Computer Science 1MD3

Lab 1 – An introduction to Pascal for C programmers.

There are a few subtle, yet important, differences between C and Pascal. It is the intention of this lab to familiarize you with Pascal syntax and concepts introduced to you in C.

BASIC PROGRAM STRUCTURE:

Pascal:

```
program program_name;

include units;
global variable declarations;

procedures and functions;

begin
      {main program}
end.
```

C:

```
include standard libraries;
global variable declarations;
procedure and function protocols;

int main (void) {
      opt local declariations;
      {main program}
};

procedures and functions;
```

The first difference we may observe is the use of `begin` and `end` instead of `{` and `}` which are used for commenting. The period (`.`) following `end` is used to denote the end of the main program, any code after this will not be executed.

VARIABLE DECLARATION:

In Pascal, it is not possible to declare local variables in your main program. Instead, you may declare global variables, or, if you prefer, create a procedure `mainProgram` and declare local variables there.

Pascal:

```
var
label : type;

x     : integer; {-32768 to +32767}
y     : real;
z     : string; {up to 255 letters}
w     : char; {1 letter}

const
      label=anything
```

C:

```
type label;

int x;
float y;
char z[];
char w;

#define label anything
```

Arrays

Pascal:

```
var
arrayname : array[1..x] of type;

var
arrayname : array[1..x][1..y] of type;
```

C:

```
type arrayname[x];

type arrayname[x][y];
```

An array can be extended to any dimension you desire in this fashion. Array access is done by `arrayname[x]`, it is also very important to note that Pascal is **not** a zero referencing language. This means all arrays in Pascal will start at array element 1.

OUTPUT: *"Hello, World!"*

Pascal:	C:
`write('hello, world!');`	`printf("hello, world!");`
`writeln('hello, world!');`	`printf("hello, world!\n");`
`writeln('hello,',x,'world!');`	`printf("hello %d world!\n",x);`

The only difference between a `write` and a `writeln` is where the cursor will be placed after. The `write` places the cursor directly after the outputted text (`hello, world!_`), whereas `writeln` places the cursor on the next line (`hello, world!`
`_`).

Pascal's `writeln` is a lot more versatile then C's `printf` since it allows the use of fields. A field is the amount of spaces that a variable is allowed to be printed in. For instance:

```
x:=4.5;                         output:
writeln(x:7);                   _ _ _ _ 4 . 7
```

Note that a decimal place takes up an entire space, and also, that the field will fill from the right. If your number requires more spaces then the field you defined, the field length will be overridden to the length of your number.

In general:

`writeln(`*`var`*`:`*`field`*`);`

It is also possible to define the accuracy to which a real number is printed to the screen. For instance:

```
x:=4.657;                       output:
writeln(x:0:2);                 4.66

y:=4.5;
writeln(y:0:5);                 4.50000
```

When defining accuracy it is necessary to define a field length, the convention is to use zero when no field is desired. It should also be noted that the last number was rounded with respect to the following number (>=5 round up).

Finally it is possible to have a combination of both field length and accuracy.

```
x:=3.14159;                     output:
writeln(x:7:2);                 _ _ _ 3 . 1 4
```

In general, for real numbers:

`writeln(`*`var`*`:`*`field`*`:`*`accuracy`*`);`

CONDITIONAL STATEMENTS:

A conditional statement is something that evaluates to true or false, this is also called a binary statement. Conditional statements are used in any control structure to trigger the start or an end to the process. `i>6, i=5, i>=7` are examples of simple control structures.

The following is a list of binary operators that we may use.

Operation	*Pascal*	*C equivalent*
x greater then y	x>y	x>y
x less then y	x<y	x<y
x greater then or equal to y	x>=y	x>=y
x less then or equal to y	x<=y	x<=y
x is equal to y	x=y	x==y
Not	not	!
x is not equal to y	x<>y	x!=y
alt x is not equal to y	not(x=y)	!(x=y)

As in C, it is possible to bind two binary statements together using 'and' and 'or'.

Operation	*Pascal*	*C equivalent*
cond1 and cond2	cond1 and cond2	cond1 && cond2
cond1 or cond2	cond1 or cond2	cond1 || cond2
not cond1	not(cond1)	!(cond1)

Truth tables:

con1	con2	con1 and con2
T	T	T
T	F	F
F	T	F
F	F	F

con1	con2	con1 or con2
T	T	T
T	F	T
F	T	T
F	F	F

CONTROL STATEMENTS

If/then/else

Pascal

```
if condition then
begin
	{code}
end;
```

```
if condition then
begin
	{code}
end
else if condition then
begin
	{code}
end
else
begin
	{code}
end;
```

C

```
if (condition) {
	//code
};
```

```
if (condition) {
	//code
} else if (condition) {
	//code
} else {
	//code
};
```

Please note that in Pascal a semicolon (`;`) is only placed after the final `end` in an `if` statement.

