## Bridging from C++ to C for Input/Output

Students familiar with C++ often rely on streams (cin, cout) for input and output. In C, input/output is handled differently, primarily through functions in the <stdio.h> library. This document highlights key differences and provides examples to ease the transition.

### 1. Header Files

- C++ I/O uses <iostream>.
- C I/O uses <stdio.h>.

### 2. Output

In C++:

```
cout << "Hello, world!" << endl;
```

In C:

```
printf("Hello, world!\n");
```

Notes:

- printf uses format specifiers such as %d, %f, %s.
- endl in C++ corresponds to "\n" in C.

### 3. Input

In C++:

```
int x;
cin >> x;
```

In C:

```
int x;
scanf("%d", &x);
```

Notes:

- scanf requires the address of the variable (hence the & symbol).
- Format specifiers must match the variable type.

### 4. Common Format Specifiers

%d → int
%f → float
%lf → double
%c → char
%s → string (null-terminated char array)

## 5. Mixing Input/Output

In C++:

```
int age; string name;
cin >> name >> age;
cout << name << " is " << age << " years old." << endl;
```

In C:

```
int age; char name[50];
scanf("%s %d", name, &age);
printf("%s is %d years old.\n", name, age);
```

## 6. Key Differences

- C++ streams handle type-safety automatically; C requires explicit format specifiers.
- In C, strings are char arrays and must be managed manually.
- scanf and printf are less safe than cin/cout; buffer overflows may occur if not careful.

## Summary

Transitioning from C++ to C requires attention to format specifiers and variable addresses in input functions. With practice, printf and scanf become straightforward tools for console I/O.