Version 1.0: 0605

AQA

ASSESSMENT and QUALIFICATIONS ALLIANCE

General Certificate of Education

Computing 5511/6511

CPT4 Processing and Programming Techniques

Mark Scheme

2005 examination - June series

Mark schemes are prepared by the Principal Examiner and considered, together with the relevant questions, by a panel of subject teachers. This mark scheme includes any amendments made at the standardisation meeting attended by all examiners and is the scheme which was used by them in this examination. The standardisation meeting ensures that the mark scheme covers the candidates' responses to questions and that every examiner understands and applies it in the same correct way. As preparation for the standardisation meeting each examiner analyses a number of candidates' scripts: alternative answers not already covered by the mark scheme are discussed at the meeting and legislated for. If, after this meeting, examiners encounter unusual answers which have not been discussed at the meeting they are required to refer these to the Principal Examiner.

It must be stressed that a mark scheme is a working document, in many cases further developed and expanded on the basis of candidates' reactions to a particular paper. Assumptions about future mark schemes on the basis of one year's document should be avoided; whilst the guiding principles of assessment remain constant, details will change, depending on the content of a particular examination paper.

Instructions to examiners

The following forms of notation should be used on candidates' scripts:

- **Ticks - To indicate what is accepted as correct or creditworthy, placed in the body of the answer, and on diagrams;**

- **Underscoring – To identify errors/irrelevance in written answers;**

- **Crosses – to indicate a wrong answer;**

- **Brief comments – placed in at suitable points in the body of the text to amplify the marking;**

- **BOD – means benefit of the doubt and is used where the candidate's answer has been given a mark on the balance of probabilities that the candidate's answer has met the requirements of the mark scheme even though it could be interpreted differently;**

- **NE – means not enough and is applied to an answer that falls short of what is required;**

- **O/S – means outside the mark scheme. The candidate's answer is creditworthy but the answer does not match any of the answers on the mark scheme for the particular question. Nevertheless a mark is awarded;**

- **C/F – means carried forward. This arises when a candidate offers an answer which is not creditworthy in one question but is creditworthy in a later question. The mark is carried forward to the question which is creditworthy;**

- **C/B – means carried back. This is similar to a carry forward but the mark is carried back to an earlier question.**

- **T/O – means talked out. If the candidate provided extra information which contradicts the correct answer or leaves it open to interpretation which is correct.**

- **^ means missing term or symbol.**

- **F/T – means followed through. If a candidate made a mistake in an earlier part of the answer.**

- **P1 – penalise once.**

- **I - ignore**

The following notation is used in the mark scheme

- **; - means a single mark;**
- **A. - means an acceptable creditworthy answer;**
- **R. – means reject answer as not creditworthy.**

1 Compare Pascal with middle item of list / Lisp;

Compare Pascal with middle item of upper sublist / Prolog;

Compare Pascal with Pascal // compare only item in this sublist to get a match; *3*

Lose 1 mark if Pascal not explicit in comparison

stop marking from time it goes wrong

OR

List[4] = Pascal? False; **A** [4] = Pascal **R** 4 = Pascal

List[6] = Pascal? False;

List[5] = Pascal? True; **3**

if formula explicit, follow through on formula

2 (a) 140 $^1/_4$;; *one mark for correct integer part,*

140.25;; *one mark for correct fractional part* *2*

(b) (i) -14.5;;; *give 2 marks for 14.5*

partial marks for workings if result incorrect:

1 mark for negative number;

1 mark for x2^4 (accept 16 instead of $2^{4)}$;

A *showing that binary point moves 4 places right;* *3*

(ii) leftmost 2 digits/bits are different;

a significant bit is stored after the binary point;

bit after point different from bit before point;

(negative number) starts with 10… (positive number starts with 01)….;

A the first bit after the sign bit is a ‘0’;

A The second bit is a ‘0’;

A *an answer that clearly implies a ‘0’ follows the ‘1’* max *1*

(iii) to maximise accuracy/precision for a given number of bits

// to minimise rounding errors;

