RHODES UNIVERSITY

Computer Science 301 - 2000 - Programming Language Translation

Well, here you are. Here is the free information you have all been waiting for, with some extra bits of advice:

- Don't panic. It is probably easier than you might at first think.
- The solution to section B does not need an enormous amount of code. But it needs careful thought. I am looking for evidence of mature solutions, not crude hacks.
- Do make sure you get a good night's sleep!

How to spend a Special Sunday

From now until two hours before the examination tomorrow, Computer Science 3 students have exclusive use of the Braae Laboratory. You are encouraged to throw out anyone else who tries to use it, but we hope that this does not need to happen.

During this time you will be able to find the following files in the usual places, or by following links from the course web page:

- All the files as were made available for the practical course.
- The files `EXAMC.ZIP`, `exam.tgz` and `EXAMP.ZIP` which contain C++ and Pascal versions of the Coco/R system, and the sources for generating an "extended" Topsy compiler, as in the model solution for Prac 26 this year. If you unzip the appropriate file you will be able to run QEdit and Coco/R using your Favourite Language in the way that is, hopefully, familiar by now.
- The Turbo Pascal and Borland and Visual C++ compilers, which you can invoke in the usual way. You will not need to log on to a server to use these compilers; they are on the local C: drive.
- The entire set of source files for The Book, in a directory system starting at `I:CSC301\TRANS\SOURCES`, unpacked as described in Appendix A. The files are arranged by chapter and by language - for example, the C++ versions of the sources for chapter 18 are in `I:CSC301\TRANS\SOURCES\CHAP18\CPP`. If you have trouble locating these files you are free to ask for help (but the information is all in the file `I:CSC301\TRANS\SOURCES\README.SRC`).
- Notes on enumeration types from a Modula-2 text book.

Once you have copied the "exam kit" it is suggested that you reboot the machines into the "local" mode that you will use tomorrow. To do this you shut down, and then choose the "NONET" configuration as the system reboots. When the login screen reappears you log in with the username `test1` and the password `rbtest1`. This works on all the machines; none of them have access to one another or to the network.

Today you may use the files and the systems in any way that you wish subject to the following restrictions: *Please observe these in the interests of everyone else in the class.*

(a) Create a working directory on D: in the usual way, and preferably do all your work within this directory.

(b) When you have finished working, **please** delete your files from the D: directory, so that others are not tempted to come and snoop around to steal ideas from you.

(c) Since tomorrow you will not have access to the file server, work on the D: drive and not on your server file space, for practice, if for no other reason!

(d) You are encouraged to discuss the problem with one another, and with anybody not on the "prohibited" list.

(e) You are also free to consult books in the library. If you cannot find a book that you are looking for, it may well be the case that there is a copy in the Department. Feel free to ask.

I suggest that you DO spend some of the next 24 hours in DISCUSSION with one another, and some of the time in actually TRYING OUT your ideas. You have plenty of time in which to prepare and test really good solutions - go for it and good luck! Remember that you may not bring any papers or diskettes into the exam room tomorrow.

If you cannot unpack the files, or have trouble getting the familiar tools to work (unlikely!), you may ask me for help. You may also ask for explanation of any points in the question that you do not understand, in the same way that you are allowed to ask before the start of an ordinary examination. You are no longer allowed to ask me questions about any other part of the course. Sorry; you had your chance earlier, and I cannot allow this without risking the chance of sneak questions and suggestions being called for.

If you cannot solve the problem completely, don't panic. It has been designed so that I can recognize that students have reached varying degrees of sophistication and understanding.

How you will spend a Merry Monday

Just before the start of the formal examinations the laboratory will be unavailable. During that time

- The D: and C:\TEMP drives will be cleared.
- The network connections will be disabled. **You will not be able to use a Unix system**

At the start of each examination session:

- You will receive ordinary answer books for the preparation of answers, and an examination paper.
- You will be supplied with a diskette on which you will find the exam kit. This will contain the Coco/R system, as was made available on the previous day, and also various machine readable parts of the examination paper itself, in files with names like Q7.TXT (Question 7)
- A few copies of Chapter 12 of the text (running Coco/R) will be available as free information, should you need them.
- At the end of the exam you will be given a chance to copy any files that you have edited or created back to the diskette. It is, obviously, in your own interest to make sure that you know how to copy files correctly.

