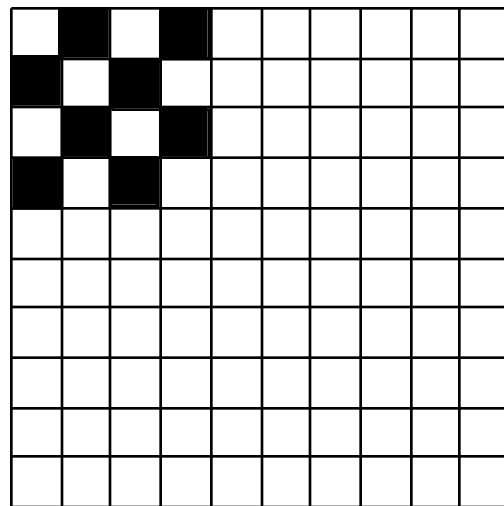
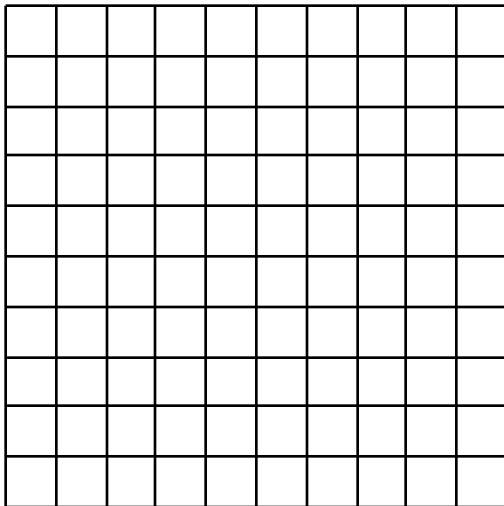
```
Block& Draw() const        // draw the block  
  
{  
    PaintRect(x, y, x + xsize, y + ysize);  
  
    return *this;  
}
```

(C) Assume that `D` is a `Block` for which `xsize` and `ysize` are both 20. Then, in the diagram below, draw the picture obtained by executing the following code:

```
for (int i = 0; i <= 3; i++) {
    for (int k = 0, k <= 3, k++) {
        if (((i + k) % 2) == 1) {
            int X = 20 * i;
            int Y = 20 * k;

            D.Set(X, Y).Draw();
        }
    }
}
```

Assume that the grid in the picture is 20 pixels by 20 pixels.



(D) Explain why the definitions of `Set` and `Draw` permit the code

```
D.Set(X, Y).Draw();
```

to work, that is, what is the critical feature that allows this style?

Because of the code `return *this;` in `Set`, the call `D.Set(X, Y)` returns `D` itself so that the call `D.Draw()` can then be made. It is important to note that the sequence would fail if `Set` did not return a reference to its calling object.

## Problem 5: The Array Class (10 Points: 2/8)

In lectures, we introduced a dynamic array class which begins:

```
template <class T> class Array {
    T*   arraydata;    // pointer to the allocated array
    long arraysize;   // size of the allocated array

    // member functions follow ...
}
```

(A) Write the destructor for this class.

```
~Array() { delete [] arraydata; };
```

(B) Write the member function `resize` that will resize the array to a new size, copy the old data values whenever possible, and fill with a constant value those array cells that are not filled by copying old data values. The header should be:

```
Array& resize(long Size, const T& Fill = T());
```

You may assume the existence of auxiliary templates `ArrayCopy` and `ArrayFill` as in lectures.

```
Array& resize(long Size, const T& Fill = T()) {
    // error check Size
    if (Size < 1) Size = 1;

    // check if resize is needed
    if (arraysize == Size)
        return* this;           // return this object

    // save old data and size
    T*   olddata = arraydata;
    long oldsize = arraysize;

    // get new space
    arraydata = new T[Size];
    arraysize = Size;

    // copy old information and fill if needed
    if (oldsize < arraysize) {
        ArrayCopy(arraydata, olddata, 0, oldsize);
        ArrayFill(arraydata, oldsize, arraysize, Fill);
    }
    else {
        ArrayCopy(arraydata, olddata, 0, arraysize);
    }

    // give back old space
    delete [] olddata;

    return *this;           // return this object
};
```

### Problem 6: Sorting (10 Points: 1/4/5)

Assume the declarations:

```
int A[10];
int i, j, k, temp;
```

Assume that A currently has the following data values:

0	1	2	3	4	5	6	7	8	9
7	13	21	26	41	18	22	12	54	1

Note that the data in cells 0 through 4 is in sorted order but the remaining data is not.

(A) Assume *i* has the value 5. Consider the code:

```
j = i;
while (j > 0 && A[j - 1] > A[j])
    j--;
```

What is the value of *j* after this code is executed? 2

(B) Assume the values of *i* and *j* at the end of part (A). Consider the code:

```
temp = A[i];
for (k = j; k < i; k++)
    A[k+1] = A[k];
A[j] = temp;
```

Show the contents of array A after this code is executed:

0	1	2	3	4	5	6	7	8	9
7	13	18	21	21	21	22	12	54	1

(C) The code in part B does not properly sort the elements in A between 0 and 5. Write a better piece of loop code that will sort these elements without destroying any data values.

The loop fails because the copy  $A[k+1] = A[k]$  goes upwards and the loop index *k* goes upwards also. This is why the value 21 overwrites 3 cells. To fix this, loop downwards instead but keep the same form of copy operation.

```
for (k = i - 1; k >= j; k--)
    A[k+1] = A[k];
```