Looping

Pascal	C

for

```
for var:= start to end do            for(var=start; var<=end; var++) {
begin                                      //code
      {code}                         };
end;

for var:= end downto start do        for(var=end; var>=start; var--) {
begin                                      //code
      {code}                         };
end;
```

while/do

```
while condition do                   while (condition) {
begin                                      //code
      {code}                         };
end;
```

repeat/until

```
repeat                               do {
      {code}                               //code
until condition;                     } while (condition);
```

ABSTRACTION

Procedures

Pascal	C

```
procedure name;                      void name() {
var declarations;                          variable declarations;
begin                                      //code
      {code}                               return void;
end;                                 };
```

There is no need for function/procedure protocols in Pascal. All your procedure are written before your main program.

Functions

Pascal	C

```
function name : returnType;          returnType name() {
var declarations;                          variable declarations;
begin                                      //code
      {code}                               return desired return;
      name:=desired return;          };
end;
```

There is no return statement in Pascal as there is in C. Instead we treat the name of the function as the variable being returned, giving it a value on the last line of the function.

Passing values to functions and procedures

```
procedure name (var1:type; var2:type; . . . varN:type);

function name (var1:type; var2:type; . . . varN:type) : returnType;
```

Invoking procedures

```
name(x,y,z. . .);
```

Invoking functions

```
x:=name(x,y,z. . .);
```

POINTERS

Declaration

Pascal

```
var
label : ^dataType;
```

C

```
dataType *label;
```

Use

operation	Pascal	C
the address of x	@x;	&x;
what x is pointing to	x^;	*x;
point x at m	x:=@m;	x=&m;
change the value x is pointing to	x^:=7;	*x=7;

ABSTRACT DATA TYPES

User Defined Types

```
type
      label = {anything, anything};

var
      x : label;
```

For example suppose we needed a variable to hold the type of a school.

```
type
      schoolType = {kindergarten, elementary, highschool, university};

var
      kindOfSchool : schoolType;
```

You may also increment a user defined data type in the same manner that you would in C. However we do not have the shorthand ++ and --, so we must say:

```
kindOfSchool:=kindOfSchool+1;
```

Records

Records are the Pascal equivalent of structures in C.

```
type
      label = record
            label1 : type;
            label2 : type;
                  .
                  .
            labelN : type;
      end;


var
      x : label;
```

Accessing record elements

```
x.label1:=™type;
x.label2:=™type;
```

COMPLETE PASCAL PROGRAM STRUCTURE

Pascal

```
program program_name;

uses unit1. . .unitN;

type
      {type and record
      declarations}

var
      {global declarations}

{procedures and functions}

begin
      {code}
end.
```

C

```
#include <library>

#define

enum

typedef

struct

dataType //global variables;

int main (void) {
      //code;
}
```

PASCAL QUICK REFERENCE

Command	**syntax**	**command**	**syntax**
set x equal to y	x:=y;	x plus y	x+y;
x multiplied by y	x*y;	x divided by y	x/y;
x modulus y	x mod y;	x to the power of y	x**y;
is x equal to y	(x=y)	is x not equal to y	x<>y
is x greater then y	x>y	is x less then y	x<y

SELF TEST QUESTIONS

convert the following C code into Pascal code

1)

```
int fact (int n) {

      if (n==1) return 1;
      return n*(n-1);

}
```

2)

```
#include <stdio.h>

enum week {sun, mon, tue, wed, thu,  fri, sat};
typedef enum week day_type;

int main(void) {
     day_type day=sat;
     printf("Day: %d\n",day);
     day++;
     if(day==sun) printf("True");
     return 0;
}
```

convert the following Pascal code into C code

```
procedure sort( var r : ArrayToSort; lo, up : integer );
var  i, j : integer;
    tempr : ArrayEntry;
     flag : boolean;
begin
     for i:=up-1 downto lo do begin
          tempr := r[i];
          j := i+1;
          flag := true;
          while (j<=up) and flag do
               if tempr.k > r[j].k then begin
                    r[j-1] := r[j];
                    j := j+1
               end
               else flag := false;
          r[j-1] := tempr
     end
end;
```