

CS349/SE382 A1 C Programming Tutorial

Erin Lester

January 2005

Differences from C++

Comments

Variable Declarations

Objects

Dynamic Memory

Boolean Type

structs, enums and unions

Other Differences

The Event Loop

Comments:

- ▶ C has no one-line comments, only block comments

```
/* this comment is valid in C */  
/* so is  
    this one */  
// but this one is not  
i = 2; // and neither is this one
```

Variable Declarations:

- ▶ Variables must be declared at the beginning of code blocks, before any other types of statements

```
#include <stdio.h>
void main()
{
    char c;
    int i;
    i = 10 + 2;
    char *mystring = "Hello World"; /* not allowed */
    c = '\n';
    printf("%s%c", mystring, c);
}
```

Objects:

- ▶ C has no objects
- ▶ use ADTs — remember cs246 — to be modular

Creating a mock class/ADT

Step 1. Create a header file for your mock class
include:

- ▶ empty structure declaration
- ▶ function headers w/ structure as parameter

Creating a mock class/ADT

Step 2. Implement the code for the mock class
in a '.c' file:

- ▶ include your '.h' file
- ▶ declare the structure
- ▶ implement the ADT's functions

Dynamic Memory:

Dynamic Memory Allocation

- ▶ memory is allocated using 'malloc()'
- ▶ malloc() takes the number of bytes of memory as its parameter
- ▶ it returns a (void*) pointer to the memory allocated
- ▶ the pointer can then be cast to the desired type and the memory assigned to a variable

```
typedef struct {  
    ...  
} myStruct;  
  
...  
  
myStruct *m =  
    (myStruct *) malloc( sizeof( myStruct ) );
```

Freeing Dynamic Memory

- ▶ memory is freed using 'free()'
- ▶ free() takes the address of a non-null variable as its parameter
- ▶ there is no garbage collection, so don't forget to clean-up what you create

```
free( m );
```

Boolean Type:

- ▶ zero evaluates to false
- ▶ non-zero evaluates to true
- ▶ use typedef and defines to mimic the boolean type

```
/* Here's how to mimic a boolean type */  
#define true 1  
#define false 0  
typedef int bool;  
  
...  
  
bool myBool = true;
```

Structures:

- ▶ struct types must be qualified by the word struct
- ▶ 'struct' is prepended to their name when referencing them
- ▶ can use typedef to give a single name to a C struct

Declaring a C struct normally

```
struct yourStruct
{
    ... /* variables in the structure */
};
```

versus using the typedef-method

```
typedef struct
{
    ... /* variables in the structure */
} myStruct;
```

Differences in declaring the two structures:

```
struct yourStruct boohoo;  
myStruct haha;
```

C enumerations and unions are similar: you have to use the syntax 'enum xxx' and 'union xxx' in referencing them.

Enumeration Differences:

- ▶ C++ enumerations are a set of ints
- ▶ C enumerations are typedefed ints (i.e. no range restrictions)
- ▶ compiler does not catch assignments outside of the declared range

```
typedef enum day = {SUNDAY, MONDAY, TUESDAY,  
                  WEDNESDAY, THURSDAY,  
                  FRIDAY, SATURDAY} Day;
```

...

```
Day today = 9999; /* Logical error, today should  
                  be between 0 and 6, but  
                  compiler will not complain! */
```

Other Differences:

▶ **Libraries:**

- ▶ C uses different libraries than C++
- ▶ For example C I/O uses 'stdio.h' instead of 'iostream.h', 'printf' instead of 'cout', and 'scanf' instead of 'cin'
- ▶ See a C reference for more details

▶ **C does not allow function overloading.**

▶ **C does not permit default values for function arguments.**

Basic Event Loop

```
while( 1 ) {  
    if ( getNextEvent( &event ) )  
        processEvent( event );  
    doEventIndependentProcessing();  
    updateDisplay();  
}
```

Basic Event Loop

- ▶ *getNextEvent()*: blocking or non-blocking
- ▶ *processEvent()*: depends on event type → switch stmt
- ▶ *doEventIndependentProcessing()*: do non-event specific processing (i.e. time-based processing)
- ▶ *updateDisplay()*: reflect changes made by event (& possibly time)

Questions?

Feel free to ask questions now or via email:
eclster@cgl.uwaterloo.ca