Control Flow

Statements

- An expression such as `x = 0` or `j++` becomes a statement when it is followed by a semicolon `;`
- Braces `{` and `}` are used to group declarations and statements together into a *compound statement*.
  - `{ int j; j=2+3; j++; }` /*this entire thing now is a statement */
- *Compound statement* is also called *block.*

Compound Statement

- Variables can be declared in *any* block. Discussion of this is deferred.
- There is no semicolon after the right brace that ends a block.

```
{
    int j;
    j=2+3;
    j++;
}
```

If-Else

- The if-else statement is used to express decisions.
- Formally the syntax is,
  - if (expression)
    statement1
    else
    statement2
- The else part is optional
- The expression is evaluated; If it is true (non-zero) then statement1 is executed, otherwise (if there is an else part) statement2 is executed.

```
if (expression)
        statement1
else
        statement2
```

- Both the statements could be compound or simple.

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0, y = 5, k = 10;
if (y == 5)
{
        k ++;
        j =  k * k;
}

printf("%d", j);
```

Value of j ?

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0, y = 5, k = 10;
if (y == 6)
{
        k ++;
        j =  k * k;
}

printf("%d", j);
```

Value of j ?

If-Else

- if (x != 0) is same as if (x)

- Check

```
int j=0, y = 5, k = 10;
 if (y == 6)
 {
          k ++;
          j =  k * k;
 }
else      j = k;
printf("%d", j);
```

Value of j ?

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0, y = 5, k = 10;
 if (y == 5)
 {
        k ++;
        j =  k * k;
 }
else
        j = k;
```

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0,y = 5, k = 10;
 if (y == 5)
 {
        k ++;
        j =  k * k;
 }
else
        j = k;
```

Is this correct?

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0,y = 5, k = 10;
 if (y == 5)
 {
        k ++;
        j =  k * k;
 }
else
        j = k;
```

- Check

```
int j=0,y = 5, k = 10;
 if (y == 5)

        k ++;
        j =  k * k;

else
        j = k;
```

If-Else

- `if (x != 0)` is same as `if (x)`

- Check

```
int j=0,y = 5, k = 10;
 if (y == 5)
 {
        k ++;
        j =  k * k;
 }
else
        j = k;
```

- Check

```
int j=0,y = 5, k = 10;
 if (y == 5)

        k ++;
        j =  k * k;

else
        j = k;
```

What about this?

If-Else

- Check
  - x = 0;
    if (2 != 1+1) ;
      x = 5;
    printf("%d",x);

What is the output?

If-Else

- Check
  - x = 0;
    if (2 != 1+1) ;
    x = 5;
    printf("%d",x);

- Syntax :

  if (expr) statement

- ; /* a null statement*/

- So the output is 5

If-Else

- Check
  - int j = 200;
    if (j = 5)
     printf("A");
    else
     printf("B");

- What is the output?

If-Else

- Check
  - int j = 200;
    if (j = 5)
     printf("A");
    else
     printf("B");

- What is the output?

  A

- because j=5 has value 5 which is non-zero(true)

If-Else

- Check
  - int j = 200;
    if (j = 5)
     printf("A");
    else
     printf("B");

- What is the output?

  A

- because j=5 has value 5 which is non-zero(true)

- This is a common pit-fall. Beware!

These are equivalent

```
if( n > 0 )
    if ( a > b)
     z = a;
else
    z = b;
```

```
if( n > 0 )
{
    if ( a > b)
        z = a;
    else
        z = b;
}
```

- else associates with the closest previous else-less if.

These are equivalent

```
if( n > 0 )
 if ( a > b)
      z = a;
else
      z = b;
else
      z = c;
```

```
if( n > 0 )
{
      if ( a > b)
            z = a;
      else
            z = b;
}
else
{
      z = c;
}
```

If-Else ladder

```
if (exp1)
        stmt1
else   if (exp2)
        stmt2
 else  if (exp3)
        stmt3
 else
        stmt4
```

- These are useful for multi-way decisions

If-Else ladder

```
if (exp1) {
        stmt1  }
else { if (exp2)
        stmt2
 else  if (exp3)
        stmt3
 else
        stmt4
        }
```

- These are useful for multi-way decisions

If-Else ladder