And for my last trick -

Section B [85 marks]

Please note that there is no obligation to produce a machine readable solution for this section. Coco/R and other files are provided so that you can enhance, refine, or test your solution if you desire. If you choose to produce a machine readable solution, you should create a working directory, unpack EXAMP.ZIP (Pascal) or EXAMC.ZIP (C++), modify any files that you like, and then copy all the files back to the blank diskette that will be provided.

Most computer languages provide simple, familiar, notations for handling arithmetic, character and Boolean types of data. Variables, structures and arrays can be declared of these basic types; they may be passed from one routine to another as parameters, and so on.

Some languages, notably Pascal, Modula-2, C, C++, and Ada, allow programmers the flexibility to define what are often known as *enumeration types*, or simply *enumerations*. Here are some examples to remind you of this idea:

```
TYPE  (* Pascal or Modula-2 *)
  COLOURS = ( Red, Orange, Yellow, Green, Blue, Indigo, Violet );
  INSTRUMENTS = ( Drum, Bass, Guitar, Trumpet, Trombone, Saxophone, Bagpipe );
VAR
  Walls, Ceiling, Roof : COLOURS;
  JazzBand : ARRAY [0 .. 40] OF INSTRUMENTS;
```

or the equivalent

```
typedef  /* C or C++ */
  enum { Red, Orange, Yellow, Green, Blue, Indigo, Violet } COLOURS;
typedef
  enum { Drum, Bass, Guitar, Trumpet, Trombone, Saxophone, Bagpipe } INSTRUMENTS;
COLOURS Walls, Ceiling, Roof;
INSTRUMENTS  JazzBand[41];
```

Sometimes the variables are declared directly in terms of the enumerations:

```
VAR  (* Pascal or Modula-2 *)
  CarHireFleet : ARRAY [1 .. 100] OF ( Golf, Tazz, Sierra, BMW316 );

enum CARS { Golf, Tazz, Sierra, BMW316 } CarHireFleet[101];  /* C or C++ */
```

The big idea here is to introduce a distinct, usually rather small, set of values which a variable can legitimately be assigned. Internally these values are represented by small integers - in the case of the `CarHireFleet` example the "value" of `Golf` would be 0, the value of `Tazz` would be 1, the value of `Sierra` would be 2, and so on.

In the C/C++ development of this idea the enumeration, in fact, results in nothing more than the creation of an implicit list of `const int` declarations. Thus the code

```
enum CARS { Golf, Tazz, Sierra, BMW316 } CarHireFleet[101];
```

is semantically completely equivalent to

```
const int Golf = 0; const int Tazz = 1;
const int Sierra = 2, const int BMW316 = 3;
int CarHireFleet[101];
```

and to all intents and purposes this gains very little, other than possible readability; an assignment like

```
CarHireFleet[N] = Tazz;
```

might mean more to a reader than the semantically identical

```
CarHireFleet[N] = 1;
```

In the much more rigorous Pascal and Modula-2 approach one would not be allowed this freedom; one would be forced to write

```
CarHireFleet[N] := Tazz;
```

Furthermore, whereas in C/C++ one could write code with rather dubious meaning like

```
CarHireFleet[4] = 45;            /* Even though 45 does not correspond to any known car! */
CarHireFleet[1] = Tazz / Sierra; /* Oh come, come! */
Walls = Sierra;                  /* Whatever turns you on is allowed in C++ */
```

in Pascal and Modula-2 one cannot perform arithmetic on variables of these types directly, or assign values of one type to variables of an explicitly different type, or assign values that are completely out of range. In short, the idea is to promote "safe" programming - if variables can meaningfully only assume one of a small set of values, the compiler should prevent the programmer from writing meaningless statements.

Clearly there are some operations that could have sensible meaning. Looping and comparison statements like

```
if (Walls == Indigo) Redecorate(Blue);
```

or

```
for (Roof = Red; Roof <= Violet; Roof++) DiscussWithNeighbours(Roof);
```

or

```
            if (Jazzband[N] >= Saxophone) Shoot(JazzBand[N]);
```

might reasonably be thought to make perfect sense - and would be easy to "implement" in terms of the underlying integer values.

In fact, the idea of a limited enumeration is already embodied in the standard character and Boolean types - type Boolean is really the enumeration of the values {`0, 1`} identified as {`false, true`}, although this type is so common that the programmer is not required to declare the type explicitly. Similarly, the character type is really an enumeration of a sequence of (typically) ASCII codes, and so on.