A more accurate/precise for a given number of bits;

a given number can only be expressed in one way in a given number of bits

// a given number can only be expressed in one way in a given format;

to simplify arithmetic/logical operations; *1* **7**

I range

3 (a) 1101 0001 0101 1010;; *1 mark for each correct byte* *2*

I *leading bits*

(b) 16; *1* **3**

4 (a) a signal/command/message; **R** request

from a device/source seeking the attention of the processor;

or from a device/source to the processor; *2*

(b) finish the current fetch-execute cycle // finish execution of current instruction;

identify interrupt // type of interrupt is found // identify priority of interrupt;

disable lower priority interrupts;

save (contents of) registers // save volatile environment // save current status of machine;

source of interrupt provides vector address / offset;

offset added to base address;

to give start of ISR // vector address is used as start address of interrupt service routine;

Entry point *equivalent to*
Start address

jump to interrupt service routine

// PC/IR loaded with this address // interrupt serviced;

restore (contents of) registers // restore volatile environment;

enable interrupts; m*ax 5* **7**

R goes back to where it left off

steps must be in a sensible order. **I** *any steps out of order.*

5 *Penalise once if wrong case used*

(a) female(helen)./female("Helen").; *1*

(b) mother(M,X) IF female(M) AND parent (M,X) *must use M and X*

// mother(M,X) IF parent(M,X) AND NOT male(M); *1*

(c) (i) *sibling (S,X) IF* mother(M,S) AND mother(M,X);

AND father(F,S) AND father (F,X);

OR *sibling (S,X) IF* parent($\underline{Z}$,S) AND parent($\underline{Z}$,X)

AND parent($\underline{M}$,S) AND parent($\underline{M}$,X);

AND Z<>M // AND Z NOT M // AND Z≠M // AND NOT Z=M

A NotEqual (Z,M) *2*

(ii) AND S<>X; **A** S NOT X **A** S≠X **A** NOT S=X *1*

(d) $\underbrace{\text{grandfather(G,}\underline{\text{P}}\text{) IF}}_{1}\ \underbrace{\text{father(G,X)}}_{]}\ \underbrace{\text{AND parent(X,P).}}_{]}$

Must use P to represent the child

OR $\underbrace{\text{grandparent(G,}\underline{\text{P}}\text{) IF parent(G,X)}}_{1}\ \underbrace{\text{AND parent(X,P).}}_{1}$

$\underbrace{\text{grandfather(G,}\underline{\text{P}}\text{) IF male(G); AND grandparent(G,P).}}_{1}$

OR $\underbrace{\text{grandfather(G,}\underline{\text{P}}\text{) IF}}_{]}\ \underbrace{\text{male(G) AND parent(G,X)}}_{1}\ \underbrace{\text{AND parent(X,P)}}_{1}$ 3 **8**

6 (a) need to access/address machine registers / exact memory addresses / hardware directly;

fast speed of operation required

// assembler code runs faster than programs written in HLL;

code needs to take up little memory

// assembler code gives smaller object code programs;

A minimise the size of the program/code;

no compiler/interpreter exists yet for machine // no other translator exists;

R manipulate bits *max 2*

(b) A pseudocode statement gets no mark. Order of statements is important.

P1 *for using wrong columns*

Immediate addressing must be used for constants but could be indicated by annotation e.g. load register with 0. Indicate any omissions

If subtract is used in place of CMP check value of x is not destroyed.

Accept numbers as symbolic addresses. Accept relative addressing for jumps.

No marks for jumps not altering flow of control.

```
Start:   STORE #0,Count / store zero; in Count;

label1:  LOAD register, X/ label; load X ;

         ADD register, X              / add X to register ; A ASL register;

         STORE register, X            / save X ;

         INC Count or ADD Count,#1    / increment Count; A equivalent using registers

         CMP register, # 999          / compare ;

         BCS label1                   / conditional branch to correct label;

                                               if [AC]<999;                max 9
```

Or
Add X, X;;;
2 marks if no comment that says where result is stored

Alternative:

```
Start:   LDA #0                       / initialisation ;

         STA Count                    / store zero in Count ;

label1:  LDA X                        / label ; load X ;

         ADD X                        / add X to register ; A ASL register;

         STA X                        / save X 1 mark

         INC Count                    / increment ; A equivalent using registers

         CMP # 999                    / compare ;

         BCS label1                   / conditional branch; if [AC]<999 ;       max 9    11
```

Instead of ***BCS*** *allow an opcode such as* ***BGT*** *(Branch if Greater Than)*

BLT *(Branch if Less Than),* ***BPS*** *(Branch on Positive),* ***BGE*** *(Branch on Greater than or Equal) in the correct combination to give same meaning*

