Print:

First Name:............................Last Name:............................

Login ID:...

Student Number:...

University of Toronto
Faculty of Applied Science and Engineering

Final Examination – December 1996

APS105F – Computer Fundamentals

Examiner – Paul Chow

1. There are 6 questions and **22** pages. Do **all** questions. The total number of marks is 100.
2. **ALL WORK IS TO BE DONE ON THESE SHEETS!** Use the back of the pages if you need more space. Be sure to indicate **clearly** if your work continues elsewhere.
3. Please put your final solution in the box if one is provided.
4. **No calculators or other computing devices allowed.**
5. **Paper Type A** – No Aids permitted. Appendix B describes the basic OOT syntax you will require and Appendix C describes the basic C syntax you will require.
6. Place your student card on your desk.

1 [10]	
2 [20]	
3 [15]	
4 [15]	
5 [10]	
6 [30]	
Total [100]	

[2 marks] 1. (a) What is the purpose of the program counter inside the CPU?

[3 marks] (b) What are the three basic steps of CPU operation carried out for each instruction? Give a short description of what happens in each step.

[2 marks] (c) Why does a fast CPU need to have caches?

[3 marks] (d) You have been made chief designer of the *Intel Pentium Super Pro Plus Plus* processor. Give a very general overview of the kinds of experiments or studies that you would do to figure out the size of caches that you require.

2. (a) Consider the following C program.

```
#include <stdio.h>

int mystery(int, int, int);

main()
{
  int a, b, c;
  do
    {
      printf("Enter three integers: ");
      scanf("%d%d%d", &a, &b, &c);
      printf("The result is: %d\n",mystery(a,b,c));
    } while (a > 0);
}

int mystery(int x, int y, int z)
{
  int this = x;

  if (y > this)
    this = y;
  if (z > this)
    this = z;
  return this;
}
```

[2 marks] What does the program do?

Continued...

[9 marks] Write an equivalent OOT program.

[9 marks] (b) Write an equivalent C program for the following OOT program:

```
procedure Order (var X, Y : real)
    var temp : real
    if X > Y then
        temp := X
        X := Y
        Y := temp
    end if
end Order

var a, b : real

for i : 1 .. 10
    put "Enter a pair of numbers: "..
    get a, b
    Order(a,b)
    put "In order, the numbers are: ",a," ",b
end for
```


[2 marks] 3. (a) Define a pointer.

[4 marks] (b) Consider the following code fragment. Recall that `&varname` means `addr(varname)` and `*varname` means `deref(varname)`.

```
int ivar, jvar;
int *iptr;

iptr = &ivar;
ivar = 5;
jvar = 20;

/* A */

*iptr = jvar + ivar;

/* B */
```

Assume that the variables are stored in memory at the following addresses:

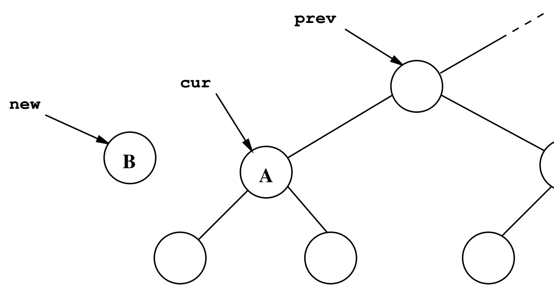
Variable	Address
`ivar`	1000
jvar	1230
iptr	2030

What are the values of the variables at points `A` and `B`?

	`ivar`	`jvar`	`iptr`
A			
B			

Question 3 continued...

(c) Consider the part of a binary tree shown below:



[2 marks] i. Write an OOT declaration for a node in the tree. Assume the node needs to store a string variable.

[3 marks] ii. Assume that `new`, `cur`, and `prev` are pointers to the nodes shown. Write OOT code to delete node **A** and replace it with node **B**.

Question 3 continued...

[4 marks] (d) Draw the sorted binary tree that results when the letters `c`,`o`,`m`,`p`,`u`,`t`,`i`,`n` and `g` are inserted in the order shown.

[15 marks] 4. You have been asked to lead the development of a new software product. For a status report that you are writing, you need to include a few paragraphs describing the programming language that you have chosen to use and the reasons for selecting it. Write those paragraphs here, and include in your discussion at least two arguments for and two against the use of strongly-typed languages such as Turing versus C. Finally give your choice and the reasons.

Marks will be given for both the technical content and the style of presentation.

