```
**********************************************************
   78919 Thu Jun  6 06:55:35 2019
new/usr/src/contrib/zlib/deflate.c
11204 smatch issue in zlib/deflate.c
**********************************************************
______unchanged_portion_omitted_

 126 #ifdef FASTEST
 127 local const config configuration_table[2] = {
 128 /*      good lazy nice chain */
 129 /* 0 */ {0,    0,  0,    0, deflate_stored},  /* store only */
 130 /* 1 */ {4,    4,  8,    4, deflate_fast}}; /* max speed, no lazy matches */
 131 #else
 132 local const config configuration_table[10] = {
 133 /*      good lazy nice chain */
 134 /* 0 */ {0,    0,  0,    0, deflate_stored},  /* store only */
 135 /* 1 */ {4,    4,  8,    4, deflate_fast}, /* max speed, no lazy matches */
 136 /* 2 */ {4,    5, 16,    8, deflate_fast},
 137 /* 3 */ {4,    6, 32,   32, deflate_fast},

 139 /* 4 */ {4,    4, 16,   16, deflate_slow},  /* lazy matches */
 140 /* 5 */ {8,   16, 32,   32, deflate_slow},
 141 /* 6 */ {8,   16, 128, 128, deflate_slow},
 142 /* 7 */ {8,   32, 128, 256, deflate_slow},
 143 /* 8 */ {32, 128, 258, 1024, deflate_slow},
 144 /* 9 */ {32, 258, 258, 4096, deflate_slow}}; /* max compression */
 145 #endif

 147 /* Note: the deflate() code requires max_lazy >= MIN_MATCH and max_chain >= 4
 148  * For deflate_fast() (levels <= 3) good is ignored and lazy has a different
 149  * meaning.
 150  */

 152 /* rank Z_BLOCK between Z_NO_FLUSH and Z_PARTIAL_FLUSH */
 153 #define RANK(f) (((f) * 2) - ((f) > 4 ? 9 : 0))

 155 /* ===========================================================================
 156  * Update a hash value with the given input byte
 157  * IN  assertion: all calls to UPDATE_HASH are made with consecutive input
 158  *    characters, so that a running hash key can be computed from the previous
 159  *    key instead of complete recalculation each time.
 160  */
 161 #define UPDATE_HASH(s,h,c) (h = (((h)<<s->hash_shift) ^ (c)) & s->hash_mask)


 164 /* ===========================================================================
 165  * Insert string str in the dictionary and set match_head to the previous head
 166  * of the hash chain (the most recent string with same hash key). Return
 167  * the previous length of the hash chain.
 168  * If this file is compiled with -DFASTEST, the compression level is forced
 169  * to 1, and no hash chains are maintained.
 170  * IN  assertion: all calls to INSERT_STRING are made with consecutive input
 171  *    characters and the first MIN_MATCH bytes of str are valid (except for
 172  *    the last MIN_MATCH-1 bytes of the input file).
 173  */
 174 #ifdef FASTEST
 175 #define INSERT_STRING(s, str, match_head) \
 176    (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
 177     match_head = s->head[s->ins_h], \
 178     s->head[s->ins_h] = (Pos)(str))
 179 #else
 180 #define INSERT_STRING(s, str, match_head) \
 181    (UPDATE_HASH(s, s->ins_h, s->window[(str) + (MIN_MATCH-1)]), \
 182     match_head = s->prev[(str) & s->w_mask] = s->head[s->ins_h], \
 183     s->head[s->ins_h] = (Pos)(str))
 184 #endif
```



```
 186 /* ===========================================================================
 187  * Initialize the hash table (avoiding 64K overflow for 16 bit systems).
 188  * prev[] will be initialized on the fly.
 189  */
 190 #define CLEAR_HASH(s) \
 191     do { \
 192         s->head[s->hash_size-1] = NIL; \
 193         zmemzero((Bytef *)s->head, \
 194                  (unsigned)(s->hash_size-1)*sizeof(*s->head)); \
 195     } while (0)
 192     zmemzero((Bytef *)s->head, (unsigned)(s->hash_size-1)*sizeof(*s->head));

 197 /* ===========================================================================
 198  * Slide the hash table when sliding the window down (could be avoided with 32
 199  * bit values at the expense of memory usage). We slide even when level == 0 to
 200  * keep the hash table consistent if we switch back to level > 0 later.
 201  */
 202 local void slide_hash(s)
 203     deflate_state *s;
 204 {
 205     unsigned n, m;
 206     Posf *p;
 207     uInt wsize = s->w_size;

 209     n = s->hash_size;
 210     p = &s->head[n];
 211     do {
 212         m = *--p;
 213         *p = (Pos)(m >= wsize ? m - wsize : NIL);
 214     } while (--n);
 215     n = wsize;
 216 #ifndef FASTEST
 217     p = &s->prev[n];
 218     do {
 219         m = *--p;
 220         *p = (Pos)(m >= wsize ? m - wsize : NIL);
 221         /* If n is not on any hash chain, prev[n] is garbage but
 222          * its value will never be used.
 223          */
 224     } while (--n);
 225 #endif
 226 }
______unchanged_portion_omitted_
```