Advanced Input Formats in C (`scanf` and Related)

Overview

This document expands on the basic C input formats and introduces more complex input handling techniques useful in real-world programming.

Basic Format Specifiers Recap

- `%d` - signed decimal integer
- `%u` - unsigned decimal integer
- `%f` - float
- `%lf` - double
- `%c` - single character
- `%s` - string (word)
- `%x` - hexadecimal integer
- `%o` - octal integer
- `%lld` - long long integer
- `%p` - pointer address

Advanced Examples

1. Reading multiple inputs at once

Listing 1: Multiple values in one scanf

```c
int a, b;
float f;

scanf("%d %d %f", &a, &b, &f);
// User inputs: "10 20 3.14" in one line
```

2. Width specifiers and input suppression

Listing 2: Width limiting and suppression

```c
char str1[10], str2[10];

// Read up to 9 characters into str1 (reserve 1 for null char)
scanf("%9s", str1);
```

```c
// Suppress reading input (e.g., skip next word)
scanf("%*s");

// This will skip one word and not assign it anywhere
```

3. Using scanset %[] to read specific character sets

Listing 3: Scanset example

```c
char word[20];

// Read until a space or comma is found
scanf("%[^ ,]", word);

// If input: "hello,world", word will be "hello"
```

4. Reading input with delimiters

Listing 4: Reading input with delimiters

```c
int day, month, year;

// Read date format dd/mm/yyyy
scanf("%d/%d/%d", &day, &month, &year);
```

5. Reading arrays of values

Listing 5: Reading an array of integers

```c
int arr[5];

for(int i = 0; i < 5; i++) {
    scanf("%d", &arr[i]);
}
```

6. Error checking with scanf

Listing 6: Checking scanf return value

```c
int num;
int ret = scanf("%d", &num);
if(ret != 1) {
    printf("Input error: Expected an integer.\n");
}
```

7. Reading entire lines with fgets and parsing with sscanf

Listing 7: Reading a line and parsing

```c
char line[100];
int a, b;
```

```c
if(fgets(line, sizeof(line), stdin) != NULL) {
    // Parse two integers from the line
    if(sscanf(line, "%d %d", &a, &b) == 2) {
        printf("Read %d and %d\n", a, b);
    } else {
        printf("Invalid input\n");
    }
}
```

8. Reading strings with spaces

Listing 8: Reading a full line with spaces using scanset

```c
char str[100];

// Reads up to 99 characters until newline
scanf("%99[^\n]", str);
```

9. Reading until a special character

Listing 9: Reading until a semicolon

```c
char input[100];

// Reads characters until a semicolon or newline
scanf("%99[^;\n]", input);
```

10. Using `scanf` to read formatted floating point numbers with exponent

Listing 10: Reading scientific notation floats

```c
double val;

scanf("%lf", &val);
// User can input: 1.23e4 or 3.45E-2 etc.
```

Tips and Common Pitfalls

- When mixing `scanf` and `fgets`, be careful with leftover newline characters.
- Always check the return value of `scanf` to detect invalid input.
- Use width specifiers with `%s` and scanset to avoid buffer overflow.
- The space before `%c` in `scanf(" %c", &ch);` helps consume any trailing whitespace or newlines.