7.3 Variable-length Argument Lists This section contains an implementation of a minimal version of printf, to show how to write a function that processes a variable-length argument list in a portable way. Since we are mainly interested in the argument processing, minprintf will process the format string and arguments but will call the real printf to do the format conversions. The proper declaration for printf is int printf(char *fmt, ...) where the declaration ... means that the number and types of these arguments may vary. The declaration ... can only appear at the end of an argument list. Our minprintf is declared as void minprintf(char *fmt, ...) since we will not return the character count that printf does. The tricky bit is how minprintf walks along the argument list when the list doesn't even have a name. The standard header <stdarg.h> contains a set of macro definitions that define how to step through an argument list. The implementation of this header will vary from machine to machine, but the interface it presents is uniform. The type va_list is used to declare a variable that will refer to each argument in turn; in minprintf, this variable is called ap, for "argument pointer." The macro va_start initializes ap to point to the first unnamed argument. It must be called once before ap is used. There must be at least one named argument; the final named argument is used by va_start to get started. Each call of va_arg returns one argument and steps ap to the next; va_arg uses a type name to determine what type to return and how big a step to take. Finally, va_end does whatever cleanup is necessary. It must be called before the program returns. These properties form the basis of our simplified printf: #include <stdarg.h> /* minprintf: minimal printf with variable argument list */ void minprintf(char *fmt, ...) { va_list ap; /* points to each unnamed arg in turn */ char *p, *sval; 128 int ival; double dval; va_start(ap, fmt); /* make ap point to 1st unnamed arg */ for (p = fmt; *p; p++) { if (*p != '%') { putchar(*p); continue; } switch (*++p) { case 'd': ival = va_arg(ap, int); printf("%d", ival); break; case 'f': dval = va_arg(ap, double); printf("%f", dval); break; case 's': for (sval = va_arg(ap, char *); *sval; sval++) putchar(*sval); break; default: putchar(*p); break; } } va_end(ap); /* clean up when done */ }