```
if (exp1)
        stmt1
else   if (exp2)
        stmt2
 else  if (exp3)
        stmt3
 else
        stmt4
```

- Only one statement in the ladder is executed
- If exp1 is true then stmt1 is executed and all other in the ladder are ignored.
- If exp1 is false and exp2 is true then only stmt2 is executed
- Stmt4 can be seen as a default

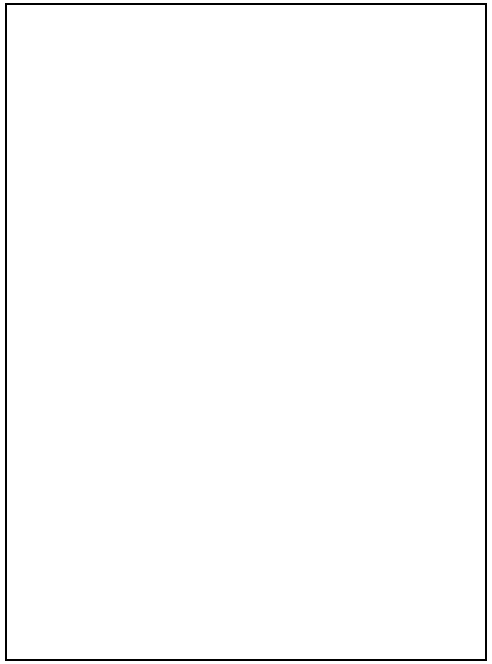
Switch

- The switch statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly.
- switch (expression) {
  case const-expr : statements
  case const-expr : statements
  default : statements
  }
  - The type of the expression should be either int or char.

Switch

```
int j;
scanf (“%d”, &j);
switch(j) {
        case 0: printf(“ zero\n”);
        case 1: printf(“ one\n”);
        case 2: printf(“ two\n”);
        default: printf(“ other\n”);
}
```

Output?



Switch

```
int j;
scanf (“%d”, &j);
switch(j) {
        case 0: printf(“ zero\n”);
        case 1: printf(“ one\n”);
        case 2: printf(“ two\n”);
        default: printf(“ other\n”);
}
```

Output?

```
 $./a.out
0
zero
one
two
other
 $
```

- switch simply transfers control once to the matching case.

breaking a switch

- break; /*this is a statement which can break a switch */
- break exits the switch block.
- break can be used with other control flow structures, but discussion is deferred.

Use break statements

```
int j;
scanf (“%d”, &j);
switch(j) {
      case 0: printf(“ zero\n”);
               break;
      case 1: printf(“ one\n”);
               break;
      case 2: printf(“ two\n”);
      default: printf(“ other\n”);
}
```

```
$./a.out
0
zero
$./a.out
1
one
$./a.out
2
two
other
$
```

Switch

- `default: statements /*optional*/`
- The control is transferred to default, if it exists and none of the cases matches the expression value.
- Even if there are multiple statements to be executed in each case there is no need to use `{` and `}` (i.e., no need for a compound statement as in if-else).
- One can not have something like

  `case j <= 20:`
- All that we can have after the `case` is a constant expression.

switch

- You can also use char values.
  - char c ; c = getchar ( );

```
switch (c)
{
        case ‘a’:
        case ‘A’: printf(“apple”);  break;
        case ‘b’:
        case ‘B’: printf(“banana”); break;
}
```

  - Empty cases might be useful

goto

- goto label; /* label is similar identifier like a variable name */

  /* this transfers control to */

  label:

goto

- goto makes the execution to jump to the label:
- Backward jump can create a loop.

- But, goto is not used in practice.
- It makes programs illegible.
- Whatever you can write with a goto statement can be written without goto also (but by using appropriate conditional or loop statements).

- What will be the output of the C program?

```
#include<stdio.h>
int main()
{
        int i = 5, j = 6, k = 7;
        if(i > j == k)
                printf("%d %d %d", i++, ++j, --k);
        else
                printf("%d %d %d", i, j, k);
        return 0;
}
```

A. 5 7 6

B. 5 6 7

C. 6 6 6

D. 5 7 7

- What will be the output of the C program?

```c
#include<stdio.h>
int main()
{
        int i = 5, j = 6, k = 7;
        if(i > j == k)
                printf("%d %d %d", i++, ++j, --k);
        else
                printf("%d %d %d", i, j, k);
        return 0;
}
```

A. 5 7 6

B. 5 6 7 ✓

C. 6 6 6

D. 5 7 7

What will be the output of the C program?

```
#include<stdio.h>
int main()
{
int i = 5;
  if(i = i - 5 > 4)
          printf("inside if block");
  else
          printf("inside else block");
return 0;
}
```

A. Compilation Error

B. None

C. inside if block

D. inside else block

What will be the output of the C program?

```
#include<stdio.h>
int main()
{
int i = 5;
  if(i = i - 5 > 4)
          printf("inside if block");
  else
          printf("inside else block");
return 0;
}
```

A. Compilation Error

B. None

C. inside if block

D. inside else block ✓

/* end of 1-if-else-switch-goto */