5. Consider the following program:

```
#include <stdio.h>
#define SIZE 5

void someFunction(int b[ ],int size);

main()
{
  int a[SIZE] = {32,27,64,18,95};

  printf("Here is the output:\n");
  someFunction(a,SIZE);
  printf("\n");
  return 0;
}

void someFunction(int b[ ], int size)
{
  /* Enter the values of b[0] and size at this point */
  if (size > 0)
    {
      someFunction(&b[1], size-1);
      printf("%d ",b[0]);
    }
}
```

Continued on next page...

Question 5 continued...

[6 marks] (a) Trace the execution of the program by filling in the values of `b[0]` and `size` just after the entry to each call of `someFunction`. Note that the table may be larger than necessary.

Call No.	`b[0]`	`size`
1		
2		
3		
4		
5		
6		
7		
8		

[2 marks] (b) What does this program do?

[2 marks] (c) Show the final output of the program.

6. In this question you will develop part of a program that implements a part of a language called *Logo*. *Logo* was famous for the concept called *turtle graphics*. Imagine a mechanical turtle that walks around the room under the control of a *Turing* program. The turtle holds a pen in either the *up* position or the *down* position. While the pen is down, the turtle draws on the surface as it moves; while the pen is up, the turtle moves without writing anything. The program being developed will simulate the operation of the turtle and create a computerized drawing.

Assume the turtle moves about an array called `World` that is initialized to be all zeroes. The size of the world is determined by two constants called `EndOfWorldX` and `EndOfWorldY`. Commands are read from the terminal as a series of integer numbers. The program keeps track of the position of the turtle using the variables `CurX` and `CurY`. Whenever the turtle hits the edge of the world, it will stop at the edge. (Otherwise, we will lose too many turtles due to bad driving!)

There are four variables (`N, S, E, W`) that determine the direction of travel, corresponding to north (up), south (down) , east (right), and west (left).

The turtle is initially placed approximately at the middle of the `World` with the pen up. The variable `PenIsDown` is used to keep track of the position of the pen. When the turtle moves with the pen down, the appropriate elements of the `World` array are set to 1.

When the print command is given, the array is printed on the terminal. Wherever a 1 is encountered in the array, print an '`x`', otherwise print a blank.

The set of commands to be processed by the turtle are:

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5 N	Move forward, the next number read (N) indicates how many spaces to move
6	Print the array
9	End the program

Continued...

In the following example, the turtle is assumed to be near the middle of the world. The commands will cause it to draw a 6 by 6 square and print the output.

```
2
5 6
3
5 6
3
5 6
3
5 6
1
6
9
```

The skeleton of the program you are to complete is shown in Appendix A on page 17. Study this code to see how it works.

(a) You will write the appropriate Turing code for the functions below: You only need to write the code that goes inside the procedure declarations, which are shown in Appendix A.

[2 marks] `PenUp`

[2 marks] `PenDown`

Continued...

[4 marks] `RightTurn`

[4 marks] `PrintArray`

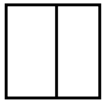
Question 6 continued…

[12 marks] `Forward(Distance)` You only need to write the code for the directions `N` and `W`, but indicate with comments where the code for `E` and `S` should be.

Continued…

Question 6 continued…

[6 marks] (b) Starting with the example program but omitting the *End* command (9), add the code needed to draw a vertical line through the middle of the box that ends at the top and bottom edges of the box. Print the result and leave the pen up. Please put comments beside your program!

The resulting figure would look something like this: 

A Turtle Code

```
var PenIsDown : boolean  % true means pen is down, false means up
var CurX : int           % X,Y coordinate of the turtle
var CurY : int
const EndOfWorldX := 20
const EndOfWorldY := 20
var World : array 1 .. EndOfWorldX, 1 .. EndOfWorldY of int
var N,S,E,W : boolean    % direction variables.  Only one can be true.

procedure PenUp
    % Your code here
end PenUp

procedure PenDown
    % Your code here
end PenDown

procedure RightTurn
    % Your code here
end RightTurn

procedure LeftTurn
    % Has been done, not shown here.
end LeftTurn

procedure Forward(Distance : int)
    % Your code here
end Forward

procedure PrintArray
    % Your code here
end PrintArray

function ParseCommands : int
    var Command : int
    var Distance : int

    put "Instruction? "..
    get Command

    if Command = 1 then
        PenUp
        result 1
    elsif Command = 2 then
        PenDown
        result 1
```



```
    elsif Command = 3 then
        RightTurn
        result 1
    elsif Command = 4 then
        LeftTurn
        result 1
    elsif Command = 5 then
        get Distance
        Forward(Distance)
        result 1
    elsif Command = 6 then
        PrintArray
        result 1
    elsif Command = 9 then
        result 0
    else
        put "Illegal instruction: ",Command
        result -1
    end if
end ParseCommands

% Main program

    CurX := EndOfWorldX div 2    % start in the middle
    CurY := EndOfWorldY div 2
    PenIsDown := false

    % start with a blank world
    for row : 1..EndOfWorldX
        for col : 1..EndOfWorldY
            World(row,col) := 0
        end for
    end for

    N := true
    S := false
    E := false
    W := false

    loop
        var ret_code : int
        ret_code := ParseCommands
        if ret_code = 0 then
            exit
        end if
    end loop
```

