Optimising Compilers

Many modern architectures have provision for only 32-bit values in registers. However, ANSI C requires code such as

```c
extern void g(int);
extern void f(int x)
{
  char c = x;
  c += 1;
  g(c);
}
```

to have the effect that a call to f() causes g() to receive a parameter value as though c were stored in memory, i.e. in the range CHAR_MIN to CHAR_MAX. You may assume that char holds 8-bit values and the range is either -128..127 or 0..255.

While this example clearly requires range reduction following the incrementation of c, it is claimed that this is not always necessary.

One brutal technique to allocate char variables to registers is to treat them as int variables and achieve ANSI C conformance by range reduction just before each reference in the same manner that load-byte hardware might.

Develop an optimisation technique which might reduce the amount (static or dynamic) of such range reduction in code like:

```c
extern char p(int v[100])
{
  unsigned char r = 0;
  int i;
  for (i=0; i<100; i++)
    r = (r<<1 ^ r) + v[i];
  return (r & 128) != 0;
}
```

Little credit will be given for mere hand-compilation. [20 marks]

Hints: 1. Consider similarities to live variable analysis.
2. Consider whether register reads or writes occur more often.