Dexterity
The Value Accelerators

Quantum Syntax Training

Market Research

Market research is the collection and analysis of information about consumers, competitors and the effectiveness of marketing programs.

Small business owners use market research to determine the feasibility of a new business, test interest in new products or services, improve aspects of their businesses, such as customer service or distribution channels, and develop competitive strategies.

Create sophisticated survey data tables

Survey data is only useful when it's accurate, understandable, and current. Quantum is a powerful tool that improves data quality and provides useful tables—quickly.

Introduction to Quantum

- Quantum is a highly sophisticated and very flexible computer language designed to simplify the process of obtaining useful information from a set of questionnaires.
- Quantum has been designed with market researchers in mind so its syntax and grammar are similar to English. Nevertheless, it is still a computer language and as such should be used with precision and understanding.

What Quantum does

Dexterity
The Value Accelerators

Quantum is very flexible language which performs a variety of tasks. It can:

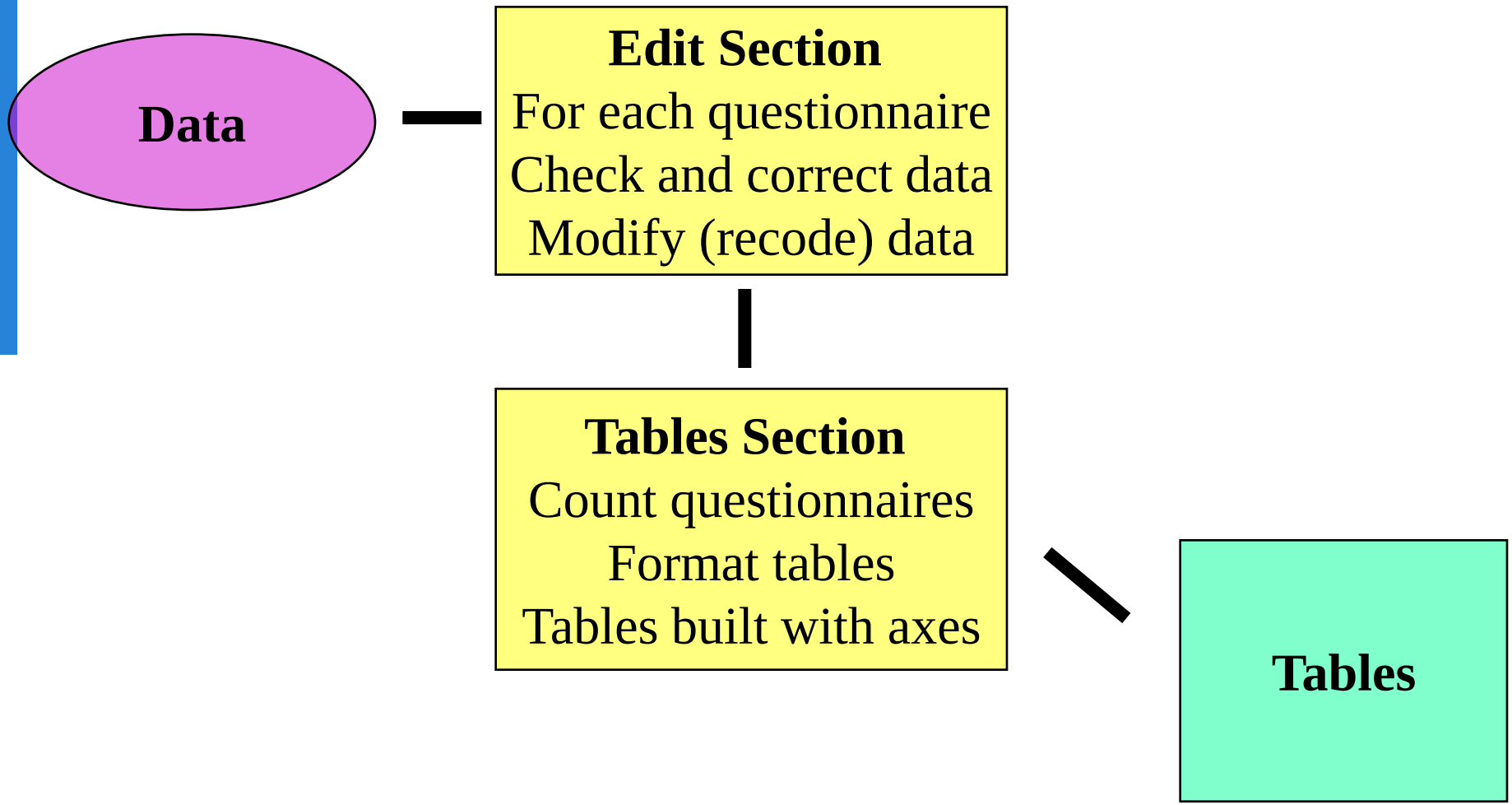
- Check and validate the data.
- Edit and correct the data.
- Produce different types of lists and reports of data.
- Produce new data files.
- Recode data and produce new variables.
- Generate tables.
- Perform Statistical calculations.

Stages in Quantum run

First the data is read onto disk. Data on disk can come from a number of different sources, for example:

- It may be entered directly via a terminal by a telephone interview using Quancept CATI.
- It may be collected over the World Wide Web using software such as Quancept CAWI.
- It may be entered directly into a computer by an interviewer conducting a personal interview using Quancept CAPI.
- It may be entered by a data entry clerk using a data entry package.

Stages in a Quantum Run



A simple Quantum program

```
struct;read=2;ser=c(7,10);crd=c11
ed
      if (c112n'1') reject ; return
end
a;op=12;dsp;flush;decp=0

tab q1 brk1

l q1
ttlQ1. Whether bought soft drinks in last 4 weeks
n10Total
col 124;Yes;No

l brk1
n23Sex;unl1
col 120;Male;Female
n23Age;unl1
col 121;18-34;35-54;55+
```

A simple Quantum program

A typical program might look like this:

```
Struct;read=2;ser=c(5,8);crd=c(9,10);max=32
*include vars
ed
   *include edit
end
a;dsp;spechar=-*;decp=1;flush;wm=0;axttr;
+dec=0;rinc;acr100;dp;nsw;nopage;notype;
+paglen=64;pagwid=145;
wm1 wax1 wax2;rim;input;
+20;30;50;
+50;50;
+33;33;33
*include tabs
*include axes
*include breaks
```

Structure of the Record

External Variables and Arrays are declared in a file called Vars and included before including edit section

Edit section will have calculations of counts, column settings to get counts which are not straight-forward.

