COMsW 1003-1

Introduction to Computer Programming in C

Lecture 6

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html
Announcements

Homework 1 is due next Monday

Exercise 2 is out
Today

- Strings
- Control Flow
- Loops (if time permits)
• **Multidimensional arrays**

\[
\text{int } X[4][3]; \quad // \text{ a matrix containing } 4 \times 3 = 12 \text{ integers}
\]

\[
\begin{array}{ccc}
X[0][0] & X[0][1] & X[0][2] \\
X[1][0] & X[1][1] & X[1][2] \\
X[3][0] & X[3][1] & X[3][2]
\end{array}
\]

• **Indexing starts at 0!**

\[
X[0][0] = 1;
X[3][1] = 7;
\]

• **Initialize says**

\[
\begin{align*}
\text{int } \text{arr}[4] &= \{ 3, 6, 7, 89 \}; \\
\text{int } \text{arr2}[2][4] &= \{ \{19, 2, 6, 99\}, \{55, 5, 555, 0\} \}; \\
\text{int } \text{arr[]} &= \{ 3, 6, 77\};
\end{align*}
\]

This automatically allocates memory for an array of 3 integers.
Strings

- Strings are arrays of `char`
- ‘\0’ is a special character that indicates the end of a string

```c
char s[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

We need 6 characters because there is ‘\0’

```c
char s[10] = "Hello";
```

- Difference between string and `char`

```c
char c = 'a';  // 'a'
char s[2] = "a";  // 'a' '\0'
```
Strings functions

String specific functions are included in the library `string.h`

```c
#include <string.h>

char s[6];
s = "Hello";
```

Illegal! String assignment can be done only at declaration!

- `strcpy()` : copy a string to another

```c
strncpy(string1, string2);
```

Copy string2 to string1

```c
char s[6];
strncpy(s, "Hello");
```
String functions

String specific functions are included in the library `string.h`

- `strcmp()`: compare two strings

```c
char s1[] = "Hi";
char s2[] = "Him";
char s3[3];
strcpy(s3, s1);
int x = strcmp(s1, s2);  // x != 0
int y = strcmp(s1, s3);  // y = 0
```

>Returns:
0 if `string1` and `string2` are the same value
!= 0 otherwise
Strings functions

String specific functions are included in the library `string.h`

- `strcat()` : concatenate two strings
  ```c
  strcat( string1 , string2); 
  ```
  Concatenate `string2` at the end of `string1`

  ```c
  char s1[] = “Hello ”;
  char s2[] = “World!”;
  strcat(s1, s2);
  ```
  `'H' 'e' 'l' 'l' 'o' ' ' 'W' 'o' 'r' 'l' 'd' \0`

- `strlen()` : returns the length of a string (does not count `\0`)
  ```c
  strlen( string );
  ```

  ```c
  char s1[] = “Hello”;
  int x = strlen(s1);    // x = 5
  ```
Reading Strings

Use functions from library `stdio.h`

- `fgets()`: get string from standard input (command line)

```c
fgets( name , sizeof(name) , stdin);
```

```c
char s1[100];
fgets( s1, sizeof(s1), stdin);
```

Reads a maximum of `sizeof(name)` characters of a string from stdin and saves them into string `name`

NOTE: `fgets()` reads the newline character `\n`, so we should substitute it with `\0`;

```
s1[strlen(s1)-1] = '\0';
```

```
\H e l l o
```

• `sizeof()`: returns the size (number of bytes occupied in memory) of a variable (for strings it counts the number of elements, including `\0`)
Reading numbers – Option 1

• First, read a string
• Then, convert string to number
• `sscanf()` : get string from standard input (command line)

```
sscanf( string, "format", &var1, ..., &varN);
```

```c
char s1[100];
int x, y;
printf("Please enter two numbers separated by a space\n")
fgets( s1, sizeof(s1), stdin);
User enters:  3 18
sscanf( s1, "%d %d", &x, &y );
// x = 3; y = 18;
```
Reading numbers – Option 2

• Read directly the number
• `scanf()` : get string from standard input (command line) and automatically convert into a number

```c
scanf( "format", &var1, ..., &varN);
```

```c
int x, y;
printf("Please enter two numbers separated by a space\n")
```

User enters: 3 18

