

```
extract_number_and_incr (destination, source) int
*destination; unsigned char **source; { extract_number_and_incr (destination, *source); *source += 2; } #ifndef EXTRACT_MACROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUMBER_AND_INCR(dest, src) \ extract_number_and_incr (&dest, &src) #endif /* not EXTRACT_MACROS */ #endif /* DEBUG */ /* If DEBUG is defined, Regex prints many voluminous messages about what it is doing (if the variable `debug' is nonzero). If linked with the main program in `iregex.c', you can enter patterns and strings interactively. And if linked with the main program in `main.c' and the other test files, you can run the already-written tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h> /* It is useful to test things that `must' be true when debugging. */ #include <assert.h> static int debug = 0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define DEBUG_PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf (x1, x2, x3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DEBUG_PRINT_COMPILED_PATTERN(p, s, e) \ if (debug) print_partial_compiled_pattern (s, e) #define DEBUG_PRINT_DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2) extern void printchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap) char *fastmap; { unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++]) { was_a_range = 0; printchar (i - 1); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if (was_a_range) { printf ("-"); printchar (i - 1); } } putchar ('\n'); } /* Print a compiled pattern string in human-readable form, starting at the START pointer into it and ending just before the pointer END. */ void print_partial_compiled_pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; unsigned char *p = start; unsigned char *pend = end; if (start == NULL) { printf ("(null)\n"); return; } /* Loop over pattern commands. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op"); break; case exactn: mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); printchar (*p++); } while (--mcnt); break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt, *p++); break; case stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++); break; case duplicate: printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar"); break; case charset: case charset_not: { register int c; printf ("/charset%s", (re_opcode_t) *(p - 1) == charset_not ? "_not" : ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit; unsigned char map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if ((map_byte & (1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case begline: printf ("/begline"); break; case endline: printf ("/endline"); break; case on_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt); break; case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf ("/on_failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break; case push_dummy_failure: printf ("/push_dummy_failure"); break; case maybe_pop_jump: extract_number_and_incr (&mcnt, &p); printf ("/maybe_pop_jump/0/%d", mcnt); break; case pop_failure_jump: extract_number_and_incr (&mcnt, &p); printf ("/pop_failure_jump/0/%d", mcnt); break; case jump_past_alt: extract_number_and_incr (&mcnt, &p); printf ("/-
```