B OOT Syntax

Construct	Syntax
Basic Types	`int real boolean char string`
Basic Variable Declaration	var <varname> : <type>
Assignment	:=
Input/Output	get <var name> put <var name>
Math Operators	+ - * / ** rem mod < > = <= >= not= ( )
Boolean Operators	not and or

Selection	if <condition> then
	    <statements>
	else
	    <statements>
	end if
	or
	if <condition> then
	    <statements>
	end if

Counted Repetition	for <index> : <first> .. <last> <statements> end for
General Repetition	loop <statements> exit when <condition> <statements> end loop

Continued...

Construct	Syntax
Arrays	var <name> : array <limit1> .. <limit2> of <type> <name>(<index>)
Modules	procedure <name> (<parameters>) <statements> end <name> or function <name> (<parameters>) : <type> <statements> result <value> <statements> end <name>

Records	type <recname> :
	    record
	        <fieldname> : <type>
	        <fieldname> : <type>
	    end record
	var <varname> : <recname>
	<varname>.<fieldname>

Pointers	pointer to <type> new <pointer type variable> free <pointer type variable> <pointervar>-><fieldname>

C C Syntax

<table>
<tr><th>Construct</th><th>Syntax</th></tr>
<tr><td>Header Files</td><td><stdio.h>, <string.h>, <stdlib.h>, <math.h></td></tr>
<tr><td>Program Definition</td><td>#include <header.h>
type_spec main(<parameter_list>)
{
<statements>;
}</td></tr>
<tr><td>type_spec (basic)</td><td>int float double char</td></tr>
<tr><td>Basic Variable Declaration</td><td>type_spec <varname>;
type_spec <array_name>[<size>];</td></tr>
<tr><td>Assignment</td><td>= += -= *= /= %=</td></tr>
<tr><td>Comparison</td><td>== => <= !=</td></tr>
<tr><td>Increment Operations</td><td>++variable, variable++, --variable, variable--</td></tr>
<tr><td>Input/Output</td><td>printf("%io_type\n",var_name);
scanf("%io_type",&var_name);</td></tr>
<tr><td>io_type</td><td>%c character
%d decimal
%f float
%s string</td></tr>
<tr><td>Math Operators</td><td>+ - * / % ( )</td></tr>
<tr><td>Boolean Operators</td><td>&& || !</td></tr>
<tr><td>Selection</td><td>if (<condition>)
{
<statements>;
}
else
{
<statements>;
}
or
if (<condition>)
{
<statements>;
}</td></tr>
</table>

Continued...

<table>
<tr><th>Construct</th><th>Syntax</th></tr>
<tr><td>Counted Repetition</td><td>for (<stmt1> ; <expr> ; <stmt2>)
{
<statements>;
(break;)
(continue;)
<statements>;
}</td></tr>
<tr><td>While Repetition</td><td>while (<condition>)
{
<statements>;
(break;)
(continue;)
<statements>;
}</td></tr>
<tr><td>Functions</td><td>type_spec funct_name (param_list)
{
<statements>;
return <value>;
<statements>;
}</td></tr>
<tr><td>Records (structures)</td><td>struct <recname> :
{
type_spec <fieldname>;
type_spec <fieldname>;
}
struct <recname> <varname>;
<varname>.<fieldname></td></tr>
<tr><td>Pointers</td><td>type_spec *varname;
&var1 means addr(var1)
*var2 means deref(var2)
ptrvar = malloc(sizeof(type_spec));
free(ptrvar);
<pointervar>-><fieldname></td></tr>
</table>