
Qassim University
جامعة القصيم

CS 237: Concepts of Programming Languages.
Project Part 1: Pascal.

Student names :

-Mohammed abdulaziz alsaud.
-Abdulrahman sulaiman.
-Abdulrahman Alslamah.

1.1 describe Pascal grammar and compare it with C grammar .

Grammar	In C	In Pascal
Variable Declaration	type variable_list; for example : int i, j, k;	var variable_list : type; for example : var age, weekdays : integer;

Decision Making	if(boolean_expression)
	statement;
	else
	statement;
	
	for example :
	if(a < 20)
	printf("a is less than 20\n");
	else
	printf("a is not less than 20\n");
		if condition then
		statement 1
		else
		statement 2;
		
		for example :
		if(a < 20) then
		writeln('a is less than 20')
		else
		writeln('a is not less than 20');

Loops	while(condition) {
	statement(s);
	}
	For example :
	int main () {
	int a = 10;
	while(a < 20) {
	printf("value of a: %d\n", a);
	a++; } return 0; }
		while (condition) do
		statement;
		
		For example :
		while number>0 do
		begin
		sum := sum + number;
		number := number - 2;
		end;

Defining a Function	return_type function_name(
	parameter list) {
	body of the function
	}
	
	For example :
	int max(int num1, int num2) {
	int result;
	if (num1 > num2)
	result = num1;
	else
	result = num2;
	
	return result; }
		function name(argument(s):type1;
		argument(s): type2; ...): function_type;
		local declarations;
		begin
		< statements >
		name:= expression; end;
		for example : function max(num1, num2:
		integer): integer;
		var result: integer;
		begin if (num1 > num2) then
		result := num1
		else
		result := num2;
		max := result; end;

grammar	In C	In Pascal
Declaring Arrays	type arrayName [arraySize] ; for example : double balance[10];	var array-name: array[index-type] of element-type ; for example : var n: array [1..10] of integer;

1.2 give a general program skeleton in Pascal .

```
program {name of the program}
uses {comma delimited names of libraries you use}
var  {global variable declaration block}


function {function declarations, if any}
{ local variables }
Begin
...
end;
procedure { procedure declarations, if any}
{ local variables }
begin
...
end;
var
begin { main program block starts}
...end. { the end of main program block }
```

1.3- Give examples for the main features of the language (function, procedure, parameter passing (by value and by reference), forward declaration, recursion, and array…).
1.4- Control statements (for, if, while, repeat, case…) should be described and used.
1.5- Run some simple programs using the features describes in 3 using a Pascal programming environment.

- Passing by reference
  And value with procedure:

```
program passing_by_reference_and_value_with_procedure;

var
  x:integer;

procedure mulByValue(x:integer); {passing by value}
begin
        writeln('Passing by value ..');
        x:= x*x;
end;

procedure mulByReference(var x:integer); {passing by reference}
begin
        writeln('Passing by reference ..');
   x:= x*x;

end;

begin
  writeln ('Enter an integer to find it's square :');
  readln(x);
  mulByValue(x);
  writeln(x); {outputs 5}
  mulByReference(x); {outputs 25}
  writeln(x);
  readln;

end.
```

Output:

```
Enter an integer to find its square :
4
Passing by value ..
4
Passing by reference ..
16
```

-forward declaration:

```
Program forward;
Procedure First; forward;

Procedure Second;
begin
  WriteLn('In second. Calling first...');
  First;
end;

Procedure First;
begin
  WriteLn('In first');
end;

begin
  Second;
  readln;
end.
```

Output:

In second. Calling first...
In first

-recursion:

```
program recursion;
var
 x:integer;
 function factorial (x:integer):integer;
 begin
if(x=1)or(x=0)then
 factorial:=1
else
factorial:=x*factorial(x-1);
 end;

begin
 write('Enter the number to see the factorial : ');
 readln(x);
 writeln(factorial(x));
 readln;
 end.
```

Output:

Enter the number to see the factorial :
5
120

- if else:

```
Program if_else;
uses crt;
var x:integer;
begin
 writeln('Enter number that is positive or negative');
 readln(x);
 if x > 0 then
 writeln(x,' is positive number ')
 else if x < 0 then
 writeln(x,' is negative number ')
 else
 writeln('it/'s 0');
 read;

end.
```

```
Output:
 Enter any number
 -3
  3 is negative number
```

- for and while:

```
program for_and_while;
var
number:array[1..3] of integer;
x:integer;
y:integer=1;
sum:integer=0;

begin
writeln('Enter 3 integer to get the sum: ');

for x:=1 to 3 do
readln(number[x]);
 while y <= 3 do
 begin
 sum := sum + number[y];
 y:= y+1;
 end;
 writeln('the sum is = ',sum );
 readln;
end.
```

```
Output:
 Enter 3 integer to get the sum:
 3
 4
 2
 the sum is = 9
```

- if-else:

```
program repeat_and_if;
 var
 x,y:integer;

 begin
  writeln('enter two numbers to find the greatest value');

      readln(x,y);

    if x > y then
 writeln(x,' is greater than ',y )
 else if y > x then
 writeln(y,' is greater than ',x)
 else
 writeln(x,' equle ',y);

 readln;

end.
```