Although Pascal and Modula-2 forbid programmers from abusing variables and constants of any enumeration types that they might declare, the idea of "casting" allows them to bypass the security where necessary. The standard function `ORD(x)` can be applied to a value of an enumeration type to do nothing more than cheat the compiler into extracting the underlying integral value. This, and the inverse operation of cheating the compiler into thinking that it is dealing with a user-defined value when you want to map it from an integer are exemplified by code like

```
            IF (ORD(Bagpipe) > 4) THEN .....
            Roof := COLOURS(I + 5);
```

Rather annoyingly, in Pascal and Modula-2 one cannot `READ` and `WRITE` values of enumeration types directly - one has to use these casting functions to achieve the desired effects.

Enumerations are a "luxury" - clearly they are not really *needed*, as all they provide is a slightly safer way of programming with small integers. Not surprisingly, therefore, they are not found in languages like Java (simplified from C++) or Oberon (simplified from Modula-2).

During this course you have studied and extended a compiler for a small language, Topsy, which has syntax similar to C++, but in the implementation of which we have repeatedly stressed the ideas and merits of safe programming. In the examination "kit" you will find the tools we have used to develop this compiler, namely a C++ or Pascal version of the Coco/R compiler generator, frame files, the attributed grammar for Topsy, and the support modules for the symbol table handler, code generator and interpreter for the version of Topsy as it was extended during the final stages of the laboratory work.

How would you add the ability to define enumeration types in Topsy programs and to implement these types, at the same time providing safeguards to ensure that they could not be abused? It will suffice to restrict your answer to use a syntax where the variables are declared directly in terms of the enumerations, that is, do not try to handle the `typedef` (or `TYPE`) form of declaration.

A completely detailed solution to this reasonably large exercise might take the form of a complete set of attribute grammar and supporting module source files, and it is highly likely that you cannot provide these in full in the time available in the examination session. However, in the 24 hours available before the formal examination period, by careful study of the example Topsy programs in the exam kit, and taking advantage of the opportunity to discuss your approach with other members of the class, you should be able to get a long way towards making this little language "absolutely perfect". (As I said early one Friday morning - any language would be absolutely perfect if it had just one more feature!)

The examiners will be looking for evidence that you are familiar with the use of Cocol, symbol table manipulation, type and range checking, and error detection. During the formal examination period you would be advised to concentrate simply on describing the changes and alterations to the attribute grammar and support files in as much detail as time permits, preferably by providing selected and appropriate sections of code. Listings of the attribute grammar will be available to candidates who require them.

You may wish to read up a little more on enumeration types as they are used in languages like Modula-2. An essay on these can be found on the course WWW page by following a fairly obvious link.

A typical test program in the kit reads:

```
void main (void) { // exam.top
// Illustrate some simple enumeration types in extended Topsy++
// Some valid declarations
  enum DAYS { Mon, Tues, Wed, Thurs, Fri, Sat, Sun } Today, Yesterday;
  enum WORKERS { BlueCollar, WhiteCollar, Manager, Boss } Staff[12];
  int i, j, k, PayPacket[12];
  const pay = 100;
  bool rich;
// Some invalid declarations - your system should be able to detect these
  enum DEGREE { BSc, BA, BCom, MSc, PhD };              // No variables declared
  enum FRUIT { Orange, Pear, Banana, Grape } Favourite;  // This is okay
  enum COLOURS { Red, Orange, Green } Paint;            // Orange not unique
// Some potentially sensible statements
  Today = Tues;
  Yesterday = Mon;                                      // That follows!
  if (Today < Yesterday) cout << "Compiler error";      // Should not occur
  Today++;                                              // Working past midnight?
  if (Today != Wed) cout << "another compiler error";
  int totalPay = 0;
  for (Today = Mon; Today <= Fri; Today++) totalPay = totalPay + pay;
  for (Today = Sat; Today <= Sun; Today++) totalPay = totalPay + 2 * pay;
  rich = Staff[i] > Manager;
  Yesterday = DAYS(int(Today) - 1);  // unless Today is Mon
// Some meaningless statements - your system should be able to detect these
  Sun++;                 // Cannot increment a constant
  Today = Sun; Today++;  // There is no day past Sun
  if (Today == 4)        // Invalid comparison - type incompatibility
    Staff[1] = rich;     // Invalid assignment - type incompatibility
  Manager = Boss;        // Cannot assign to a constant
  PayPacket[Boss] = 1000; // Incompatible subscript type
}
```

Examination period allocations

The morning session will run from 08h30 until 11h30. Candidates must be in the Struben Building from 08h15, and will not be allowed to leave before 11h30.

The afternoon session will run 12h00 until 15h00. Candidates must be in the Struben Building from 11h15, as we have to make sure that there is no collaboration between sessions.