Global commands which control the overall characteristics of a run

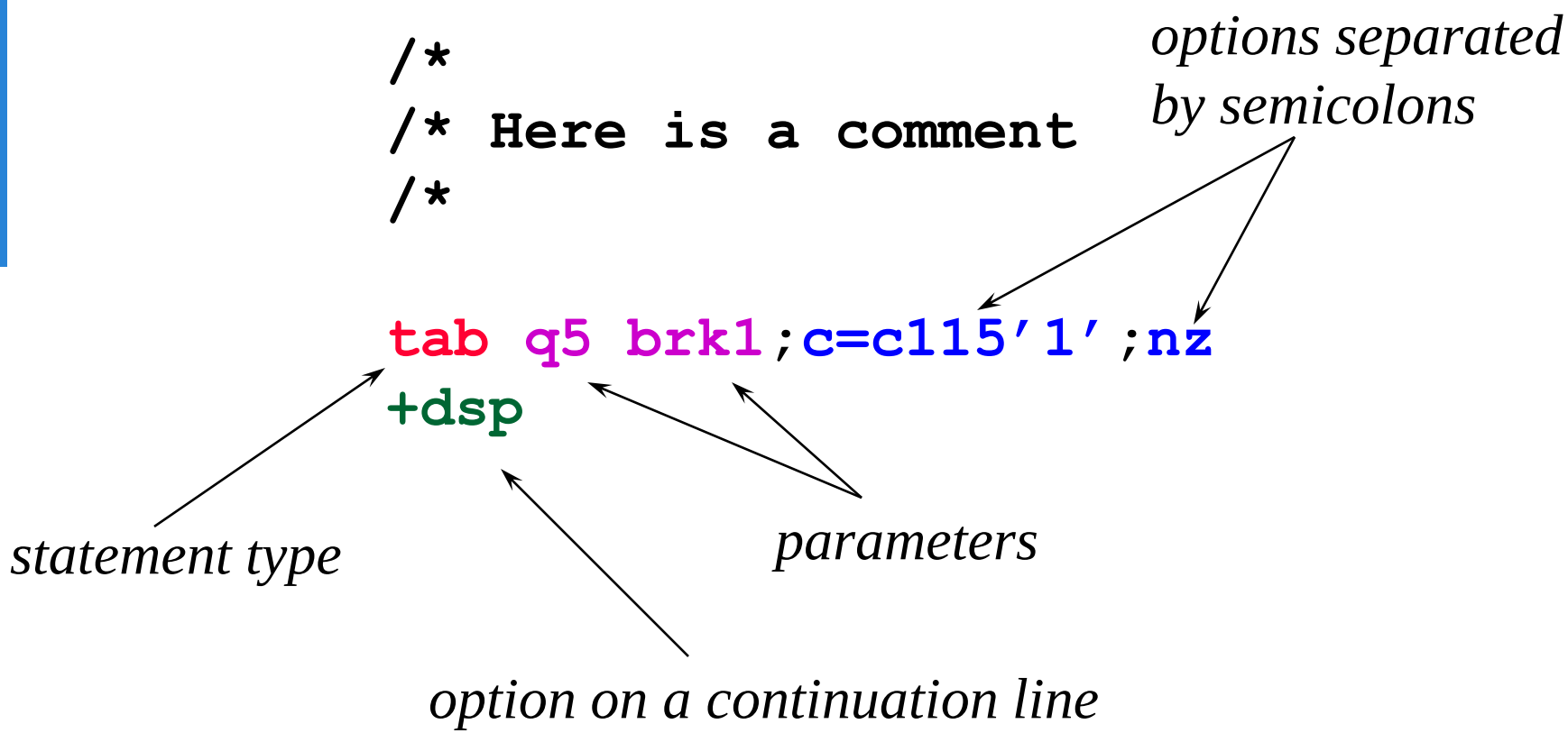
Weights related stuff

Will have details of what to be tabulated with what in order to get a table

Contains the definitions of all variables used as Rows

Contains the definitions of all variables used as Columns

A Sample of Quantum Programming



Describing Data Structure

- data is fixed-position "***columns***"
- data is read into the ***c*** array
- the ***struct*** statement defines how
- first statement in a Quantum program