Alternative:

```
Start:   MOVE #0,Count  / store zero; in Count;

label1:  MOVE X, register                      / label; load X ;

         ADD register, X                       / add X to register ; A ASL register;

         Move register, X  / save X ;

         INC Count or ADD Count,#1             / increment Count; A equivalent using registers

         CMP register, # 999                   / compare ;

         BCS label1                            / conditional branch; if [AC]<999;          max 9
```

7 (a) a class has properties/fields/attributes/characteristics and methods/procedures/functions of the parent class it is derived from
// a subclass/derived class inherits all the properties/fields/attributes/characteristics and methods/procedures/functions from a super-class/base-class/parent class; *1*

(b) StockItem (=) Class // Class (=) StockItem; *1 mark for keywords Class and StockItem* (**A** Object *instead of* Class)

```
    Public (procedure DisplayDetails) (virtual)(virtual;abstract)
1        (procedure) SetLoan   (virtual)(virtual;abstract)
    Private;
         Title: String
         OnLoan: Boolean                                      1
         DateAcquired: String/DateTime/Date
    End
```

1 mark for keyword Public and correct methods

A protected

A text *instead of* string

1 mark for correct data fields & data types

don't allow the other fields

Book = Class (StockItem) // Class Book extends/derives from StockItem

// Book Subclass: StockItem; **A** *without keyword Class*

```
Private
     Author: String
     ISBN: String                 1
Public
     (Procedure) DisplayDetails          (override)     1
End
```

If candidate declared 'getters' and 'setters' for the base class fields then don't have to have DisplayDetails as a base class method

CD = Class (StockItem) // Class CD extends/derives from StockItem // CD Subclass: StockItem;

```
Private
     Artist: String
     PlayingTime: Integer/Real/Time/DateTime      1
Public
     (Procedure) DisplayDetails    (override)
End
```

No marks for a diagrammatic answer. **I** *method parameters* max 7

Java version:

```
Public Class StockItem

{

        Private String title;

        Private boolean onLoan;

        Private String dateAquired;

        Public void displayDetails ();

        Public void setLoan ();

}


Public Class Book extends StockItem

{

        Private string author;

        Private string isbn;

        Public void displayDetails ();

}


Public Class CD extends StockItem

{

        Private string artist;

        Private integer playingTime;

        Public void displayDetails();

}
```

8 (a) it calls itself / is defined in terms of itself

/ contains within its body a reference to itself; *1*

Ensure 'it' refers to procedure, if meaning program or object no mark

(b) the current state of the machine is saved/preserved;

so can return <u>correctly</u> (to previous invocation/call of **Process)**;

or

return address / procedure parameter / status register / other register values

/ local variables

must be saved/preserved; so can return <u>correctly</u> to ‘*correctly’ can be implied*

(previous invocation of Process); *2*

(c) Printed Output:

1; 3; 5, Bird; Bremner; 4, Fortune, Jones; 2, Smith; *6*

mark from left and stop marking when error encountered

ignore punctuation.

(d) (in-order) traversal of a tree; **A** printing of tree (elements in order) *1* **10**

I *wrong order*

9 *accept Immediate Access Store or IAS as an alternative to main memory*

(a) (i) hard disk / secondary memory; **R** backing store **R** hard drive 1

(ii) (used when execution of a program/process)

not enough <u>main</u> memory / RAM / physical memory;

to run process(es) // load program(s) // allocate data area; **A** store

R computer fooled into thinking more main memory *max 2*

(iii) physical/main memory/RAM is conceptually divided into a number of

(fixed size) page frames; **A** pages/segments;

(virtual address space of) program/process is divided into a number of (fixed size) pages;

page table indicates which pages of process are loaded (and where);

A page table keeps track of pages;

<u>pages</u> are swapped between disk/secondary memory and <u>main</u> memory as required

// <u>pages</u> are swapped into, and out of, main memory as required; *max2*

No marks for a point which references just data when it should reference a program

backing store is not acceptable as a substitute for disk

(b) (i) a thread is a line/path of execution within a process;

a thread is the processing performed on a single set of data in the system;

a thread is a process that shares most of its environment;

a thread is a light-weight sub-process; max 1

(ii) each thread has its own thread control block / TCB; **A** thread ID;

it has its own program counter / stack pointer / register values / local variables max 1

(iii) threads have access to the same memory so they can communicate easily;

saves main memory / RAM;

faster execution than separate processes // faster execution overall // faster start-up time;

Allow carry forward/back between (i) and (ii) max 1 **8**

R *program instead of process*