INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392-I

SMR/101 - 18

SECOND COLLEGE ON MICROPROCESSORS: TECHNOLOGY AND APPLICATIONS IN PHYSICS

(18 April - 13 May 1983)

SOFTWARE TOOLS

I. WILLERS

DD Division
CERN
CH-1211 Geneva 23
Switzerland

Two talks.

1. Programming Methods

- abstractions in programming languages

2. Program Development Cycle

- Overview
- tools.

Aim of good programming methods.

Readability

Modifyability

- We will discuss those programming methods that help us towards these aims.

1. Aim of good Program Development Meth-
od

- Write program that is required.
- Maintainability through documentation.

2. The Programming Cycle.

- The computer tools that you will use.

Programming languages, Abstractions.

Unstructured programming.

Structured programming.

Procedural abstraction.

- Structure Charts.

Module/package abstraction.

[Task structure]

- [Filters]

Programming languages, Abstractions.

Unstructured data.

Structured data.

[Relational Database]

- [Data model]

Data hiding.

[Data flow]
- [Data flow diagrams]

Program Structure.

Structure implies readability.

Readability implies correspondance between time and text.

Program Structure.

- Program can be dissected into parts each of which has a start and end point.

Example from Pascal.

```
if <test> then
   <body>
```

Example from Pascal.

```
if i=0 then
   count := count + 1
```

Example from Pascal.

```
if <test> then
   <body-1>
else
   <body-2>
```

Example from Pascal.

```
if x>y then
   begin max:=x; min:=y end
else
   begin max:=y; min:=x end
```

Example from Pascal.

```
case <expression> of
     value-1: <body-1>
     value-2: <body-2>
        . . .
end
```

Example from Pascal.

```
case month of
     1,3,5,7,8,10,12:    NrDays:=31;
     4,6,9,11:           NrDays:=30;
     2:                  if leapyear then
                            NrDays:=29
                         else
                            NrDays:=28
end
```

Example from Pascal.

```
for var=<first value> to <last value> do
              <body>
```

Example from Pascal.

```
for month=1 to 12 do
   writeln(month,daysin(month))
```

Example from Pascal.

```
while <test> do
    <body>
```

Prove to yourself that your loops will terminate.

Example from Pascal.

```
while power<10 do
   power := power * 10
```

Prove to yourself that your loops will terminate.

Example from Pascal.

```
repeat
   <body>
until <test>
```

Prove to yourself that your loops will terminate.

Example from Pascal.

```
repeat
   read(character)
until character ◇ ' '
                              ◇ is unequal
```

Prove to yourself that your loops will terminate.

Data Structure.

Structure implies readability.

Readability implies correspondance between naming, operations and meaning.

Example from Pascal.

Boolean type	or and not write #
Char type	chr read # write #
Integer type	+ - * div mod abs sqr odd read # write #

etc.

Example from Pascal.

```
Overdrawn := balance < 0


if (char>='A') and (char<='Z') then
   write(char)


number := number div 10
```

Example from Pascal.

```
type month = (Jan,Feb,Mar,Apr,May,Jun,
              Jul,Aug,Sep,Oct,Nov,Dec);

case month of

Jan,Mar,May,Jul,Aug,Oct,Dec:
                          NrDays:=31;

Apr,Jun,Sep,Nov:          NrDays:=30;

Feb:                      if leapyear then
                             NrDays:=29
                          else
                             NrDays:=28

end
```

Example from Pascal.

An ARRAY is a data structure composed of a fixed number of components, all of the same type. An array component is selected by an INDEX.

```
line = array[1..80] of char;


read(line);

for i:=10 to 20 do
                  write(line[i])
```

Example from Pascal.

A RECORD is a data structure composed of a fixed number of components (fields), which may be of different types. FIELDS are selected by name.

```
date = record
             day:1..31;
             month:Months;
             year:integer
end
```

```
date.day := 2;
date.month := May;
date.year := 1983;
```

Example from Pascal.

A PROCEDURE consists of a block of code which may be invoked, with or without parameters, elsewhere in the program.

It is the programmers means of building abstractions which do not exist in the programming language.

It effects readability

- since it divides programs into small parts each of which can be understood easily.

Divide and conquer.

Example from Pascal.

```
program printtables;

tables = array[1..12;1..12] of integer;

procedure MultiplicationTable;

   .
   .
   .

procedure PrintTable;

   .
   .
   .

begin
   MultiplicationTable;
   PrintTable;
end.
```

Top Down Programming.

A program which is designed from the highest level concentrates on the function of the procedures from the highest level. It is therefore easy to read and modify.

A program which is designed from the lowest level (bottom up) concentrates on the roles that different procedures play and is difficult to read.

A compromise is obtained in the Module or Package abstraction where new operations on data items can be grouped together.

The Structure Chart.