Morning	698C6270	Carter, LS
Afternoon	69730904	Chari, DP
Morning	698D3523	Daya, PJ
Morning	698D3070	Dickson, BJ
Afternoon	698E1944	Emmenes, Q
Afternoon	69610658	Erfani-Ghadimi, N
Afternoon	698H1711	Hartley, CG
Afternoon	698H3690	Hitchcock, JD
Afternoon	695H7571	Holose, MM
Afternoon	69730230	Howis, S
Afternoon	698J6219	Jacot-Guillarmod, PF
Afternoon	698J6035	Johnson, RD
Afternoon	698J4312	Jones, EB
Morning	697K5335	Kao, MN
Morning	698K4561	Kavuma, I
Afternoon	698K6080	Kulesza, K
Morning	69730402	Lalloo, A
Morning	69610870	Louw, JA
Morning	698M3440	Madhoo, V
Afternoon	69730153	Makaya, V
Morning	698M2452	Marx, IB
Morning	69610537	Masekoameng, RP
Morning	69731187	Mfenguza, N
Afternoon	69731732	Miler, V
Morning	698M1394	Motsoeneng, TP
Afternoon	698M3016	Mutagahywa, RN
Morning	698N1103	Naude, R
Morning	69610159	Ndlangisa, M
Afternoon	69730217	Ngoasheng, KJ
Morning	698N4624	Noudehou, FSM
Afternoon	698O6331	Ocker, DHC
Morning	698P1200	Palmer, MN
Morning	698P1836	Parry, DC
Afternoon	69750001	Patel, AS
Morning	698P1642	Paterson, AI
Afternoon	69760003	Price, RE
Afternoon	698R3067	Renwick, MR
Afternoon	698R6473	Ridderhof, MJ
Afternoon	698R1724	Riordan, DD
Morning	697R6228	Roberts, AW
Morning	698R1477	Roberts, K
Afternoon	698S6239	Stavrakis, EA
Morning	698S1317	Stevens, BR
Afternoon	697S5130	Swales, D
Morning	698T1375	Tankard, GM
Morning	698T3747	Traas, GRL
Morning	698T4414	Tsegaye, MA
Afternoon	69640121	Twala, MEN
Afternoon	698U6039	Urban, PA
Morning	69610531	Walwyn, GA
Afternoon	698W1548	Wells, TJ
Morning	698W4075	Wright, LA
Afternoon	693W5688	Wright, MK
Morning	698Y2081	Yates, SC

Test programs

The exam kits contain a selection of silly Topsy programs which you may find useful in testing the modifications you make. In particular look at the ones below. They are not all "correct" of course.

You will also find an executable `TEST.EXE` derived from my model solution to this exercise. Rather cruelly, it has had all the debugging information suppressed; don't bother to try anything like reverse engineering this!