Specifying Long Records

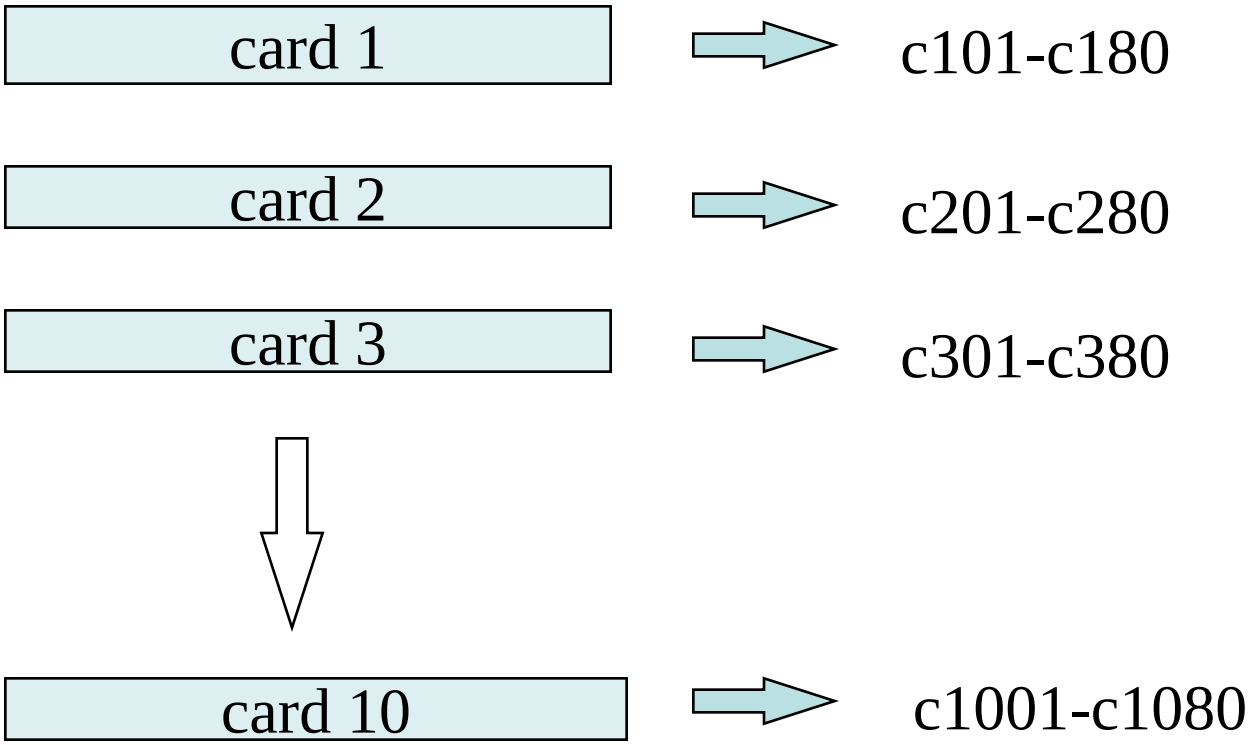
struct;read=0;ser=c(1,7);reclen=1000

columns 1 to 200

read into → c1 to c200 of the c array

Specifying Multi-card Data Sets

struct;read=2;ser=c(1,5);crd=c(6,7);max=23



Absolutes/col percents

Q.15a Satisfaction level

Age axis

Q15a axis

	Total	Age of respondent 18-25	26-35	36-45	46-55	55 +
Base	2347	246	663	663	517	257
Much better than expected	596	70	168	156	140	62
	25%	28%	25%	24%	27%	24%
A little better than expected	848	88	247	252	176	84
	36%	36%	37%	38%	34%	33%
As expected	603	62	164	170	128	79
	26%	25%	25%	26%	25%	31%
A little less than expected	219	20	68	63	49	19
	9%	8%	10%	10%	9%	7%
Much less than expected	46	4	11	12	12	7
	2%	2%	2%	2%	2%	3%
DK/NA	35	2	5	10	12	6
	1%	1%	1%	2%	2%	2%

Defining Axes

- Usually define each question in a questionnaire
- Cross tabulated to create tables
- Can be used as either rows or columns

Specifying the name of an axis

- May consist of letters, numbers and the _ character
- Must start with a letter
- Must be unique in the program
- Examples:

```
l q1
l sex
```

Specifying Table Titles In The Axis

- ***ttl*** left justified
- ***ttc*** centralised
- ***ttr*** right justified
- Example:

```
l q15
ttlQ.15 Opinion of performance
ttl
ttlBase: All answering
```

Defining simple elements using the col statement

- Format:

  `col N;Base=text;hd=Subheading;element defs`

- Where:

  N is the number of the column to be analysed.

  Base is an optional keyword. Prints a total row or column.

  hd= is an optional keyword. Prints a subheading

Element definition on col statements

- Text required on table for row/column
- Separated by ; or by + on subsequent line
- Implicit order of coding

  1234567890-&

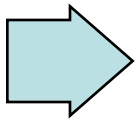
Quantum Axis Using col

Original question

Q.3 How old are you?

(118)

18-24……..(1)
25-34……..(2)
35-44……..(3)
45-54……..(4)
55-64……..(5)
65 or older……..(6)



Quantum axis

```
l q3
ttlQ.3 Age of respondent
ttlBase: All Respondents
col 118;Base
+18-24
+25-34
+35-44
+45-54
+55-64
+65 or older
```

Overriding The Built In Order Of Punches

- Must add = to end of text with code specified
- place ‘ (single quotes) around multiple codes (creates an *or* condition)
- Example

```
col 137;Brand C=3;Brand A/B=‘12’
```

- / to denote a range of codes in default order

```
‘3/7’ = ‘34567’
```

Picking Up No Answer Records

- =*rej* at end of axis
- picks up all records not appearing in preceding elements
- Used to pick up DK's or NA's (usually blanks)

```
col 118;Base;18-24;25-44;45-64;65+
+DK/NA=rej
```

Cross-Tabulating Axes

- tab statement
- 2-dimensional table

tab *axis1 axis2*

where

axis1 defines the rows

axis2 defines the columns

tab q15a age

Page 1

Absolutes/col percents

Q.15a Satisfaction level

age

q15a

	Age of respondent					
	Total	18-25	26-35	36-45	46-55	55 +
Base	2347	246	663	663	517	257
Much better than expected	596 25%	70 28%	168 25%	156 24%	140 27%	62 24%
A little better than expected	848 36%	88 36%	247 37%	252 38%	176 34%	84 33%
As expected	603 26%	62 25%	164 25%	170 26%	128 25%	79 31%
A little less than expected	219 9%	20 8%	68 10%	63 10%	49 9%	19 7%
Much less than expected	46 2%	4 2%	11 2%	12 2%	12 2%	7 3%
DK/NA	35 1%	2 1%	5 1%	10 2%	12 2%	6 2%

