Pascal and C

This handout contrasts C and Pascal, and is adapted from one done by Antoinette Heberlein.

C construct	PASCAL construct
Simple declaration: `type ID, ID;` where type is int, float, or char (use int for boolean)	Simple declaration: `var ID, ID: type;` where type is integer, real, char, boolean

Array declaration:
`type ID[];`
example `int scores[5];`
access `scores[3]`
	Array declaration:
	`type ID = array[] of type;`
	example
	`type sc = array[0..4] of integer;`
	`var scores : sc;`
	access `scores[3]`

Complex declaration:
`struct tag {`
`  type field_name;`
`} ID;`
example `struct X {`
`  int a;`
`  float b;`
`} Z;`
access `Z.a, Z.b`
	Complex declaration:
	`type ID = record`
	`  type ID;`
	`end;`
	example `type ZT = record`
	`  a : integer;`
	`  b : real;`
	`end;`
	`var Z : ZT;`
	access `Z.a, Z.b`

Pointers
`type *ID;`
example `int *p;`
allocate space
`p = (int *)malloc(sizeof(int));`
dereference `*p = 6;`	

C construct	PASCAL construct

Pointers to struct
`struct tag *ID;`
example `struct X *pZ;`
reference field `pZ->a`
or `(*pZ)->a`
	Pointers to record
	`type ID = ^R;`
	`type R = record`
	`type field_name;`
	`end;`
	example `type pZ = ^Z;`
	`type Z = record`
	`a : integer;`
	`b : real ;`
	`end;`
	reference field `^pZ.a`

Unusual Operators `/` division (if both args are ints, it's integer division; else float) `%` remainder `&&` `||` logical and, or use lazy evaluation (when value can be determined, don't evaluate the rest) `++` increment `--` decrement `?:` a?b:c evaluates a; if it's 0, it takes on the value of c; if a is nonzero, it takes on the value of b; only one of b and c is evaluated	Equivalent Operators `/` real division `div` integer division `mod` remainder `and or` logical and, or don't use lazy evaluation so all parts of the expression are evaluated *no equivalent to the other operators in C listed at left*
Assignment lvalue = right hand side lvalue evaluates to memory location an expression, returning as value the value stored in the lvalue *example* `i = j = 0;` *shorthand* LHS op= RHS means LHS = LHS op RHS *example* `i *= 7; i = i * 7` are the same note that the LHS is evaluated once, not twice, if the short form is used	Assignment left hand side := right hand side left hand side is memory location not an expression, no value returned *example* `i := 0;` *no equivalent to the C shorthand*

Compound Statements
`{`
`statement;`
`statement;`
`}`
	Compound Statements
	`begin`
	`statement;`
	`statement;`
	`end`

C construct	PASCAL construct

Branching:
`if (expr) statement;`
`else statement;`
the else binds to the most previous unmatched if; only one statement is executed unless you use a compound statement
	Branching:
	`if bool then statement`
	`else statement;`
	the else binds to the most previous unmatched if; only one statement is executed unless you use a compound statement

Switch
`switch (expression){`
`case val1: statement1; break;`
`case val2: statement2;`
`default: statement3; break;`
`}`
expression evaluates to an integer value which is compared to the values in the cases; execution begins at the matching case; a break takes you out of the switch; if nothing is matched, the default case is executed if present; else nothing is done
	Case
	`case expression of`
	`val1: statement1;`
	`val2: statement2;`
	`no default`
	`end`
	expression evaluates to an integer value which is compared to the values; the corresponding statement is executed; failure to match any value is an error

For Loops:
`for(expr1; expr2; expr3)`
`statement;`
evaluate expr1; evaluate expr2; if expr2 is true, do statement, then expr3; reevaluate expr2; repeat until it is false
Note expr1, expr2, expr3 can be any expressions
example
`for(i = 0; i <= 10; i+=1)`
`printf("%d ", i);`
	For Loops:
	`for ID := expr1 to expr2 do`
	`statement;`
	To count down, say downto rather than to
	example
	`for i := 0 to 10 do`
	`write(i ' ');`

While Loops:
`while(expr) statement;`
evaluate expr; if true, do statement; stop loop when expr is false
example
`while(i <= 10){`
`printf("%d ", i);`
`i += 2;`
`}`
	While Loops:
	`while boolean do statement;`
	while boolean expression is true, do statement; stop loop when it is false
	example
	`while i <= 10 do begin`
	`write( i:2);`
	`i := i + 2;`
	`end`

C construct	PASCAL construct

Do … While Loop:
`do`
`  statement;`
`while (expr);`
do statement; evaluate expr; loop until expr is false
example
`do {`
`  printf("%d ", i);`
`  i += 2;`
`} while (i <= 10);`
	Repeat … until Loop:
	`repeat`
	`  statement;`
	`until boolean expression;`
	do statement; evaluate boolean expression; loop until it is true
	example
	`repeat`
	`  write(i:2);`
	`  i = i + 2;`
	`until i > 10;`

Subroutines
`return_type ID(parameter list)`
`{`
`  body`
`  return(value);`
`}`
No procedures; everything is a function
return_type is void if no value to be returned
return_type is int by default
value returned using return statement return value
	Subroutines
	`function ID(parameter list) : return_type;`
	`begin`
	`  body`
	`  ID := value;`
	`end`
	`procedure ID(parameter list);`
	`begin`
	`  body`
	`end`
	value returned by assigning to function name

Parameter Passing: always call by value pass a pointer to alter the value of a variable	Parameter Passing: default is call by value declare ID as var in parameter list for call by reference