```c
scanf( "%d %d", &x, &y );
```

// x = 3; y = 18;
Strings functions - recap

char s1[] = “Hello”;  char s2[] = “He”;  int x;  char c;

• strcmp( s1, s2)  x = strcmp(s1, s2) // x != 0

• strcpy( s1, s2 )  strcpy( s2, s1 ); // s2 = “Hello”

• strcat( s1, s2)  strcat( s2, s1 ); //s2 = “HelloHello”

• strlen( s )  x = strlen(s1);   // x  = 5;

• sizeof( s )  x = sizeof(s1);    // x = 6;

• fgets( s, sizeof(s1), stdin)  fgets( s1, sizeof(s1), stdin);
User enters “7R”

• sscanf( s, “%d”, &var)  sscanf( s1, “%d%c”, &x, &c);  // x = 7; c = ‘R’;
Example – sumNums.c
Control Flow

• So far we have seen **linear programs**, statements are executed in the order in which they are written

• What if we want to skip some instructions, or execute them only under certain conditions?

• Solution: **control flow**
Control flow – General syntax

```c
keyword ( condition ) {
    body statement 1;
    ...
    body statement n;
}
```

The body is executed only if the `condition` is true!

If the body of the control flow has only one statement, we can optionally not use the `{ }`

```c
keyword ( condition )
    body statement 1;
```
Control flow – if

• To execute a particular body of statements only if a particular condition is satisfied

```
if ( condition ) {

    body statement 1;
    
    body statement n;

}
```

Example

```
int x = 3, y;

if ( x > 2 ) {

    x++; 
    y = x;

}

printf(“y = %d
”,y);
```
Control flow - else

- To execute a particular body of statements only if a particular *condition* is **not** satisfied

```c
if ( condition ) {
    body statement 1;
    .
    .
    body statement n;
}
else {
    body statement 1;
    .
    .
    body statement m;
}
```

**Example**

```c
int x = 3, y;

if ( x > 2 ) {
    x++;
    y = x;
}
else {
    y = 2 * x;
}

printf("y = %d\n", y);
```
Control Flow – if/else example

```c
int x = 3, y = 1;

if( x > 2 )
    if( x == 4)
        y = x;
else
    y = 2 * x;

printf("y = %d\n",y);
```
Control Flow – if/else example

```c
int x = 3, y = 1;
if( x > 2 )
  if( x == 4)
    y = x;
else
  y = 2 * x;
printf("y = %d\n",y);
```

*else* refers always to the last *if* that was not already closed by another *else*
Control Flow – if/else example

```c
int x = 3, y = 1;

if( x > 2 ) {
    if( x == 4 ) {
        y = x;
    }
    else {
        y = 2 * x;
    }
}

printf("y = %d\n",y);
```

This is why we need brackets and indentation!
int x = 3, y = 1;

if( x > 2 ) {
    if( x == 4 ) {
        y = x;
    }
}
else {
    y = 2 * x;
}

printf("y = %d\n",y);

Using brackets we can change the if to which the else refers
Control flow - Switch

Equivalent to a series of if/else statements

```c
int i,j;

switch( i ) {
    case 1:
        j = i + 1;
        break;
    case 10:
        j = i - 1;
        break;
    default:
        j = 1;
        break;
}
```

```c
switch ( variable ) {
    case val1:
        statement 1;
        break;
    case val2:
        statement 1;
        /* fall through */
        statement 1;
        break;
    default:
        statement 1;
        break;
}
```
Control flow - Switch

Equivalent to a series of if/else statements

```c
int i, j;

switch (i) {
    case 1:
        j = i + 1;
        break;
    case 10:
        j = i - 1;
        break;
    default:
        j = 1;
        break;
}
```

These values are CONSTANT

If variable has value different from all other cases

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Any other number</td>
<td>1</td>
</tr>
</tbody>
</table>
Control flow - Switch

Equivalent to a series of if/else statements

```c
switch ( variable ) {
    case val1:
        statement 1;
        break;
    case val2:
        statement 1;
        /* fall through */
        statement 1;
    default:
        statement 1;
        break;
}
```

```c
int i,j;

switch( i ) {
    case 1:
        j = i + 1;
        break;
    case 10:
        j = i - 1;
        break;
    default:
        j = 1;
    }
```

If variable has value different from all other cases

These values are CONSTANT

After last case I can avoid using break
Switch

Equivalent to a series of if/else statements