Your Quantum Program

Dexterity
The Value Accelerators

/* Data Structure

struct;…

/* Tables section

tab …

/* Axes definitions

l …

ttl

col

Exercise 1a

Running Quantum

- Run with the ***quantum*** command at the operating system prompt
- Format

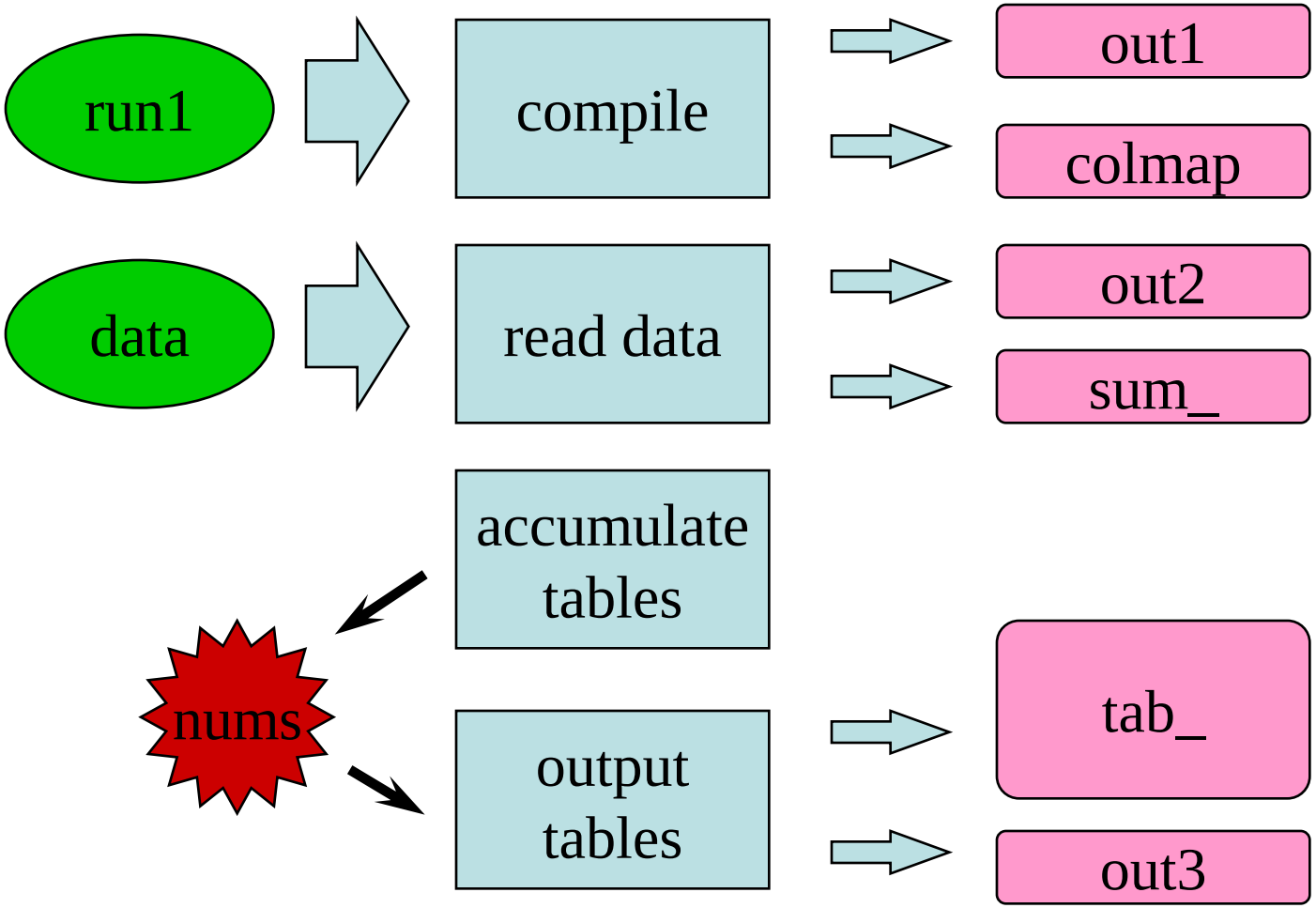
  quantum *run-file data-file*

  Where:

  run-file is the name of the Quantum program

  data-file is the name of the data to be analysed

Running Quantum



Keeping a log of your run

- *-l* option

  quantum -l run1 data

- creates a file named *log* automatically
- Useful for tracking problems

Checking the run for errors

- compile only option to check syntax before running data
- *-c* option

  quantum -c run1
- Check in *out1* for errors

Output files and cleaning up

- *out1* program listing
- *colmap* shows codes/columns used
- *out2* data errors and information
- *sum_* summary of data errors
- *tab_* output tables file
- *out3* summary of output
- Cleanup with *quclean*

  quclean -y -a

Outputting Tables With a Different File Name

- third argument to command

  quantum run1 data tables
- creates a file named *tables* instead of *tab_*

Running More Than One Run in the Same Directory

- write temp files to temp directory using *-td*
- create unique names for output files with *-id*

  quantum -td tmp1 -id r1 run1 data

  quantum -td tmp2 -id r2 run2 data
- creates files named

out1.r1, out1.r2, tab_.r1, tab_.r2

Exercise 1b

Dexterity
The Value Accelerators

Output formatting options

Global Options

- Specified on the ***a*** statement
- Format

  a*;options*
- After struct statement but before tab statements

Global Options

Option	Action	Default
op=	output type(s)	op=1

op=1 absolutes

op=2 col %'s

op=0 row %'s

Multiple types are combined

e.g. op=12 for abs & col %'s.

Example Global Options

Option	Action	Default
dsp	double spacing	nodsp
decp=	dec places for %'s	decp=1
flush	%s line up with abs	noflush
pagwid=	Page Width	132

Defining a Global Title

- tt statements immediately after a statement

```
a;op=12;dsp;flush;pagwid=199
ttcJ4538 - Project Xanadu
ttcLocal Research Ltd
```

Modifying Global Options

- a statement options also valid on tab statements

  a;op=12;decp=0

  tab q3 brk1;op=01;decp=2
- options on tab override options on a
- options on tab for that table only

Using Sectbeg to Define Table Sections

- Define table sections in the tab section of run
- sectbeg/sectend statements
- can be nested
- sectbeg can specify options or be followed by titles

```
sectbeg;op=01
ttlSection A
```

Using Sectbeg to Define Table Sections - Example

```
a;op=12;dsp;flush;spechar=-*;decp=0
ttcProject Xanadu

sectbeg
ttlSection 1
tab q1 bk1
tab q2 bk1
sectbeg;op=120
tab q3 bk1
tab q4 bk1
sectend
tab q5 bk1
sectend
sectbeg
ttlSection 2
...
```

Specifying Table Numbers

- tbr/tbl before tab statements

tbl 1

tab q1 brk1

tab q2 brk1

tbl 23

tab q7 brk1

Controlling Page Numbering

- Page numbers are the default
- May be switched off with nopage on a
- Can use page in tab section

```
pag 123
tab q23 brk1
tab q23a brk1
```

More Flexible Table and Page Numbering

- On a tt statement with nopage on a statement

  a;nopage …

  ttcTable **<<tab>>** - Page **<<pag>>**

Footnotes

- Use foot for foot of table or bot for bottom of page
- Statement before tt statement(s)

```
a;op=12;notype;flush;dsp;decp=0
ttcProject Wolverine - J.5056
bot
ttl
ttlPrepared by ATP Ltd.
ttlTable <<tab>>
ttrPage <<pag>>
```

Program Structure Review

Dexterity
The Value Accelerators

```
a;op=12;notype;flush;dsp;decp=0
ttcProject Wolverine - J.5056
foot
ttlPrepared by ATP Ltd.
ttlTable <<tab>>

sectbeg
ttlSection 1
tab q1 bk1
tab q3 bk1
sectend

l q3
ttlQ.3 Age of respondent
ttlBase : All respondents
col 118;Base
+18-24
+25-34
+35-64
+65 or older
```

Global options, title & footnote

tabs section to specify tables required.

axes section to define questions in detail

Controlling Breakdown Layouts

- Quantum automatic headings
- Use n23 statements for overall headings
- unl1 for underlining

n23Region;unl1

Controlling Breakdown Layouts

n23Region;unl1

col 125

+England

+Scotland

+Wales

+Northern Ireland

```
                Region
-------------------------------------
                                Northern
England   Scotland     Wales     Ireland
```

Summary/Order of Statements

a	global options
tt	
tb	table number
pag	page number
sectbeg/sectend	options for a block of tabs
tt	
tab	table definition
tt	
l	axis name
tt	
col	element definitions
n23	breakdown headings

Reformatting Tables

- run-time option if numbers don't change

  quantum -o run1

- Only works if previous run used quantumx

Exercise 2

Analysing Numeric and Field Data

Analysis of Numeric Data

- Numeric data examples:

  Actual age - 2 digit field

  Make/model of car - 4 digit field

The val Statement

- Format:

  val *variable*;Base;hd=*Text*;*operator*

  +*element defs*

- where *operator* is one of

  =, i or r

Specifying Individual Values

- Uses the = operator (test for equality)
- Example:

```
ttlQ.10 Bus Journeys in Last week
val c(210,211);Base;=
+0 journeys
+1 journey
+2 journeys
+3 journeys
```

Handling of Numeric Data

- Within a numeric field leading and trailing blanks are ignored
- A blank field has the numeric value 0
- Non-numeric fields are also treated numerically as 0
- Can use option *missingincs* on a statement to force blanks/non-numerics to be missing values

