CS170#01.1

Moving From C To C++

Vadim Surov

Outline

- Comments
- Accessing `stdin` & `stdout`
- Standard Library Headers
- Variable Declarations
- `const` Keyword
- `bool` Type
- Conditional Operator
- Structure Tags
- Function Prototypes
- Typecasting
- Enumerations

Comments

- C89 (ANSI C) standard for C uses only "block" comments

```
/* This is a block comment */
```

- C++ adds "to end of line" comments

```
// This is a to-end-of-line comment
```

  - Everything from `//` to the end of the line is ignored by the compiler
  - The C99 standard for C also allows these C++-style comments

Comments (contd)

- Block comments cannot be nested:

```
/*   <- Opens the block
while (i < count)
{
    i++;
    /* printf("i is %d\n", i); */ <- Closes the block
}
*/   <- What is this?
```

- Using C++ comments:

```
/*    <- Opens the block
while (i < count)
{
    i++;
    // printf("i is %d\n", i);
}
*/   <- Closes the block
```

Commenting Rules

- Use /* */ to comment out blocks of code

- Use /* */ to comment out code within a line

```
    if (p == NULL)
        /* printf("Error\n") */ ;
```

- Use // for short comments

- Some text editors show comments in a different color (e.g., Crimson Editor, Wordpad++)

Accessing `stdin` & `stdout`

- The C functions for accessing `stdin` and `stdout` (used by `printf, scanf, puts`, etc.) are all completely valid in C++
  - `#include <cstdio> // counterpart of <stdio.h> in C`

- C++ adds new ways to do this using
  - `std::cout` object to perform printing to `stdout` with overloaded `<<` operator
  - `std::cin` object to receive information from `stdin` with overloaded `>>` operator

  They are declared in `<iostream>`

Accessing `stdin` & `stdout` (contd)

- Example

```
#include <cstdio> // printf, scanf
#include <iostream> // cout, cin

int main(void) {
    int i, j;
    std::printf("Enter first number: ");
    std::scanf("%d", &i);
    std::cout << "Enter second number: ";
    std::cin >> j;
    std::cout << "Sum is: " << i + j << std::endl;
    return 0;
}
```

- Output

```
Enter first number: 54
Enter second number: 23
Sum is: 77
```

Standard Library Headers

- C standard library header files (e.g., `<math.h>`) have their C++ counterparts (e.g., `<cmath>`)

- The C++ version has no ".h" suffix but has an additional "c" prefix
  - E.g., `<ctype.h>` becomes `<cctype>`

- All standard library functions should be in the `std` namespace
  - E.g., the `sqrt` function in `<cmath>` is `std::sqrt`

Variable Declarations

- C89 requires all variable declarations to be made at the start of a block
- C99 and C++ do not force this requirement

```
int main(void)
{
    int i1; // Declaration
    int i2 = 20; // Declaration with Initialization

    i1 = 10; // Assignment

    int i3; // Declaration (error in C, valid in C++)
    int i4 = 40; // Initialization (error in C, valid in C++)

    return 0;
}
```

Variable Declarations In `for`

- Recall the syntax of the `for` statement in C:

  `for` (*expr*$_1$; *expr*$_2$; *expr*$_3$)

  statement

- In C++, it is now changed to:

  `for` (*for-init-statement;* *expr*$_2$; *expr*$_3$)

  statement

Variable Declarations In `for` (contd)

- *`for-init-statement`* is either an *expression statement* or a *simple declaration*
  - A *simple declaration* is a declaration that can be specified in one line, e.g,

    ```
    int i = 5, *pi = &i, ai[100] = {0}, j;
    ```

- This syntax allows the declaration of variables with initialization in the `for` loop

- The scope of variables declared in this way ends at the end of the `for` loop body

Variable Declarations In for Example

```
int main(void) {
    int a[10];

    for (int i = 0; i < 10; i++)
        a[i] = i;

    /* i does not exist here */

    for (i = 0; i < 10; i++) // ERROR: i is
                                   undefined!
        a[i] = i * i;

    return 0;
}
```

`const` Keyword

- In C, a variable that is declared `const` is not a compile-time constant, so this is illegal:

```
const int SIZE = 10;
int array[SIZE]; // Error
```

- To solve this in C, use a macro:

```
#define SIZE 10
int array[SIZE];
```

- In C++, variables declared `const` *are* compile-time constants, so the first code above is legal

`const` Keyword (contd)

- In C++, declaring a global variable `const` also gives it internal linkage (accessible in the file ONLY)

- Example:

```
// In file1.c
#include <stdio.h>

const int foo = 1;
void fn(void);

int main(void) {
    fn();
    printf("%d\n", foo);

    return 0;
}
```

```
// In file2.c
#include <stdio.h>

const int foo = 2;

void fn(void){
    printf("%d\n", foo);
}
```

Do they compile with C and C++ together and apart?