```c
switch ( variable ) {
    case val1:
        statement 1;
        ...
        break;
    case val2:
        statement 1;
        ...
        /* fall through */
        ...
    default:
        statement 1;
        ...
        break;
}
```

```c
int i,j;
switch ( i ) {
    case 1:
        j = i + 1;
        break;
    case 10:
        j = i - 1;
        default:
        j = 1;
}
```

`variable` can only be `char` or `int`!
Control Flow - Loops

• What if we want to perform the same operation multiple times?
• Example: we want to initialize all elements in a 100 dimensional array of integers to the value 7

```c
int arr[100];
arr[0] = 7;
arr[1] = 7;
arr[2] = 7;
arr[3] = 7;
.
.
arr[99] = 7;
```

This is crazy!
Loops - while

- To execute a particular body of statements only until a particular condition is satisfied

```c
while ( condition ) {
    body statement 1;
    
    body statement 2;
    
    body statement n;
}
```

**Example**

```c
int i = 0;
int arr[100];

while ( i < 100 ) {
    arr[i] = 7;
    i++;
}
```
Loops – do/while

- **First** execute body statements, **then** check if condition is satisfied

```c
int i = 10, j = 0;
do {
    body statement 1;
    body statement n;
} while ( condition );
```

**Example**

```c
int i = 10, j = 0;
while ( i < 10 )
{
    j++;
    i++;
}
```

```c
int i = 10, j = 0;
do
{
    j++;
    i++;
} while ( i < 10 );
```

```c
j = ?
```
Loops – do/while

• **First** execute body of statements, **then** check if *condition* is satisfied

```c
int i = 10;
int j = 0;
do {
    body statement 1;
    ...
    body statement n;
} while ( condition );
```

**Example**

```c
while ( i < 10 )
{
    j++;
    i++;
}
```

**Example**

```c
j = 0
```
Loops - break

• To interrupt a loop once a certain condition different from the one in the loop declaration

When `break` is reached, the statements after it are ignored and the program exits the loop

```c
while( condition1 ){
    body statement 1;
    .
    .
    if( condition2 )
        break;
    .
    .
    body statement n;
}
```

Example

```c
int i = 0;
char s[10] = "hi";

while( i < 10 )
{
    if(s[i]==\0)
        break;
    .
    .
    printf("%c",s[i]);
    i++;
}
```
Loops - continue

• To ignore the following instructions in a loop

```c
while( condition1 ){
    body statement 1;
    :
    if( condition2 )
        continue;
    :
    body statement n;
}
```

Example

```c
int i = 0, sum = 0;
int s[3] = {7, 5, 9};

while( i < 3 )
{
    if(s[i] < 6)  
        continue;

    sum += s[i];
}
```
break vs. continue

int x = 0, y = 0;
while (x < 10) {
    x++;
    if (x == 3) {
        continue;
    }
    y++;
}

y = ?

int x = 0, y = 0;
while (x < 10) {
    x++;
    if (x == 3) {
        break;
    }
    y++;
}
break vs. continue

```c
int x = 0, y = 0;
while( x < 10 ) {
    x++;
    if(x == 3) {
        continue;
    }
    y++;
}
y = 9
```

```c
int x = 0, y = 0;
while( x < 10 ) {
    x++;
    if(x == 3) {
        break;
    }
    y++;
}
y = 2
```
Loops - for

```c
for (initial state; condition; state change) {
    body statement 1;
    ...
    body statement n;
}
```

Example

```c
int i;
int arr[100];
for ( i = 0; i < 100 ; i++ ) {
    arr[i] = 7;
}
```

```c
int i = 0;
int arr[100];
while ( i < 100 ) {
    arr[i] = 7;
    i++;
}
```
Homework 1 review

HOW TO COMPRESS/UNCOMPRESS folders in UNIX

• Compress folder ~/COMS1003/HW1 to HW1.tar.gz
  
  ```
  tar -zcvf HW1.tar.gz ~/COMS1003/HW1
  ```

• Uncompress HW1.tar.gz to folder ~/COMS1003/HW1new
  
  ```
  tar -zxvf HW1.tar.gz -C ~/COMS1003/HW1new
  ```

  (note: ~/COMS1003/HW1new must exist already)