Analysing Values Not in the Text

- If text not numeric add =*value* to end
- Example:

```
ttlQ.17 Car owned
val c(150,153);Base;=
+Austin Rover=1000
+Bristol=1200
+Citroen=1250
```

Analysing Ranges of Values

- Uses the i operator (test for integer range)
- Example:

```
ttlQ.50 Age
val c(421,422);Base;i
+15-24 years
+25-34 years
+35 or older
```

Mixing Single and Range Values

- Example:

```
ttlQ.14 Magazines read last week
val c(215,216);Base
+=;None=0;1;2;3
+i;4-6;7-10;11 or more
```

Data in Multiple Fields

- Alternative to multi-punching
- Estimate the maximum number of likely responses and allocate a field per response
- Number is allocated to each response
- Typically field is 2 columns wide allowing for responses 00 to 99

Data in Multiple Fields

- Questionnaire:

```
Q.10 What do you like about this
product? WRITE IN VERBATIM


|_||_| |_||_| |_||_| |_||_|  |_||_|
16-17  18-19  20-21  22-23  24-25
```

- Code Frame:

```
01 General appearance/look/feel
02 Like it/very nice/other likes
03 Similar to (usual) brand
04 Like it because of advert
05 Colour is relaxing
```

The fld Statement

- Similar to col and val statements
- Format:

```
fld col-specs;[Base];[hd=];Elm-specs
```

- Where:

col-specs **are the fields to be analysed**
Base **is an optional base line**
hd= **is an optional sub-heading**
Elm-specs **define the elements**

Column Specification on fld

- Range of fields on a series of columns

```
col-start, col-end : field-width
```

- For example:

```
fld c116, c124:2; …..
```

Column Specification on fld

- Non-sequential series of fields specified in brackets.
- All fields must be the same width
- Start column of each field is specified, separated by commas
- For example:

```
fld ( c125, c134, c167 ):3
```

Column Specification on fld

- Several separate sequences of fields
- Define the start and end column of each series
- Separate each group by slashes
- For example:

```
fld c210,c216/c310,c316:2
```

Definition of Elements on fld

- Simplest form is implicit order with text elements separated by semi-colons
- To override specify the new value

```
fld c210,c220:2;Base;Like appearance=23
```

- Ranges may be specified:

```
fld c210,c220:2;Base;Like quality=23-26
```

- Discontinuous ranges:

```
fld c210,c220:2;Base;Likes=23-26,35
```

- Alphanumeric strings enclosed in $'s:

```
fld c210,c220:2;Base;...DK/NA=$&&$
```

Exercise 3

Dexterity
The Value Accelerators

Logical Expressions

Logical Expressions

- Form the basis of filtering in Quantum
- Boolean value - true or false
- col, val, fld have built in logical expressions
- More complex analysis requires more complicated logical expressions

Constants

- 3 types:
- Punch codes (*1234567890-&*)
- Strings of ASCII characters
- Numbers (integer or real)

Punch codes

- Referenced in apostrophes
- Refer to single column of data
- Punches listed individually
- Range denoted by /
- Examples

'1'	punch 1
'123'	punches 1 or 2 or 3
'1/5'	punches 1 or 2 or 3 or 4 or 5
' '	no punches (blank)

Strings

- Referenced in $ signs
- Refer to more than 1 column of data
- Examples

1234	4 character string
ABC	3 character string
$ $	blank string of ANY length

Numbers

- No delimiters
- In range

  -2,000,000,000

  to

  +2,000,000,000 *(approx)*

Data Variables

- Single column data variables c*N* where *N* is the column number
- Field of columns c(*m*,*n*) may be either string or numeric

Logical Expressions Using Punches

- c*N*'*p*' true if '*p*' occurs in column *N*
  c15'1'
  c23'1/57'
- c*N*n'*p*' true if '*p*' is not in column *N*
  c109n'23'
- c*N*='*p*' true if column *N* is *exactly* '*p*'
  c230='1'

Logical Expressions Using Strings

- c(*m*,*n*) = $*string*$ equal to
  c(101,104)=1512
  c(353,380)=$ $
  (special case for blank strings)
- c(*m*,*n*) u $*string*$ *un*equal to
  c(10,15) u $August$

Logical Expressions Using Numeric Values

.eq.	equal to	c(101,104).eq.151
.ne.	not equal to	c(350,353).ne.0
.gt.	greater than	c(10,11) .gt. 64
.ge.	greater/equal	c(123,124).ge.2
.lt.	less than	c(23,25).lt.500
.le.	less than/equal	c587.le.5

Logical Expressions Comparing Variables

c(101,104) = c(901,904) checks strings

c(101,104) .eq. c(901,904) checks numerics

c19 = c23

c(155,158) .gt. c(160,164)

Building Up Logical Expressions

- Using combination operators

  .or. .and.

  .not. ( )

- Use ()'s to avoid ambiguity
- Examples

  c(101,104)=0105 .and. c115n'3'

  c25'1' .and. (c19='1' .or. c20'1/5')

  c(310,311).ge.20 .and. c(310,311).le.29

Exercise 4

More Complex Axis Definitions

Applying a Logical Expression

- Logical expressions specified by the c= option
  c=*logical-expression*
- May be used on a, sectbeg, tab, l statements
- Example

```
tab q1 brk1;c=c155'12'
ttlBase: All using test product
```

Defining Individual Elements

- n statements define individual axis elements
- General form

  n*NNText*;*options*

  where *NN* is a 2 digit number

Elements that use logical expressions

- n10 Base element
- n01 Ordinary axis element
- Format:

  n01*Text*;*options*
- Example options:

  c=*logical-expression*

  c=-*N*

  nz

n statements instead of col

col 130;Base;Red;Blue;Green;Yellow;DK/NA=rej

is the same as:

n10Base
n01Red; c=c130'1'
n01Blue; c=c130'2'
n01Green; c=c130'3'
n01Yellow; c=c130'4'
n01DK/NA; c=-

Text only elements

- n03 Text only
- Only prints when axis used as side of table
- Underlining with unl1 option
- Example:

n03Reasons for preferring product;unl1

n03 *(creates a blank row)*

Filters Within an Axis

```
l agesex
ttlAge within sex
n10Base
n00;c=c115'1'
n03Male;unl1
n03
col 116
+18-24
+25-44
+45+
n00;c=c115'2'
n03Female;unl1
n03
col 116
+18-24
+25-44
+45+
```

⇦ filter for Males

⇦ filter for Females

Non-printing Elements

- n11 non-printing (*invisible*) n10
- n15 non-printing n01

Exercise 5

Simple Statistics

Statistics Based on Relative Weights

- Usually applied to scale or rating questions
- factor applied to each answer using fac= option
- factors are numeric and may be positive, negative or decimal numbers
- Need to be followed by appropriate statistical n statements