Complete Development Cycle

Requirements

- yes/no answers when finished

Specification

- In what way will the requirements be satisfied

Data Model

- Properties of the data

Programs/structure charts

- documentation of the program

[tasks/data flow diagrams]

N.B. In perfect world this is not cyclic however the concrete world of the data model may effect your ideas on the specification or requirements.

Requirements and Specifications

Requirements

- yes/no

Specification

- What and how.

DO NOT WRITE THE WRONG PROGRAM

Modifyability

- Reduce to a minimum

Requirements and Specifications

Example Requirement

The program must at any time be able to inform the user of the time of day.

Example Specification

When the program has given a prompt, the user will type 'TIME' followed by carriage return in order to obtain the time. The program will print the time in the following format 'hh:mm:ss' where hh is the hour, mm the minutes and ss the seconds. This will be according to the 24 hour clock.

Program Development Cycle

Data Model

- Describe the attributes and properties of the data.

car := [REGISTRATION NUMBER, make, model, colour]

REGISTRATION NUMBER
- is that number issued by the authorities which uniquely identifies the car.

.

.

.

colour
- is the colour inscribed on the car's papers.

Program Development Cycle

Data Model

model := [MAKE, MODEL, engine size, power, body type]

.

.

.

body type

- will be one of [sedan, wagon, truck]

Program Development Cycle

The Tools

Edit

Compile

[Link]

Test

The mechanical process.

Editing

Line Editor

- References by line number.
- Line numbers remain fixed.
- Gaps left so that lines can be inserted.
- Displays current line.

Context Editor

- References by strings in text.
- Refer to nth. line in text.
- Displays current line.

Full Screen Editor.

- References by pointing.
- Displays many lines.

Language Oriented Editor

- Helps with and checks syntax of text.

Editing

```
input 100
100= SET I=1
110= SET J=1
120= I=I+1
130= J=J+1
140= GOTO 120
150= ***

insert 135
131= I2=I**2
132= J3=J**3
133= IF I2>J3 GOTO 130
134= IF I2=J3 THEN PRINT I2,J3
135= I=I+1
136= GOTO 131
137= ***

delete 100,110

etc.
```

Editing

```
find 'name'

change 'me' to 'you'

next 2

etc.
```

Language Translation

A language translator translates a higher level language into a lower level language.

It includes

compilers
interactive interpreters
interpreters
preprocessors
[assemblers]

The lowest level consists of the bit patterns which the computer understands and obeys. A load module contains that information.

Assembler (known)

There is a one-to-one correspondance between the language text and the machine code that the computer understands.

op-code <arguments>

Assembler looks up op-code for corresponding bit pattern and format of the instruction.

The format enables it to understand the arguments which are converted into bit patterns.

Normally, two passes since the arguments can refer to things which occur textually later

```
e.g.        JMP LAB
                 .

      LAB
```

The user has access to all machine capabilities.

Compiler

The compiler is free to generate whatever code it wants that will do the job.

User has lost control of the machine but has aids to programming.

Stages of compiling

language text

Lexical Analysis

Syntax Analysis

Code Generation

load module

Language can be machine independent e.g. Fortran, Pascal, Basic etc.

Lexical Analysis

The lexical analyser recognises words and symbols in the text. A compiler turns those into an internal representation.

e.g. if i=0 then i:=1;

could translate into

reserved word if	8
symbol i	12 1 'i'
logical operator =	32
integer constant 0	29 0
reserved word then	9
symbol i	12 1 'i'
assignment operator	40
integer constant 1	29 1
semicolon	41

Syntax Analysis

This turns the output of the lexical analyser into a structure known as the SYNTAX TREE.

e.g. if i=0 then i:=1;

From this structure code or intermediate code is generated.

Intermediate code

language text

Lexical Analysis

Syntax Analysis

Intemediate Code Generator

intermediate code

Code Generator Code Generator

object code A object code B

Run Time Support

The run-time support is code that contains and initialises the programs environment.

Interactive Interpreter

one statement of
language text

Lexical Analysis

Syntax Analysis

Perform Operation

e.g. Basic

Interpreters (not interactive)

language text

Lexical Analysis

Syntax Analysis

Interpretive Code Generation

interpretive code

Interpreter

A compiler's intermediate code may be used as an interpretive code.

e.g. Interpretive Pascal (UCSD), ...

Language Preprocessors

text for language A

Language Preprocessor

text for language B

COMPILER

load module

Language A is richer than language B.

e.g. MORTRAN, Ratfor, ...

Testing

Testing shows the presence of bugs.

Testing does NOT show the absence of bugs.

1. Exercise your program in everyway that you can in order to remove all 'infant mortallity' bugs.

2. Read your program looking for cases not accounted for, checks not made and errors still present.

3. Get someone-else to read your program.

MAKE YOUR PROGRAM READABLE

Other Tools

Debuggers - ROSY is a typical microprocessor debugger. High level language debuggers with assertions are beginning to appear.

Document Formatters - many.