Utilities Template B/utilt

Purpose

Miscellaneous utility routines for some fundamental I6 needs.

§1. Saying Phrases.

```
[ SayPhraseName closure;
    if (closure == 0) print "nothing";
    else print (string) closure-->2;
];
```

§2. Kinds.

```
[ KindAtomic kind;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return kind;
    return kind-->0;
];
[ KindBaseArity kind;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return 0;
    return kind-->1;
];
[ KindBaseTerm kind n;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return UNKNOWN_TY;
    return kind-->(2+n);
];
```

§3. DigitToValue.

This takes a ZSCII or Unicode character code and returns its value as a digit, or returns −1 if it isn't a digit.

```
[ DigitToValue c n;
    n = c-'0';
    if ((n<0) || (n>9)) return -1;
    return n;
];
```

§4. GenerateRandomNumber. The following uses the virtual machine's RNG (via the I6 built-in function `random`) to produce a uniformly random integer in the range n to m inclusive, where n and m are allowed to be either way around; so that a random number between 17 and 4 is the same thing as a random number between 4 and 17, and there is therefore no pair of n and m corresponding to an empty range of values.

```
[ GenerateRandomNumber n m s;
    if (n==m) return n;
    if (n>m) { s = n; n = m; m = s; }
    n--;
    return random(m-n) + n;
];
Constant R_DecimalNumber = GenerateRandomNumber;
Constant R_PrintTimeOfDay = GenerateRandomNumber;
```

§5. GroupChildren. The following runs through the child-objects of `par` in the I6 object tree, and moves those having a given property value together, to become the eldest children. It preserves the ordering in between those objects, and also in between those not having that property value. The property is required to be a common property.

We do this by temporarily moving objects into and out of `in_obj` and `out_obj`, objects which in all other circumstances never have children in the tree.

```
[ GroupChildren par prop value;
    while (child(par) ~= 0) {
        if (child(par).prop ~= value) move child(par) to out_obj;
        else move child(par) to in_obj;
    }
    while (child(in_obj) ~= 0)  move child(in_obj) to par;
    while (child(out_obj) ~= 0) move child(out_obj) to par;
    return child(par);
];
```

§6. PrintSpaces. Which prints a row of n spaces, for $n \geq 0$.

```
[ PrintSpaces n;
    while (n > 0) {
        print " ";
        n = n - 1;
    }
];
```

§**7. RunRoutines.** This function may not be very well-named, but the idea is to take a property of a given object and either to print it (and return `true`) if it's a string, and call it (and pass along its return value) if it's a routine. If the object does not provide the property, we act on the default value for the property if it has one, and otherwise do nothing (and return `false`).

The I6 pseudo-object `thedark` is used to give the impression that Darkness is a room in its own right, which is not really true. Note that it is not permitted to have properties other than the three named here: all other properties are redirected to the current location's object.

Properties with numbers under `INDIV_PROP_START` are "common properties". These come along with a table of default values, so that it is meaningful (in I6, anyway) to read them even when they are not provided (so that the address, returned by the `.&` operator, is zero).

```
[ RunRoutines obj prop;
    if (obj == thedark) obj = real_location;
    if ((obj.&prop == 0) && (prop >= INDIV_PROP_START)) rfalse;
    return obj.prop();
];
```

§**8. SwapWorkflags.** Recall that we have two general-purpose temporary attributes for each object: `workflag` and `workflag2`. The following swaps their values over for every object at once.

```
[ SwapWorkflags obj lst;
    objectloop (obj ofclass Object) {
        lst = false;
        if (obj has workflag2) lst = true;
        give obj ~workflag2;
        if (obj has workflag) give obj workflag2;
        give obj ~workflag;
        if (lst) give obj workflag;
    }
];
```

§**9. TestUseOption.** This routine, compiled by NI, returns `true` if the supplied argument is the number of a use option in force for the current run of NI, and `false` otherwise.

```
{-routine:Config::Inclusions::UseOptions::TestUseOption}
```

§**10. IntegerDivide.** We can't simply use I6's `/` operator, as that translates directly into a virtual machine opcode which crashes on divide by zero.

```
[ IntegerDivide A B;
    if (B == 0) { RunTimeProblem(RTP_DIVZERO); rfalse; }
    return A/B;
];
```

§**11. IntegerRemainder.** Similarly.

```
[ IntegerRemainder A B;
    if (B == 0) { RunTimeProblem(RTP_DIVZERO); rfalse; }
    return A%B;
];
```

§12. UnsignedCompare. Comparison of I6 integers is normally signed, that is, treating the word as a twos-complement signed number, so that `$FFFF` is less than `0`, for instance. If we want to construe words as being unsigned integers, or as addresses, we need to compare them with the following routine, which returns 1 if $x > y$, 0 if $x = y$ and -1 if $x < y$.

```
[ UnsignedCompare x y u v;
    if (x == y) return 0;
    if (x < 0 && y >= 0) return 1;
    if (x >= 0 && y < 0) return -1;
    u = x&~WORD_HIGHBIT; v= y&~WORD_HIGHBIT;
    if (u > v) return 1;
    return -1;
];
```

§13. ZRegion. I7 contains many relics from I6, but here's a relic from I5: a routine which used to determine the metaclass of a value, before that concept was given a name. In I6 code, it can be implemented simply using `metaclass`, as the following shows. (The name is from "region of the Z-machine".)

```
[ ZRegion addr;
    switch (metaclass(addr)) {
        nothing: return 0;
        Object, Class: return 1;
        Routine: return 2;
        String: return 3;
    }
];
```

§14. Library Messages. This is another fossil, and probably soon to change.

```
[ GL__M a b c d;
    if ((actor ~= player) || (untouchable_silence)) rtrue;
    return L__M(a,b,c,d); ];
[ AGL__M a b c d;
    if (untouchable_silence) rtrue;
    return L__M(a,b,c,d); ];
```