Output:
enter two positive numbers to show
the greatest value
4
7
7 is greater than 4

- Case:

```
program case_;
var
mark:integer;
begin
  writeln('enter the mark from 1 to 100 to show the grade :');
   readln(mark);
  case mark of
  0..59 : writeln('F');
  60..69: writeln ('D');
  70..79: writeln ('C');
  80..89:writeln ('B');
  90..100:writeln ('A');
  end;
   if(mark>100)then
    writeln('Invalid input!');
    if(mark<0)then
    writeln('Invalid input!');
    readln;   end.
```

Output:
enter the mark from 1 to 100 to show
the grade :
99
A

Part 2: scheme.

2.1- Study more predefined function in addition to those studied in course.
2.2- Run some simple programs using the Scheme programming environment given in the website.

A- 2.1

What are Lists ?
Lists are the basic structured data type in Scheme.

- some list handling functions in Scheme:

-cons:
takes two arguments and returns a pair (list).

-car:
returns the first member of a list or dotted pair.

-cdr:
returns the list without its first item, or the second member of a dotted pair.

-null?:
returns #t if the object is the null list and It returns () if the object is anything else.

-list:
returns a list constructed from its arguments.

-length:
returns the length of a list.

-reverse:
returns the list reversed.

-append:
returns the concatenation of two lists.

A- 2.2

(car '(da fa f)) ---→ 'da

(cons '(j k) '(g a f)) ---→ '((j k) g a f)

(cdr '(adss fe ge eq)) ---→ '(fe ge eq)

(null? '()) ---→ #t

(null? '(s)) ---→ #f

(list 'd 'a '(a fg)) ---→ '(d a (a fg))

(list '(a r) 'h) ---→ '((a r) h)

(length '(4 51 12 4)) ---→ 4

(reverse '(a b c d)) ---→ '(d c b a)

(append '(ad (f a) r) '(q w)) ---→ '(ad (f a) r q w)

(append 'a) ---→ 'a

(define (is_even? a)(even? a)) ---→ (is_even? 5) ---→ #f

((lambda (l) (car(car l))) '((a b) cd)) ---→ 'a

- this program will find the greatest number

```
(define (max? a b)
 (cond
      ((> a b) (display "the maximum number is ")(display a))
      ((> b a) (display "the maximum number is ")(display b))
          (else  (display "numbers are equal"))
)
)


> (max? 6 5)
```

Output:

the maximum number is 6

- this program will calculate the factorial of a number.

```
(define (fact n)
  (cond
      ((= n 0) 1)
      ((= n 1) 1)
      (else (* n (fact (- n 1))))
      ) )


> (fact 7)
```

Output:

5040

- this program is to find element in the list

```
(define (find lis k)
(cond
((null? lis)(display "it’s not there"))
( (eq? (car lis) k)(display "yes it's there"))

(else (find (cdr lis) k))
  )
 )

    > (find '(a d e y z) 'z)
```

Output:

yes it's there

-This program will multiply list of numbers together.

```
(define (multi lis)
 (cond
  ((null? lis) 1)
   (else (* (car lis) (multi (cdr lis))))
    )
 )


  > (multi '(3 4 5 5))
```

Output:

300

Part 3: prolog

3.1- Translate most of the functions seen in Pascal, Scheme to Prolog.
3.2- Run these programs using the Prolog programming environment given in the website.

- This program get the factorial of a number.

```
fact(0, 1).
fact(X, Y):-X > 0 ,X1 is X-1,fact(X1,Y1),
Y is X*Y1.


      ?- fact(5,R).
```

Output:
R = 120

- This program lists all items in a list.

```
list_items([]).
list_items([X|R]):- write(X),nl,list_items(R).



 ?- list_items([5,4,3,2,1]).
```

Output:
5
4
3
2
1
true

- This program reverses a list.

```
reverse(List, Reversed) :-
        reverse(List, [], Reversed).

reverse([], Reversed, Reversed).

reverse([Head|Tail], SoFar, Reversed) :-
        reverse(Tail, [Head|SoFar], Reversed).


    ?-reverse([a,b,c,d],X).
```

Output:

X = [d, c, b, a]

- This program checks if an atom is in the list or not.

```
member(X,[X|_]).
member(X,[_|T]):- member(X,T).



  ?-member(1,[2,4,7,0]).
```

Output:

false

-This program adds a prefix or suffix to a given word.

```
put_prefix(P,C,R):- name(P,Pcode),name(C,Ccode),
   append(Pcode,Ccode,Rcode),
   name(R,Rcode).

put_suffix(S,C,R):- name(S,Scode),name(C,Ccode),
   append(Ccode,Scode,Rcode),
   name(R,Rcode).



   ?-put_prefix(un,used,R).
```

Output:

R= unused

- This program finds the nth element of a list.

```
nth(0,[X|_],X).
nth(N,[_|T],R):- M is N-1,nth(M,T,R).



?- nth(2,[a,b,c,d],R).
```

Output:

R= c