EECS2031- Software Tools

C-Control Flow
(K&R Ch.3)

YORK UNIVERSITÉ UNIVERSITY

Statements

- Program to execute
  - Ended with a ;
- Expression statement (ch2)
  - `i+1;  i++;  x = 4;`
- Function call statement (ch4)
  - `printf("the result is %d");`
- Control flow statement (ch3)
  - `if else, for(), while,  do while, switch`

Same in Java

YORK UNIVERSITÉ UNIVERSITY

Control Flow Statements

- Variables hold data, types tell us what kind of data we have, and expressions allow us to inspect and modify data.
- Control flow statements specify order of computation: i.e. which statements do we execute and when?
- You should find the following familiar (much like Java)

YORK UNIVERSITÉ UNIVERSITY

Control Flow

- There are three kinds of control structures to control the execution flow of a program:
  - Sequence
    - Blocks{} as a group of statements specify sequential flow.
  - Selection
    - `if … else …, switch`
  - Repetition
    - `for, while, do … while`

YORK UNIVERSITÉ UNIVERSITY

if-else

```
if (expression)
   statement 1
else
   statement 2
```

- If **`expression`** is “true” (i.e. not 0) then **`statement 1`** is executed, otherwise **`statement 2`** is executed
- The **`else`** part is optional.

YORK UNIVERSITÉ UNIVERSITY

if-else-if else

```
if (expression 1)
   statement 1
else if (expresson 2)
   statement 2
else if (expresson 3)
   statement 3

.....
else
    statement k
```

YORK UNIVERSITÉ UNIVERSITY

Be extremely careful!

```
if (x = 1)   // should be x==1
   statement-1
else if (expresson 2)
   statement-2
else if (expresson 3)
   statement-3
.....
```

Statememt-1 will always be executed!

reason: x=1 has a return value 1, which is true in ANSI C!

```
indigo 311 % javac Hello.java
Hello.java:13: incompatible types
found   : int
required: boolean
             if (x = 1){
                   ^
1 error
```

Java catches this typo

Conditional Expression

`test ? expr_true: expr_false`

- A specialized form of "**`if`**…**`else`**…"
- If the expression **`test`** is true, then the value of the whole expression is **`expr_true`**, otherwise the value is **`expr_false`**

Conditional Expression

`x= y ? 4:3`

has the same effect as

```
if (y)
   x=4;
else
   x=3;
```

Except that **`x=y? 4:3`** is an expression and has a value (whatever **`x`** gets set to)

YORK UNIVERSITÉ UNIVERSITY

Conditional Expression

```
z=(a > b) ? a: b
      /*z= max(a,b)*/

y=(x>=0)? x:-x;
     /*y = |x| */
```

YORK UNIVERSITÉ UNIVERSITY

while

```
while(expression)
   statement
```

- while **`expression`** is true, execute **`statement`**
- **`expression`** is evaluated at the beginning of the loop
  - This means before the first time the loop is executed

YORK UNIVERSITÉ UNIVERSITY

do-while

```
do
   statement
while(expression);
```

- like “while”, except that expression is executed after the loop
  - i.e. the first time through, the loop is executed before expression is evaluated

YORK UNIVERSITÉ UNIVERSITY

do-while Example

```
do{
   printf("Enter a positive integer:");
   scanf("%d", &response);
while ( response <=0);
```

- What does this do?
- How to write it using **`while (…) { }`**?

YORK UNIVERSITÉ UNIVERSITY

while vs. do-while

```
void stringcopy(char dest [], char src [])
{
   int i=0;
   while (src[i] != '\0'){
      dest[i] = src[i];
      i++;
   }
   dest [i]= '\0';
}
```

```
void stringcopy(char dest [], char src [])
{
   int i=0;
   do{
      dest[i] = src[i];
      i++;
   }
   while (src[i] != '\0')
}
```

