CISC 1600/1610
Computer Science I

Flow of control

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

Linear execution of statements

- Each action performed in written order

What is the result of this set of statements?

```
int a=1, b=2, c;
c = a+b;
a=5;
cout << c;
```

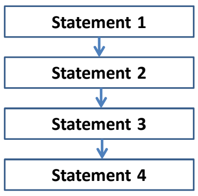
2

Linear execution of statements

- Each action performed in written order

What is the result of this set of statements?

```
int a=1, b=2, c;
a=5;
c = a+b;
cout << c;
```

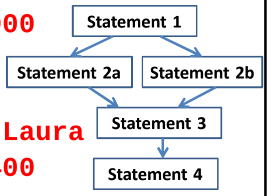


3

Alternatives to “linear execution”

Conditional actions

```
> ./myProgram
What is your name? Joe
What time is it? 0900
Good morning, Joe.
> ./myProgram
What is your name? Laura
What time is it? 1400
Good afternoon, Laura.
>
```

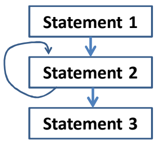


4

Alternatives to “linear execution”

Repeated actions

```
> ./myProgram
Infinite bottles of beer. Take one down.
Infinite bottles of beer. Take one down.
Infinite bottles of beer. Take one down.
Infinite bottles of beer. Take one down.
>
```



5

The `if-else` statement

- if-else is used to perform a two way branch

```
if ( condition )
   statement1;
else
   statement2;
```

- `statement1` is performed if `condition` is true
- `statement2` is performed if `condition` is false
- Only one of the two statements is performed!

6

`condition` – a Boolean expression

- Boolean expressions are either true or false
- Conditions often consist of **comparisons**
  - age ≥ 21 // can buy drinks
  - age < 4 // can ride subway for free
  - year = 2 // you are a sophomore

7

Comparisons in C++

=	equal to	==	a == b
≠	not equal to	!=	a != b
<	less than	<	a < b
≤	less than or equal to	<=	a <= b
>	greater than	>	a > b
≥	greater than or equal to	>=	a >= b

8

Be careful with =

= is the assignment operator
- `a=b;` assigns the value of b to a

== tests equivalence
- a==b determines if a and b have the same value

9

Multi-character comparisons

Where spaces matter:
- Correct: `a>=b` `a<=b` `a!=b`
- Incorrect: `a> =b` `a< =b` `a! =b`

No space between > and =, < and =, ! and =

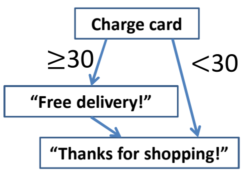
Where spaces don't matter:
- Correct: `a>=b` `a <=b` `a !=b`

10

if statement

Can write `if` statement without `else`

```
> ./myProgram
Enter charge: 32.00
Free delivery!
Thanks for shopping!
> ./myProgram
Enter charge: 10.00
Thanks for shopping!
>
```



11

Compound statements: the use of { }

- Must group multiple statements with { } in `if-else`

```
if ( condition )
{
   statement1;
   statement2;
   statement3;
}
else
{
   statement4;
   statement5;
}
```

12

What does this do?

```
int numBagels=5;

cout << "You are getting" << numBagels;
cout << " bagels!\n";

if ( numBagels>12 )
{
   numBagels++;
   cout << "You also get an extra bagel free!";
   cout << endl;
}

cout << "Have a good day.\n“;
```

13

What does this do?

```
int numBagels=5;

cout << "You are getting" << numBagels;
cout << " bagels!\n";

if ( numBagels>12 )
   numBagels++;
   cout << "You also get an extra bagel free!";
   cout << endl;

cout << "Have a good day.\n“;
```

14

Groups of statements

- White space (indents, extra blank lines) ignored by compiler … BUT
- White space is good programming style
- Visually groups statements together
- Braces { } create groups for compiler

15

Compound Boolean expressions

Expressions can be combined with logical operators

- The AND operator &&:
  `expression1 && expression2` true only if both `expression1` and `expression2` are true

`if ( ( 2<x ) && ( x<7 ) )`

- true only if x is between 2 and 7, e.g, x is 4, x is 6
- false otherwise, e.g., x is 0, x is 10
- Equivalently: `if ( 2<x && x<7 )`
- Invalid: `if ( 2<x<7 )`