`bool` Type

- There is no built-in boolean type in C

- We used 3 macros:

```
#define BOOL     int
#define TRUE     1
#define FALSE    0
```

- To declare a boolean variable, we did this:

```
BOOL isDone = TRUE;
```

- `isDone` is actually of type **`int`**

bool Type (contd)

- C++ has a built-in `bool` type
- A variable of type `bool` has two possible values: `true` and `false`
- Conversion between `bool` and `int`:

```
int i = false; // i is 0
int j = true; // j is 1
bool b1 = 0; // bool is false
bool b2 = 32; // bool is true
bool b3 = -6; // bool is true
```

The Definition of NULL

- In C, NULL is a void pointer and is defined as:

```
#define NULL ((void *)0)
```

- In C++, NULL is an integer and is defined as:

```
#define NULL 0
```

- These two definitions are not the same (integer vs. void pointer) and, depending on the circumstances, may cause compiler warnings

- In practice, this is generally not a problem

Conditional Operator

- In C, the conditional operator returns an r-value

- In C++, the conditional operator can be an l-value if both the **second and third arguments are l-values of the same type**

- Example:

```
int a = 1, b = 2;
(a>b ? a : b) = 3; // a is 1, b is 3
```

Structure Tags

- In C, when we create a `struct`, e.g.,

```
struct Weapon {
    int min_damage, max_damage;
};
```

- To declare variables of type `Weapon`:

```
struct Weapon w1, w2;
```

- For C++, we can omit the keyword `struct`

```
Weapon w3, w4;
```

- Same applies to **`union`** and **`enum`**

Function Prototypes

- In C, if the compiler reads a function call before its prototype or definition, it makes some assumptions:
  - Return type is **`int`**
  - Parameter types are the types of the arguments

- Example:

```
fn(1, 3.0); // assumes fn returns int,
            // 1st argument is int,
            // 2nd argument is double
```

- In C++, this is illegal. Either function prototype or function definition must appear before the function can be called

Typecasting

- In C, typecasting is done as follows:

```
int i = 5;
float f = (float) i / 2; // f is 2.5
```

- This is still legal in C++
- C++ provides an alternative syntax for typecasting:

```
float f = float(i) / 2; // f is 2.5
```

Enumerations

- In C, `enum` variables are simply `int` variables
- In C++, `enum` variables are restricted to only the defined values
  - Each value corresponds to an integer like in C
  - `enum` variables in arithmetic operations are converted to `int`
- In C++, to assign an `int` to an `enum`, it must
  - Be typecast into the `enum`
  - Be within the enumeration's value range

Enumerations (contd)

- Determining value range:
  - Upper limit
    - Find the largest value
    - Upper limit is the smallest power of 2 that is greater than this value, minus 1
    - E.g., if largest value is 47, upper limit is $2^6 – 1 = 63$
  - Lower limit
    - Find the smallest value
    - If it is non-negative, then lower limit is 0
    - Else, lower limit is calculated like upper limit, but negative

Enumerations Example

```
enum suit {SPADE=-6, HEART, DIAMOND=8, CLUB};
suit mysuit;

mysuit = SPADE; // valid
mysuit++; // invalid
mysuit = 1; // invalid
mysuit = suit(-5); // valid (HEART)
```

Enumerations (contd)

```
enum suit {SPADE=-6, HEART, DIAMOND=8, CLUB};
suit mysuit;
mysuit = HEART + CLUB; // invalid (int to
                         enum conversion)
int i = 3 + CLUB; // valid (i is 12)
mysuit = suit(-7); // valid (in range)
mysuit = suit(5); // valid (in range)
mysuit = suit(9); // valid (in range)
mysuit = suit(16); // invalid (out of range)
```

Note: In the last line the result is "undefined" so it can be anything at runtime.