A command like

```
TEST t0.top
```

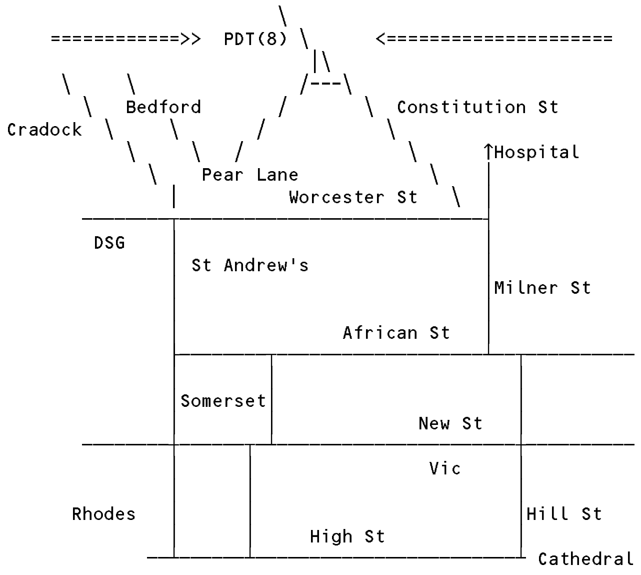
will attempt to compiler the file `t0.top`.

```
$D+ // Turn diagnostic mode on for testing the compiler - t0.top
void main (void) { // Declarations only
  enum MyType { a, b, c } X, Y, Z[4];
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t1.top
void main (void) { // Declarations and initialisation
  enum MyType { a, b, c } X = a, Y = b, Z[4];
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t2.top
void main (void) { // Several enumerations
  enum MyType { a, b } X, Y, Z[4];
  enum YourType { p, q } A;
  enum SillyType { LonelyValue } Ace;
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t3.top
void main (void) { // Some bad enumeration declarations
  enum MyType { a, b, c } X, Y, Z[4];
  enum YourType { a, p, q, r } A;
  enum HisType { e1, e2, e3 };
  enum MyType { my1, my2 };
  enum YourType { d1, , } DD;
  enum HerType { f1 f2 } FF;
  enum ItsType { } it;
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t4.top
void main (void) { // Declarations and invalid assignments
  enum MyType { a, b, c } X, Y, Z[4];
  int i;
  i = MyType;  // invalid
  i = X;       // invalid
  X = i;       // invalid
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t5.top
void main (void) { // Declarations and invalid input/output
  enum MyType { a, b, c } X, Y, Z[4];
  cin >> MyType;   // invalid
  cin >> X;        // invalid
  cout << X;       // invalid
  cout << a;       // invalid
}

============================================================================
$D+ // Turn diagnostic mode on for testing the compiler - t6.top
void main (void) { // Simple for loops
  enum MyType { a, b, c } X, Y, Z[4];
  int i = 1;
  for (X = a; X <= c; X++) { cout << i; i++; }
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t7.top
void main (void) { // For loop going backwards
  enum MyType { a, b, c } X, Y, Z[4];
  int i = 2;
  for (X = c; X >= a; X--) { cout << i; i--; }
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t8.top
void main (void) { // Simple while loop
  enum MyType { a, b, c } X, Y, Z[4];
  int i = 1;
  X = a; while (X <= c) { cout << i; i++; X++; }
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t9.top
void main (void) { // Simple casting
  enum MyType { a, b, c } X, Y, Z[4];
  int i = 0;
  for (X = a; X <= c; X++) { cout << i << int(X); i++; }
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t10.top
void main (void) { // Casting between types
  enum MyType { a, b, c } X, Y, Z[4];
  int i = int(a);
  X = val(MyType, 2);
  X = MyType(1);
  bool value = bool(0);
  value = bool(a);
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t11.top
void main (void) { // Casting between types, range errors
  enum MyType { a, b, c } X, Y, Z[4];
  X = val(MyType, -2);
  X = MyType(4);
  bool Bad = bool(c);
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t12.top
void main (void) { // Compound type casting
  enum MyType { a, b, c } X, Y, Z[4];
  int i = 0;
  for (X = a; X <= c; X++) { cout << i << char(int(X) + 65); i++; }
}

============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t13.top
void main (void) { // Relational operations
  enum MyType { a, b, c } X, Y, Z[4];
  if (a < c) cout << "a < c\n";
  if (c < a) cout << "compiler error";
  bool okay = a == a;
  okay = a > c;
  okay = int(c) >= 2;
}

============================================================================
$D+ // Turn diagnostic mode on for testing the compiler - t14.top
void main (void) { // Relational operations - illegal
  enum MyType { a, b, c } X, Y, Z[4];
  if (a < 4) cout << "a < 4\n";
  bool okay = a == char(a);
  okay = X < 4;
}

=============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t15.top
void main (void) { // Several enumerations and scopes
  enum MyType { a, b } X, Y, Z[4];
  enum YourType { p, q } A;
  // now start another block and scope
  { enum MyType { a, b } X, Y, Z[4];
    enum YourType { p, q } A;
  }
}

=============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t16.top
void main (void) { // Some bad enumeration declarations
  enum MyType { a, b, c } X, Y, Z[4];
  enum YourType { a, p, q, r } A;
  enum HisType { e1, e2, e3 };
  enum YourType { d1, d2, d3 } DD;
}

=============================================================================

$D+ // Turn diagnostic mode on for testing the compiler - t17.top

void main (void) { // Declarations with casting (bad)
  enum MyType { a, b, c } X = MyType(1), Y = MyType(5), Z[4];
}
```

Cessation of Hostilities Party

As previously agreed, Sally and I would like to invite you to an informal end of course party at our house on 13 November. There's a map below to help you find your way there. It would help if you could let me know whether you are coming so that we can borrow enough glasses, plates etc. E-mail to p.terry@ru.ac.za.

Time: from 18h30 onwards. Dress: Casual



PDT(8)
Bedford
Constitution St
Cradock
Hospital
Pear Lane
Worcester St
DSG
St Andrew's
Milner St
African St
Somerset
New St
Vic
Rhodes
Hill St
High St
Cathedral