

ELEC3730 Embedded Systems Lecture 2: C Essentials

- Conditional statements
- Arrays
- Characters and string operations
- Booleans
- #define and typedef



Conditional Statements

- Example:

```
if (age < 0)
{
    printf ("warning: negative age\n");
    age = -age;
}
```

- Syntax:

```
if (condition) statement;
if (condition) statement else statement;
```

- Rules:

- the condition is an int (no booleans)
- parentheses required around conditional expression
- use curly braces to make a compound statement

More Conditionals

- Example:

```
if (x < 0)
    printf ("x is less than 0\n");
else if (x == 0)
    printf ("x is equal to 0\n");
else
    printf ("x is greater than 0\n");
```

- What's wrong with this?

```
if (x < 0)
    if (y < z)
        printf ("y is less than z\n");
else
    printf ("x not less than 0\n");
```

While Loops

- Example:

```
/* print "hi" forever */
while (1)
    printf ("hi");
```

- Syntax:

```
while (condition)
    statement;
```

- Rules (again):
 - the condition is an *int* (no booleans)
 - parentheses required around conditional expression
 - use curly braces to make a compound statement

For Loops

- Example:

```
/* print "hi" three times */
int i; /* i continues to exist when
loop ends */
for (i = 0; i < 3 ; i++)
    printf ("hi");
```

- Syntax:

```
for (statement1; condition; statement2)
    statement3;
```

- Equivalent to:

```
statement1;
while (condition)
{
    statement3;
    statement2;
}
```

Loop Example

```
/* print squares up to 100 */
main ( )
{
    int j, up = 100;
    for (j = 0; j * j <= up; j++)
        printf ("%d \n", j * j);
}
```

NB - Can't initialise in loop: `for (int j = 0; ...`

More efficient version:

```
void main ( )
{
    int j, up = 100, sq;
    for (j = 0; (sq = j * j) <= up; j++)
        printf ("%d \n", sq);
}
```

No checking by compiler of array index out of bounds!

```
#include<stdio.h>
int main(int argc, char **argv)
{
    float a[10];
    a[23456] = 3.45;
    return(0);
}

csh>gcc test.c -Wall
csh>
```

Characters

- Characters

```
char a, b, c1, c2;
a = '0'; b = '\037'; b='\x1f' c1 = 'K'; c2 = c1 + 1;
```

- Assigns values: 48, 31, 75, 76
- The sequences '0', ..., '9', 'a', ..., 'z', 'A', ..., 'Z' contain characters numbered consecutively

- Casting

```
printf ("%c %d\n", c1, (int) c1);
```

- Outputs: K 75

Strings

- Strings are '\0'-terminated arrays of char :

```
char s[3] = "hi"; /* invisible '\0' */
char t[3] = {'h', 'i', '\0'};
```

- String operations

```
#include <string.h>
strlen ("there"); /* returns 5 */
strcpy (s, t); /* copy t to s */
strcmp (s, t) /* alphabetical comparison */
```

More on Booleans - an example of #define

- C has boolean operations on int:

```
== != && || !
```

- However, C has no boolean type. Make it!

```
#define boolean_t int;  
#define TRUE 1  
#define FALSE 0
```

- Example:

```
boolean_t xIsFive, zAsBig, result;  
xIsFive = (x == 5);  
zAsBig = (z >= x);  
if (xIsFive || zAsBig)  
{  
    result = TRUE;  
}
```

Naming your own type - typedef

- A better way - use typedef

```
typedef int boolean_t;
```

Syntax:

```
typedef existing-type new-type;
```

Comments

- typedef** does not create a new type
- It only creates a label for an existing type
- Enhanced readability of code and compiler error checking

Typedef in action

```
typedef char * string_t;  
typedef int boolean_t;
```

```
string_t msg="No more room";  
boolean_t full_class;
```

```
If (num_students >= MAX_SIZE)  
    full_class = TRUE;
```

Arrays

```
int years[45];
float temperatures [11];
void main ()
{
    years[0] = 2000;
    temperatures[10] = -45.67;
}
```

- Rules:

- indices **start at zero** !;
- maximum valid index is the size of the array minus 1;
- C lets you go beyond the declared boundaries with no error message or compiler warnings! Possible memory corruption!
- `temperatures[11]` compiles and runs - completely unpredictable behaviour.

Type Conversion and Casting

```
include <stdio.h>
void main(void)
{
    int i,j = 12;      /* i not initialized, only j */
    float f1,f2 = 1.2;

    i = (int) f2;      /* explicit: i <- 1, 0.2 lost */
    f1 = i;            /* implicit: f1 <- 1.0 */
    f1 = f2 + (float) j; /* explicit: f1 <- 1.2 + 12.0 */
    f1 = f2 + j;       /* implicit: f1 <- 1.2 + 12.0 */
}
```