Statistics Based on Factors

- The statistical n statements have the normal format

  n*NNText*;*options* (except c=)

  n12 Mean

  n17 Standard Deviation

  n19 Standard Error

  n20 Error Variance

Statistics Based on Factors

- Example:

```
n01Very good  (+2);                          c=c215'1';fac=2
n01Quite good (+1);                           c=c215'2';fac=1
n01Neither good nor poor (0);     c=c215'3'; fac=0
n01Quite poor (-1);                          c=c215'4';fac=-1
n01Very poor (-2);                    c=c215'5';fac=-2
n12Mean;dec=2
n17Standard deviation;dec=3
n19Standard error
n20Error variance
```

Means on Ranges

- Normally use mid-points
- Example

```
n0118-24;          c=c118'1';fac=21
n0125-34;          c=c118'2';fac=29.5
n0135-44;          c=c118'3';fac=39.5
n0145-54;          c=c118'4';fac=49.5
n0155-64;          c=c118'5';fac=59.5
n0165+;                    c=c118'6';fac=70
n12Average Age
```

col statements with factors

- Can specify individual factors on a col but don't forget to use %fac=
- Can also specifiy a start value and automatic increment for subsequent elements
- Format of option

  %fac=*startvalue* [+/-] *increment*
- If need to switch off automatic factors use %nofac

col statements with factors

- Example

```
col 215;Base
+Very good; %fac=5-1
+Quite good
+Neither good nor poor
+Quite poor
+Very poor
+No answer=rej; %nofac
n12Mean;dec=2
```

Statistics On Numeric Values

- Use the n25 statement with inc= option
- n25 is non-printing
- inc= option specifies numeric field to look at
- Can use c= to restrict entry to stats calculation
- Follow the n25 with appropriate stats n statements

Calculating Statistics On Numeric Values

- Example-

  Age coded cols 10-11 on card 1:

  val c(110,111);i
  +18-34;35-54;55+
  n25;inc=c(110,111);c=c(110,111)u$ $
  n12Average Age

Exercise 6

Dexterity
The Value Accelerators

Include Files

Include files

- Can split a Quantum program into several files
- Can "include" these files in the main run file
- Uses the *include statement
- Format

  *include *filename*
- Can be local file or anywhere on system (with full or relative pathname)

Include statement

axes

```
l q15a
*include q15.qin;prod=Washo
+col(a)=131
```

```
l q15b
ttlQ.15 Rating of product Clean-it
ttlBase: All respondents
col 132;Base
+Very good;Quite good
+Neither good nor poor
+Quite poor;Very poor
```

q15.qin

```
ttlQ.15 Rating of product &prod
ttlBase: All respondents
col a0;Base
+Very good;Quite good
+Neither good nor poor
+Quite poor;Very poor
```

Include statement for 2nd axis

axes

```
l q15a
*include q15.qin;prod=Washo
+col(a)=131


l q15b
*include q15.qin;prod=Clean-it
+col(a)=132
```

q15.qin

```
ttlQ.15 Rating of product &prod
ttlBase: All respondents
col a0;Base
+Very good;Quite good
+Neither good nor poor
+Quite poor;Very poor
```

Substitution with c variables

axes

```
l q15a
*include q15.qin;prod=Washo
+col(a)=131
```

```
l q15b
*include q15.qin;prod=Clean-it
+col(a)=132
```

q15.qin

```
ttlQ.15 Rating of product &prod
ttlBase: All respondents
col a0;Base
+Very good;Quite good
+Neither good nor poor
+Quite poor;Very poor
n01Very/quite good;c=ca0'12'
n01Very/quite poor;c=ca0'45'
```

Referring to more than one column

axes

l q20a

ttlNumber of Bus journeys
ttlBase: All respondents
val **c(216,217)**;Base;=
+0;1;2
+i
+3-5;6-9;10+

l q20b
ttlNumber of Car journeys
ttlBase: All respondents
val **c(218,219)**;Base;=
+0;1;2
+i
+3-5;6-9;10+

Referring to more than one column

- col(a) substitutes the base of an array, a0, a1, a2 … a*N*
- if col(a)=131
- then

  a0=131

  a1=132

  a2=133 etc

Referring to more than one column

axes

```
l q20a
*include q20.qin;vehicle=Bus
+col(a)=216

l q20b
*include q20.qin;vehicle=Car
+col(a)=218
```

q20.qin

```
ttlNumber of &vehicle journeys
ttlBase: All respondents
val c(a0,a1);Base;=
+0;1;2+i
+3-5;6-9;10+
```

punch code substitution

axes

l q41_1

ttlBrand awareness - 1st mention
ttlBase: All respondents
n01Washo;c=c411’1’
n01Clean-it;c=c412’1’
n01Squeezo;c=c413’1’
n01White-out;c=c414’1’

l q41_2
ttlBrand awareness - 2nd mention
ttlBase: All respondents
n01Washo;c=c411’2’
n01Clean-it;c=c412’2’
n01Squeezo;c=c413’2’
n01White-out;c=c414’2’

punch code substitution

- punch parameters can also have a name, eg. p
- in include file refer to ‘p’ , e.g.
  n01Brand A;c=c108’p’
- on include statement specify punch code to substitute:
  *include *incfile*;punch(p)=‘6’

punch code substitution

axes

```
l q41_1
*include q41.qin;ord=1st
+punch(p)='1'



l q41_2
*include q41.qin;ord=2nd
 +punch(p)='2'
```

