# Modula-2 Cheatsheet

A concise cheat sheet for the Modula-2 programming language, covering syntax, data types, control structures, and module structure.

## Basic Syntax and Structure

### Program Structure

Modula-2 programs are organized into modules. A main module is the entry point.

```
MODULE MainModule;
(* Declarations *)
BEGIN
(* Statements *)
END MainModule.
```

Comments are enclosed in `(*` and `*)`.

```
(* This is a comment *)
```

Identifiers are case-sensitive and must start with a letter.

```
ValidIdentifier (* Correct *)
invalidIdentifier (* Incorrect - Case-
sensitive *)
```

### Data Types

| | |
|---|---|
| `INTEGER` | Whole numbers (e.g., `-1`, `0`, `100`). |
| `REAL` | Floating-point numbers (e.g., `3.14`, `-0.5`). |
| `BOOLEAN` | Logical values: `TRUE` or `FALSE`. |
| `CHAR` | Single characters (e.g., `'A'`, `'z'`, `'9'`). |
| `CARDINAL` | Non-negative integers (e.g., `0`, `1`, `100`). |
| `STRING` | Array of characters (e.g., `'Hello'`) |

### Variable Declarations

Variables must be declared before use. Use `VAR` keyword.

```
VAR
  age: INTEGER;
  name: ARRAY [0..31] OF CHAR;
  isReady: BOOLEAN;
```

Constants are declared using the `CONST` keyword.

```
CONST
  Pi = 3.14159;
  MaxSize = 100;
```

## Control Structures

### Conditional Statements

```
IF condition THEN
  (* Statements *)
ELSIF anotherCondition THEN
  (* Statements *)
ELSE
  (* Statements *)
END;
```

Example:

```
IF age >= 18 THEN
  WriteLn("Adult");
ELSE
  WriteLn("Minor");
END;
```

### Looping Statements

`FOR` Loop

```
FOR i := start TO end BY step
DO
  (* Statements *)
END;
```

Example:

```
FOR i := 1 TO 10 DO
  WriteInt(i, 0);
END;
```

`WHILE` Loop

```
WHILE condition DO
  (* Statements *)
END;
```

Example:

```
WHILE age < 25 DO
  age := age + 1;
END;
```

`REPEAT` Loop

```
REPEAT
  (* Statements *)
UNTIL condition;
```

Example:

```
REPEAT
  ReadInt(input);
UNTIL input > 0;
```

### CASE Statement

```
CASE expression OF
  value1: (* Statements *)
| value2: (* Statements *)
ELSE
  (* Statements *)
END;
```

Example:

```
CASE grade OF
  'A': WriteLn("Excellent");
| 'B': WriteLn("Good");
ELSE
  WriteLn("Needs Improvement");
END;
```

## Modules and Procedures

### Module Structure

A module consists of a definition part and an implementation part.

Definition Module:

```
DEFINITION MODULE MyModule;
(* Exported declarations *)
END MyModule.
```

Implementation Module:

```
IMPLEMENTATION MODULE MyModule;
(* Implementation details *)
END MyModule.
```

Importing modules:

```
MODULE MainModule;
IMPORT MyModule;
BEGIN
  (* Use MyModule's exported procedures *)
END MainModule.
```

### Procedure Declaration

| | |
|---|---|
| Procedure declaration syntax: | `PROCEDURE MyProcedure(param1: DataType; ...): ReturnType; VAR (* Local variables *) BEGIN (* Statements *) RETURN returnValue; END MyProcedure;` |
| Example: | `PROCEDURE Add(a, b: INTEGER): INTEGER; BEGIN RETURN a + b; END Add;` |

```
PROCEDURE
MyProcedure(param1:
DataType; ...):
ReturnType;
VAR
  (* Local variables *)
BEGIN
  (* Statements *)
  RETURN returnValue;
END MyProcedure;
```

```
PROCEDURE Add(a, b:
INTEGER): INTEGER;
BEGIN
  RETURN a + b;
END Add;
```

### Function Procedures

Procedures can also act as functions, returning a value.

```
PROCEDURE Square(x: INTEGER): INTEGER;
BEGIN
  RETURN x * x;
END Square;
```

## Advanced Features

### Pointers

Pointers are used to store the address of a variable.

```
TYPE
  IntPtr = POINTER TO INTEGER;
VAR
  ptr: IntPtr;
  num: INTEGER;

NEW(ptr);
ptr^ := 10;
num := ptr^;
```

`NEW(ptr)` allocates memory, and `ptr^` dereferences the pointer.

### Arrays

Array declaration:

```
VAR
  myArray: ARRAY [0..9]
OF INTEGER;
```

Accessing array elements:

```
myArray[0] := 100;
WriteInt(myArray[0], 0);
```

### Records

Records are used to group related data items.

```
TYPE
  Person = RECORD
    name: ARRAY [0..31] OF CHAR;
    age: INTEGER;
  END;
VAR
  person1: Person;

person1.name := "John";
person1.age := 30;
```