16

Compound Boolean expressions

Expressions can be combined with logical operators

- The OR operator ||:
  `expression1 || expression2` true only if at least one of `expression1` and `expression2` are true

`if ( ( ageZoe==20 ) || ( ageZoe==25 ) )`

- true only if `ageZoe` is 20 or 25
- false otherwise
- Equivalently: `if ( ageZoe==20 || ageZoe==25 )`

17

Logical operators, continued

Expressions can be altered with logical operators

- The NOT operator !:
  `!expression` true only if `expression` is false

`if ( !( ageZoe>20 ) )`

- true only if `ageZoe` is below 20
- false otherwise
- Preferably: `if ( ageZoe<=20 )`
- Preferable to avoid `!expression`

18

In summary

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

a	!a
true	false
false	true

19

What does this code do?

```
#include<iostream>
using namespace std;
int main () {
   float soupTemp;

   cout << "What is your soup temperature? ";
   cin >> soupTemp;
   if ((soupTemp > 80) && (soupTemp<95))
      cout << "This soup is just right!\n";
   else
      cout << "This soup is no good!\n";
   return 0;
}
```

20

When do we need parentheses?

`(soupTemp > 80) && (soupTemp<95)`

is the same as

`soupTemp > 80 && soupTemp<95`

How about:

`(soupTemp > 80) && !(soupTemp>=95)`

vs.

`soupTemp > 80 && !soupTemp>=95`

21

Order of operations for logic

1. Parentheses: ( )
2. Negation: !
3. Comparison: <, >, <=, >=, ==, !=
4. And: &&
5. Or: ||

Operations on same level evaluated left-to-right

22

Order of operations in action

`int soupTemp=100;`

`(soupTemp > 80) && !(soupTemp>=95)`

vs.

`soupTemp > 80 && !soupTemp<95`

23

Cautionary notes

- Be careful using `!`, better to avoid it
- Remember `int`-to-`bool` conversion
  - `0` as `false`
  - `1` (or any non-zero number) as `true`

24

Short-circuit evaluations

- If the value of the leftmost sub-expression determines the value of the full expression, the rest of the expression is not evaluated

```
float x=0, y=20;
if ( x!=0 && y/x>=3 )  // only x!=0
                       // evaluated
{ . . .
}
if (y/x >= 3 && x!=0) // error
                      // divide-by-0
```

Different parts of the afternoon

Conditional actions

```
> ./myProgram
What is your name? Jill
What time is it? 1400
Good afternoon, Jill.
> ./myProgam
What is your name? Leon
What time is it? 2100
Good evening, Leon.
>
```

Nested ifs

```
if ( time > 1200)
   if (time < 1800)
      cout << "Good afternoon\n";
   else
      cout << "Good evening\n";
else
   cout << "Good morning\n";
```

Using const

Constant variables – replace numbers with meaningful names

```
const int noon=1200, startOfEve=1800;
if ( time > noon)
   if (time < startOfEve)
      cout << "Good afternoon\n";
   else
      cout << "Good evening\n";
else
   cout << "Good morning\n";
```

What does this code do?

```
// buying a laptop
int price=500;  // $500
float weight=50.5; // 50.5 pounds
if (weight<5.5)
   if (price<1000)
      cout << "Buy this!" << endl;
else
   cout << "Too heavy!" << endl;
```

Grouping of if and else

- else statement is connected with closest if
- Indentation ignored by compiler!
- { } braces instruct the compiler for grouping

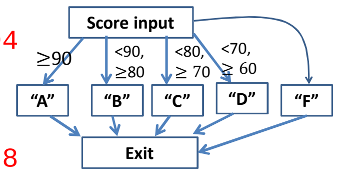
Multiway `if-else` statement

Actions for multiple mutually-exclusive conditions

```
if ( expression1)
   statement1;
else if ( expression2 )
   statement2;
. . .
else if ( expressionN )
   statementN;
else  // all above expressions false
   statementLast;
```

Branching on grade

```
> ./myProgram
Enter score: 94
You get an A.
> ./myProgram
Enter score: 78
Your get a C
```



Scope

- Variables declared inside a block are not "visible" outside the block
- Variables declared in an outer block are visible to inner blocks
- Blocks are enclosed by braces { }

What does this code do?

```
int main () {
   int a=5, b=10;
   if ( a >= 3) {
      int a=8;
      cout << a << " " << b << endl;
   }
   cout << a << " " << b << endl;
   return 0;
}
```