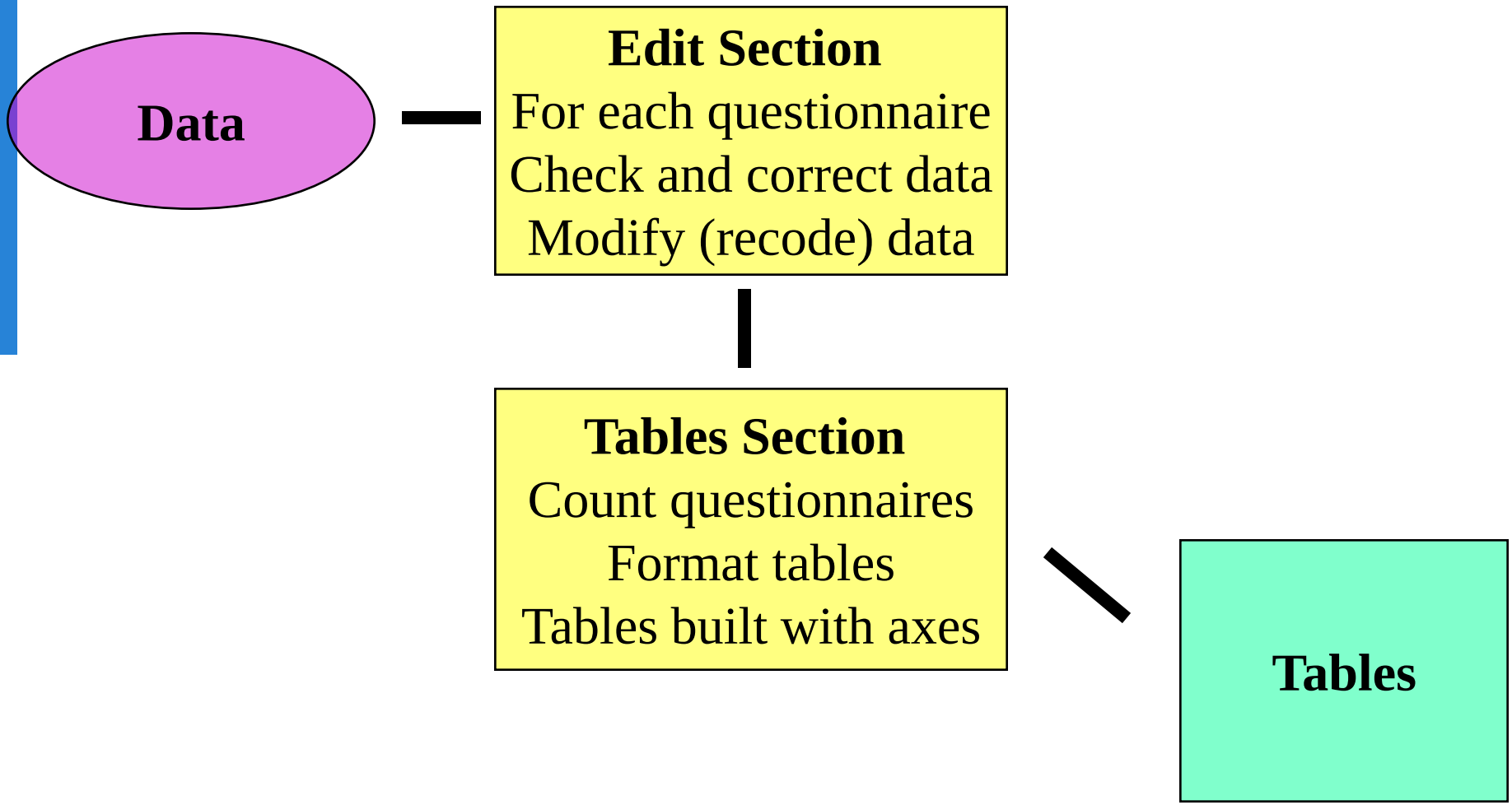
q41.qin

```
ttlBrand awareness - &ord mention
ttlBase: All respondents
n01Washo;c=c411'p'
n01Clean-it;c=c412'p'
n01Squeezo;c=c413'p'
n01White-out;c=c414'p'
```

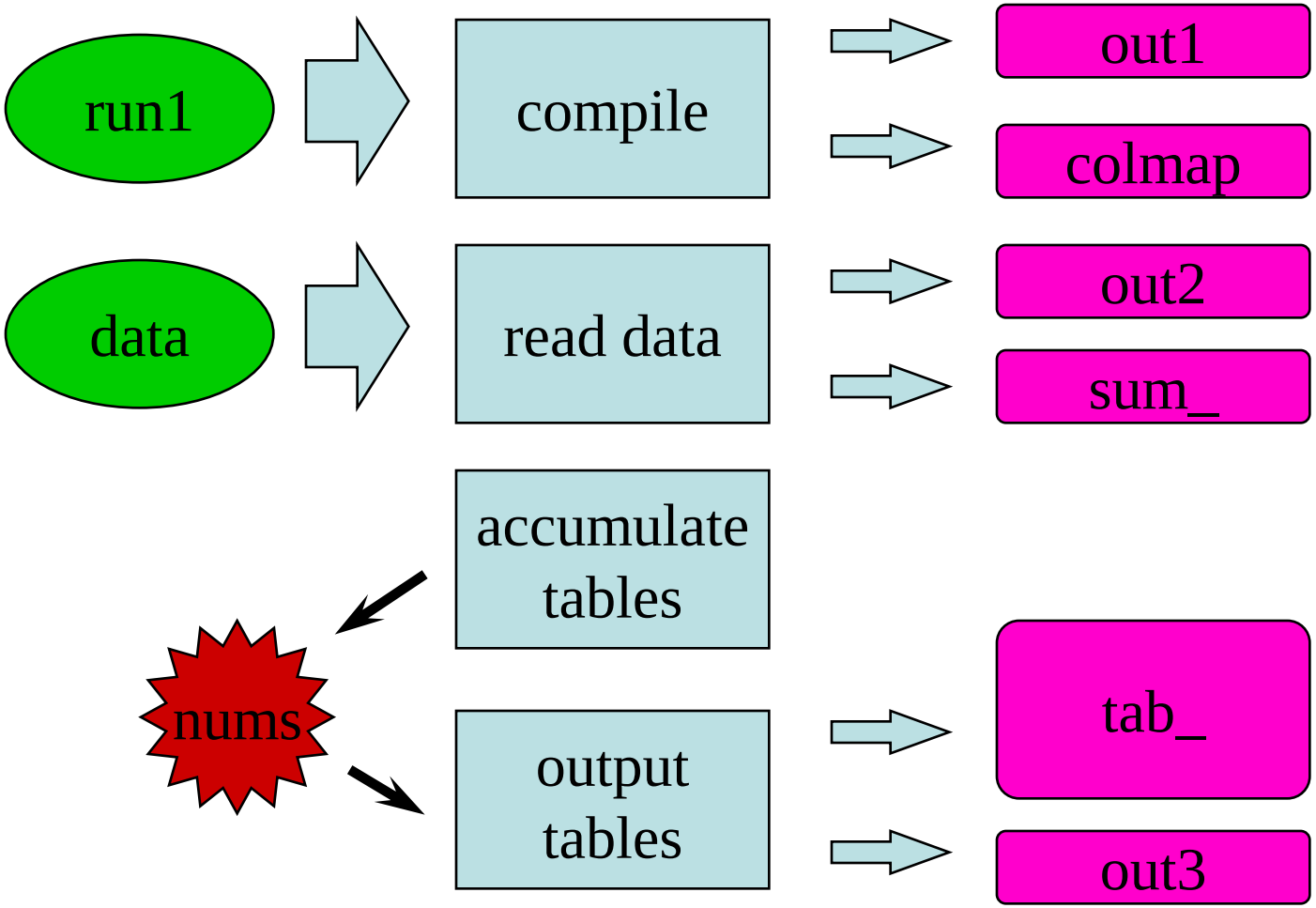
Exercise 7

The Edit Section

Stages in a Quantum Run



Running Quantum



The Edit Section

- Concerned with data examination, validation and recoding
- Achieved through the use of edit instructions
- Starts with the **ed** statement and ends with the **end** statement
- Appears before the tab section
- Comments are denoted by **/*** at the start of the line

Defining your own variables

- If deriving new information it is recommended you use named variables to improve readability
- Quantum automatically provides:
  - An integer array of 200 t variables
  - A real array of 100 x variables

Defining your own variables

- Format

  type name length s

  Where :

  type may be:

int	can contain a whole number
real	can contain numbers with a decimal place
data	can contain punch codes (like c)

  name is the name of the variable up to eight characters.

  length is the number required.

  s denotes that a subscript does not require parentheses (e.g. c1 and not c(1)).