YORK UNIVERSITÉ UNIVERSITY

for loop

```
for ( expr1; expr2;  expr3)
   statement
```

equivalent to:

```
expr1;
while(expr2) {
   statement;
   expr3;
}
```

YORK UNIVERSITÉ UNIVERSITY

for loop

```
int i;
for ( i=0; i < 10;  i++)
   statement
```

equivalent to:

```
int i;
i =0;
while(i< 10) {
   statement;
   i++;
}
```

```
for ( int i=0; i < 10; i++)
   statement
```

Not valid in C89 and lab (C89 + some C99)
Valid in C99

YORK UNIVERSITÉ UNIVERSITY

for loop

```
for ( expr1; expr2;  expr3)
   statement
```

- **expr1** and **expr3** can include multiple expressions.
  - Use the "," operator to separate them.

```
for(i=0,j=5; i<10;  i++,j--)
{
    …do something…
}
```

YORK UNIVERSITÉ UNIVERSITY

for loop

What does the following function do?

```
void fun(char s [], int c)
{
   int i, j;
   for(i=0,j=0; s[i] !='0'; i++)
      if (s[i]!=c)
         s[j++]=s[i];

   s[j]='0';
}
```

YORK UNIVERSITÉ UNIVERSITY

Controlling Loops

break

- exits a loop (for, while, do)

continue

- skips to the end of the loop (but executes **expr3** in **for (expr1; expr2; expr3))**

Example

```
#include <stdio.h>
main(){
   int n;
   do{
      printf("\nEnter the number :");
      scanf("%d", &n);
      if (n<0){
         break;
      }
      if (n>10) {
         printf("Skip the value\n");
         continue;
      }
      printf ("The number is : %d\n", n);
   } while (n != 0);
```

while and while(1) break

```
void stringcopy(char dest [], char src [])
{
   int i=0;
   while (src[i] != '\0'){
      dest[i] = src[i];
      i++;
   }
   dest [i]= '\0';
}
```

```
void stringcopy(char dest [], char src [])
{  int i=0;
   while (1) {
      dest[i] = src[i]; // including '\0'
      if (dest [i]= '\0')
        break;
      i++;
   }
}
```

switch

```
switch expresson{
      case const-expr: statements
      case const-expr: statements
      ……
      default:statements
}
```

- If expression matches one of the case, then execution begins at that case
- Otherwise execution starts at “default”

switch

- "switch" is something like "if …else if…else if …" but not quite!
- fall-through:

```
x=0;
switch(x) {
   case 0: printf("Hello\n");
   case 1: printf("goodbye\n");
}
```

- What is printed?

YORK UNIVERSITÉ UNIVERSITY

switch

- The "**break**" keyword allows us to jump out of a switch statement and avoid fall-through

```
switch(x) {
   case 0: printf("Hello\n");
                break;
   case 1: printf("Goodbye\n");
}
```

YORK UNIVERSITÉ UNIVERSITY

switch

```
#include<stdio.h>

int main(void) {
   int a1, a2; char op;
   scanf("%d %c %d", &a1,&op,&a2);

   switch (op) {
      case '+':
         a1 += a2;
         break;
      case '-':
         a1 -= a2;
         break;
      case '*':
         a1 *= a2;
         break;
      case '/':
         a1 /= a2;
         break;
   }
   printf("The result is %d\n", a1);
}
```

switch

- All case must be:
  - unique (cannot duplicate cases)
  - constant
    - e.g. `case 2*x:` is invalid if `x` is a variable

The Forbidden Control

- C also has support for “goto”

```
if (! correct)
   goto handle_error;
...
handle_error:
   oops();
```

Goto sends control to a label

Label is like a “case”in a “switch” but without”case”

Control Flow Traps

- dangling”else”

```
if   (…)
   if(…)
      …
else
   …
```



```
if (...)
{
   if(…)
      …
}
else
   …
```

This is correct

Control Flow Traps

- **`switch`**
  - avoid deliberate fall-through. Use break.
  - put a “break” after the last case(even though it’s lobically unnecessary)
- **`goto`**
  - avoid it!
  - easily leads to “spaghetti”code
  - rarely needed