What does this code do?

```
int main () {
   int a=5, b=10;
   if ( a >= 3) {
      int a=8, c=5;
      cout << a << " " << b << endl;
   }
   cout << a << " " << c << endl;
   return 0;
}
```

What does this code do?

```
int main () {
   int a=5, b=10;
   if ( a >= 3) {
      int a=8, c=5;
      b=12;
      cout << a << " " << b << endl;
   }
   cout << b << " " << c << endl;
   return 0;
}
```

Multiway `switch` statement

`switch` picks which statements to perform based on value of `controlStatement`

```
switch ( controlStatement )
{
   . . .
   case constantX :
      statementSequenceX
      break;
   . . .
}
```

Full `switch` syntax

```
switch ( controlStatement )
{
   case constant1 :
      statementSequence1
      break;
   . . .
   case constantN :
      statementSequence3
      break;
   default :
      statementSequence
}
```

`controlStatement`

Must return a value of type:

- `bool`
- integer (`int`, and related types)
- `char`

`case` statement

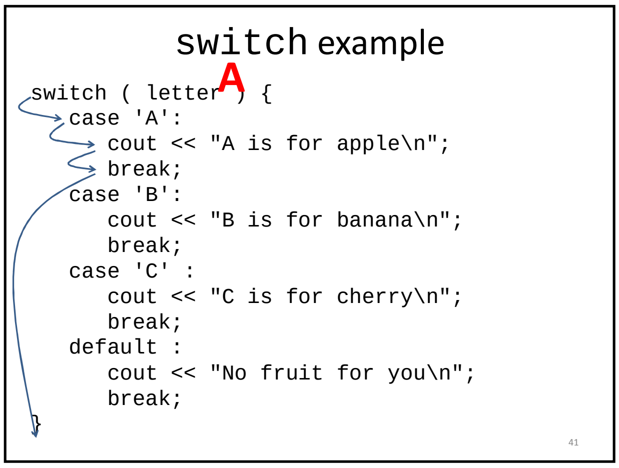
`case constantX :` tells program to start running following code if `controlStatement` has given value

`break` statement

`break;` exits the current block of code

`switch` example

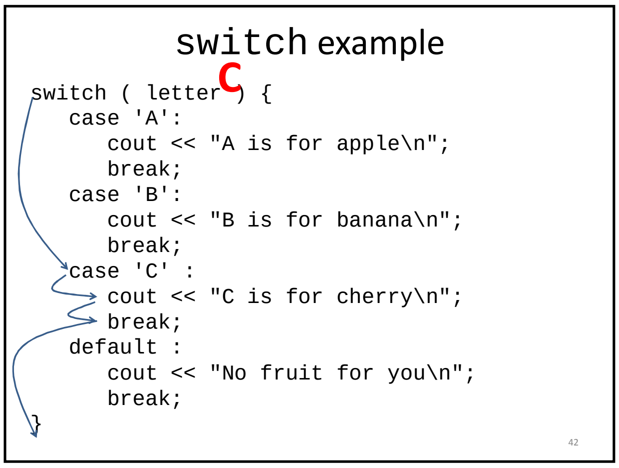
```
switch ( letter ) {
   case 'A':
      cout << "A is for apple\n";
      break;
   case 'B':
      cout << "B is for banana\n";
      break;
   case 'C' :
      cout << "C is for cherry\n";
      break;
   default :
      cout << "No fruit for you\n";
      break;
}
```



`switch` example

A

```
switch ( letter ) {
   case 'A':
      cout << "A is for apple\n";
      break;
   case 'B':
      cout << "B is for banana\n";
      break;
   case 'C' :
      cout << "C is for cherry\n";
      break;
   default :
      cout << "No fruit for you\n";
      break;
}
```



`switch` example

C

```
switch ( letter ) {
   case 'A':
      cout << "A is for apple\n";
      break;
   case 'B':
      cout << "B is for banana\n";
      break;
   case 'C' :
      cout << "C is for cherry\n";
      break;
   default :
      cout << "No fruit for you\n";
      break;
}
```

Can omit `break` statements to group conditions

```
switch ( letter ) {
   case 'A':
   case 'a':
      cout << "A is for apple\n";
      break;
   case 'B':
   case 'b':
      cout << "B is for banana\n";
      break;
   case 'C' :
   case 'c' :
      cout << "C is for cherry\n";
      break;
   default :
      cout << "No fruit for you\n";
      break;
}
```

43