Assignment of Values

- Most basic operation is assignment
  variable = expression
- Variable may be data, integer or real
- The contents of the variable are overwritten with the value of the expression

Assignment of Values

- Expressions may be constants e.g.

```
c115 = '1'
c(212,213) = $XX$
t100 = 52
x50 = 1.732
```

- Expressions may be another variable e.g.

```
c115 = c223
c(212,213) = c(217,218)
t1 = c(101,104)
x50 = cx(115,120)
```

Assignment of Values

- Expressions may be an arithmetic expression which combines variables and constants with the operators:

+	addition
-	subtraction
*	multiplication
/	division
()	denote order of evaluation (default is * and /, then + and -)

Examples:

```
t1 = c(112,113) * 4
x1 = t21 / 3.0 + c(115,116)
```

- When assigning an operation to a real variable make at least one element of the right-hand side real.

Special Cases in Assignment

- When assigning punches to a single column, the = sign may be excluded:

```
c15'1'
```

- When setting a series of columns blank only one space is necessary, regardless of the number of columns

```
c(45,73)=$ $
```

- When assigning a string to a series of columns, if the string is shorter than the column series it will be right justified. Leading columns will be set to blank

Combining column information

- A special assignment operator can be used to combine data variables
  Format:

  data-var = operator(data-variables)

- Valid operators are:

  or Copy codes present in **ANY** column

  and Copy codes present in **ALL** columns

  xor Copy code present in **ONE** column

  ONLY

Combining column information

- or

```
c182 = or(c137,c138,c139)
----+----4   …   ---8----+
        111             1
        222             2
        353             3
        477             4
                        5
                        7
```

c182 contains list of all codes present in AT LEAST ONE of the named columns

Combining column information

- **and**

```
c181 = and(c137,c138,c139)
----+----4    …    ---8----+
        111              1
        222              2
        353
        477
```

c182 contains list of all codes present in ALL of the named columns. Codes ‘3’ and ‘7’ appear in more than one column, but are not copied as they are not common to all.

Combining column information

- xor

```
c183 = xor(c137,c138,c139)
----+----4   …   ---8----+
        111             4
        222             5
        353
        477
```

Only 2 codes have been copied because all other codes appear in more than one column.

Emit and Delete

- Assignment overwrites previous contents of variable
- *emit* adds punches to data variables without overwriting their current values
- *delete* removes punches from data variables. Only the specified punches are affected.

Emit and Delete

- Format:

```
emit variable'p', variable'p'
delete variable'p', variable'p'
```

- Examples:

```
emit c310'1', c313'1', c359'1'
delete c219'234', c235'0'
```

Flow Control

- Each edit instruction is executed in turn for each record
- This can be made more selective by using flow control
- The *if* statement
- The *else* instruction
- The *goto* instruction

The if statement

- Format:

```
if (logical-expr) edit-instr
```

- If the logical expression is true the following instruction is executed
- Further instructions may be executed separated by semi-colons
- Example:

```
if (c115'1') c225'3'
```

The else instruction

- The *else* instruction follows an *if* statement. Any record that fails the *if* statement performs the action carried out after the *else*
- Long statements may be continued with a + in position 1
- Example:

```
if (c150'1') c350'3' ; c360'2'
+else; c(350,360)=$ $
```

The goto instruction

- Follows an *if* and specifies the number of a statement label to branch to.
- All intervening edit instructions are skipped.
- A statement label is a line in the edit starting with a number from 1 to 99999.
- Must be exclusive in the run, but need not appear in numeric order.

The goto instruction

- Example:

```
if (c15'1') goto 100
...
Edit instructions
...
100 continue
```

- Note the dummy statement *continue* which is usually used on statement labels

Leaving the Edit

- Each record is passed from the edit to the tab section when it reaches the end statement
- The instruction

```
reject;return
```

will finish editing a record and not pass it to the tab section

Examining Data

- The ***count*** instruction produces a hole-count
- The ***list*** instruction produces frequency distributions of fields
- The ***write*** statement will print a card image of the record to the out2 file

Hole Counts

- Format:

```
count columns $Text$
```

- Where ***columns*** are the columns to be counted in the form c(m,n)
- Example:

```
count c(1,80) $Job Title$
```

- The hole count is written to the file hct_

Frequency Distributions

- Format:

```
list columns $Text$
```

- Where ***columns*** is the field to be examined
- Example:

```
list c(10,13) $Q.13$
```

- The list is written to the file lst_
- The instruction ***lista*** will produce just the alphabetic list
- The instruction ***listr*** will produce just the ranked list

Displaying Individual Records

- Format:

```
write $Text$
```

- A card image of the entire record is written to the file out2

Exercise 8

Useful Functions

Examining Numeric Data Using Range

- Format:

  range(m,n,l,h)

- Where:

  m is the start of the field
  n is the end of the field
  l is the low value of the field
  h is the high value of the field

Examining Numeric Data Using Range

- range is a logical expression. It is true if the specified field is within the specified low and high values
- Examples:

```
if (range(10,11,1,99)) goto 100
c=range(115,117,100,115)
r (range(100,114,1,99999)) $Q5 Invalid values$
```

- If leading blanks are permitted use **rangeb**

Examining Numeric/String Fields With .in.

- Format:

  c(m,n).in.(value-list)

- Where value-list may be:
  - Individual values separated by commas
  - Ranges of values separated by colons
  - Strings enclosed in dollars

Examining Numeric/String Fields With .in.

- This is a logical expression. It is true if the field of columns contains an item in the value list
- Examples:

```
if (c(110,113).in.(1:20,25:55,99,$&&$))
goto 100
c=c(250,251).in.(35:55,99)
```

- More flexible than range in allowing for dicontinuous values and strings

Testing the Number of Punches

- Format:
  numb(column-list)
- Where column-list is one or more columns (and optionally punches)
- Returns the number of punches found in the specified column(s) and punches
- Is tested against a value and used as part of a logical expression

Priority

- Used to force multi-punched data to be single-punched
- Define the order of importance or priority of codes for a column or series of columns
- If more than one code is present the most important code is kept and the rest is deleted

Priority

- Format:
  priority var'p','p1','p2',…
- Where:
  var is a data variable
  'p','p1','p2' etc. are the codes
- Examples:

```
priority c115'3','2','4','1','5'

priority c225'1','2','3','4','5',
+c226'1','2','3'
```

Exercise 9