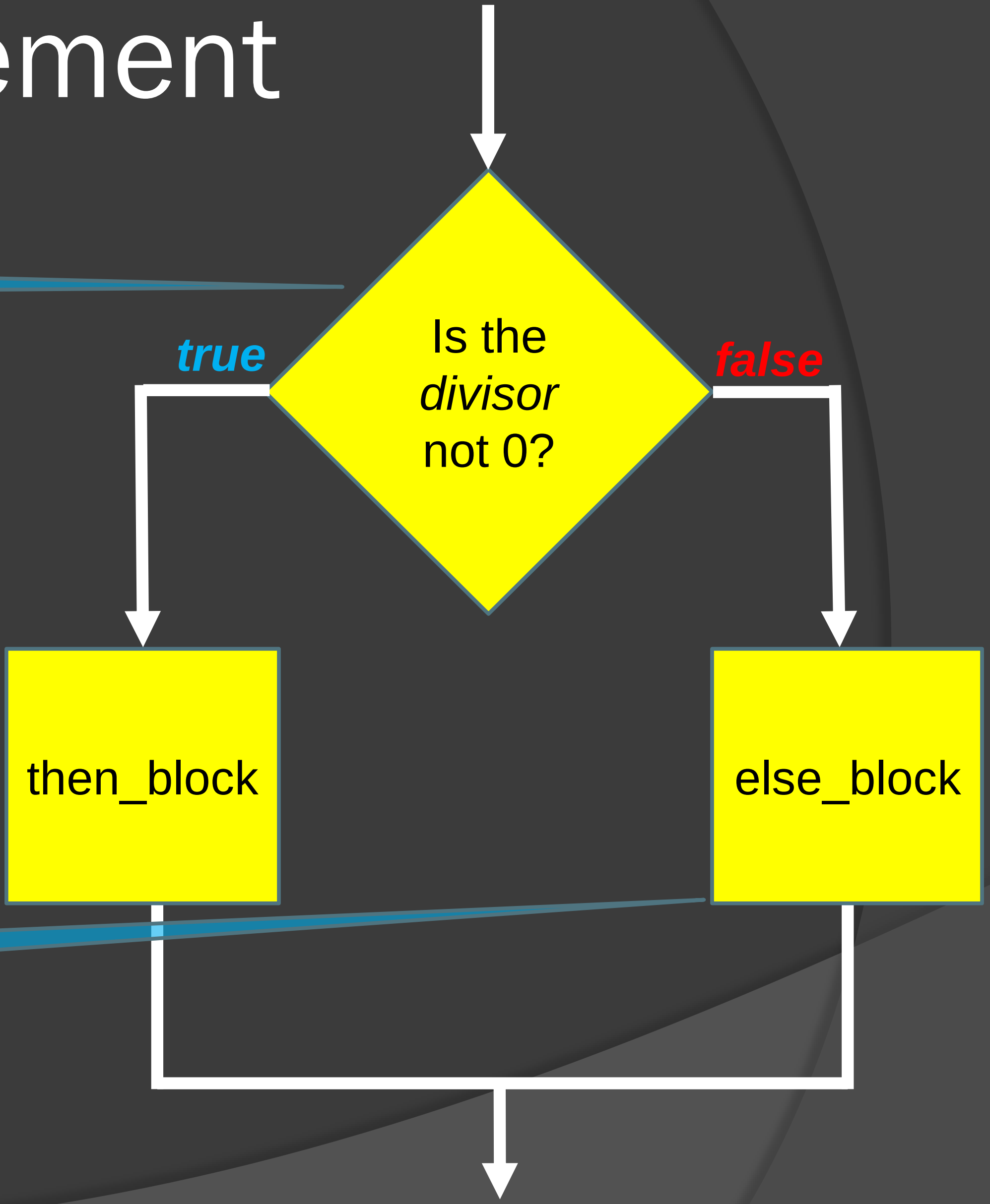
SELECTION STRUCTURES CONTINUED…

if-else statement

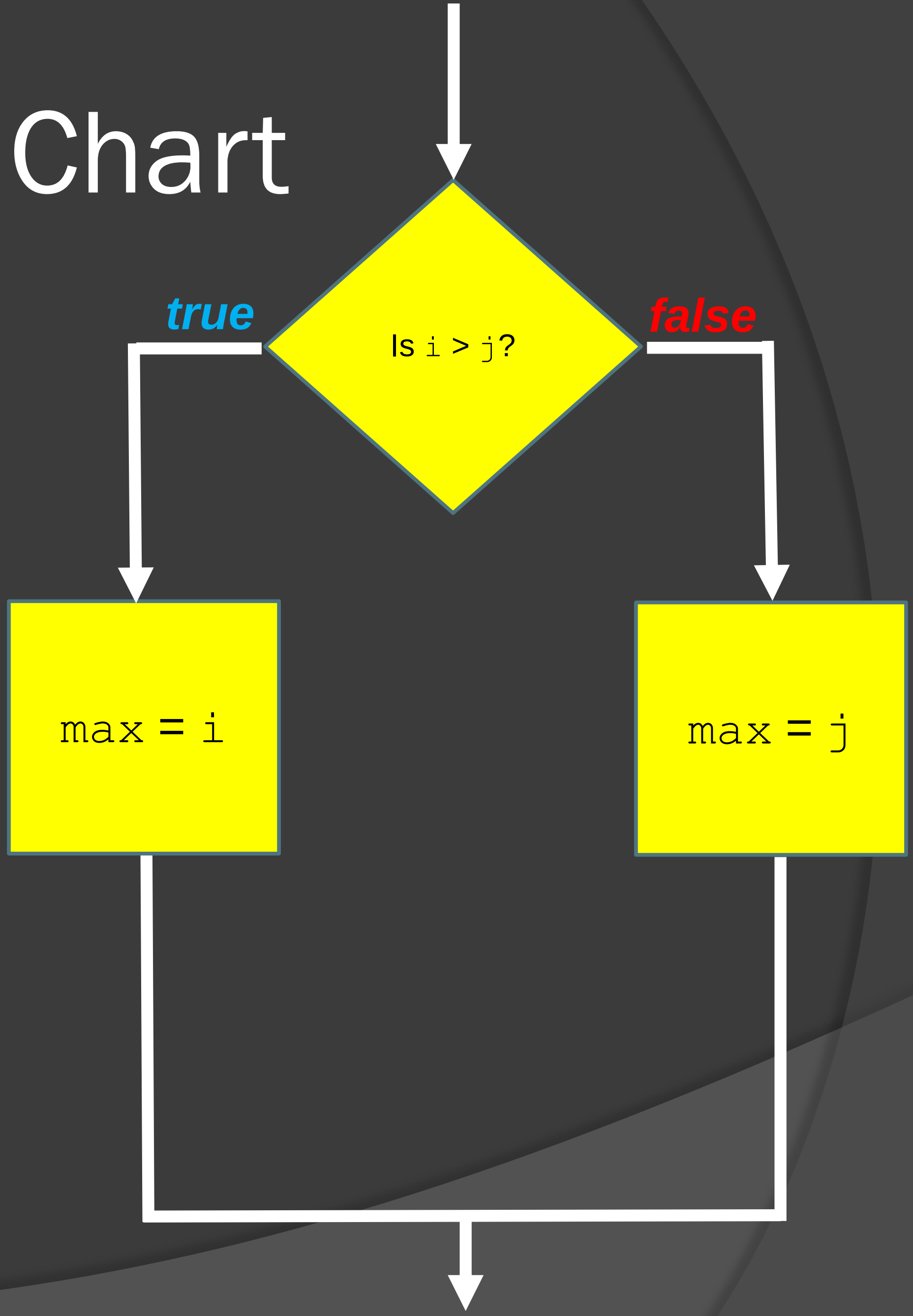
```
if (test)
{
    then_block
}
else
{
    else_block
}
```



Your Turn (I'll do with you)

- Given two integers `i` and `j` input from the user, write C++ code that assigns the integer variable `max` to the **larger** of `i` and `j`
  - If `i` is 4 and `j` is 10, then assign `max` to 10
  - If `i` is 22 and `j` is 16, then assign `max` to 22
- Draw a flow chart first?

Your Turn: Flow Chart



What if i and j are the **same**?

Your Turn

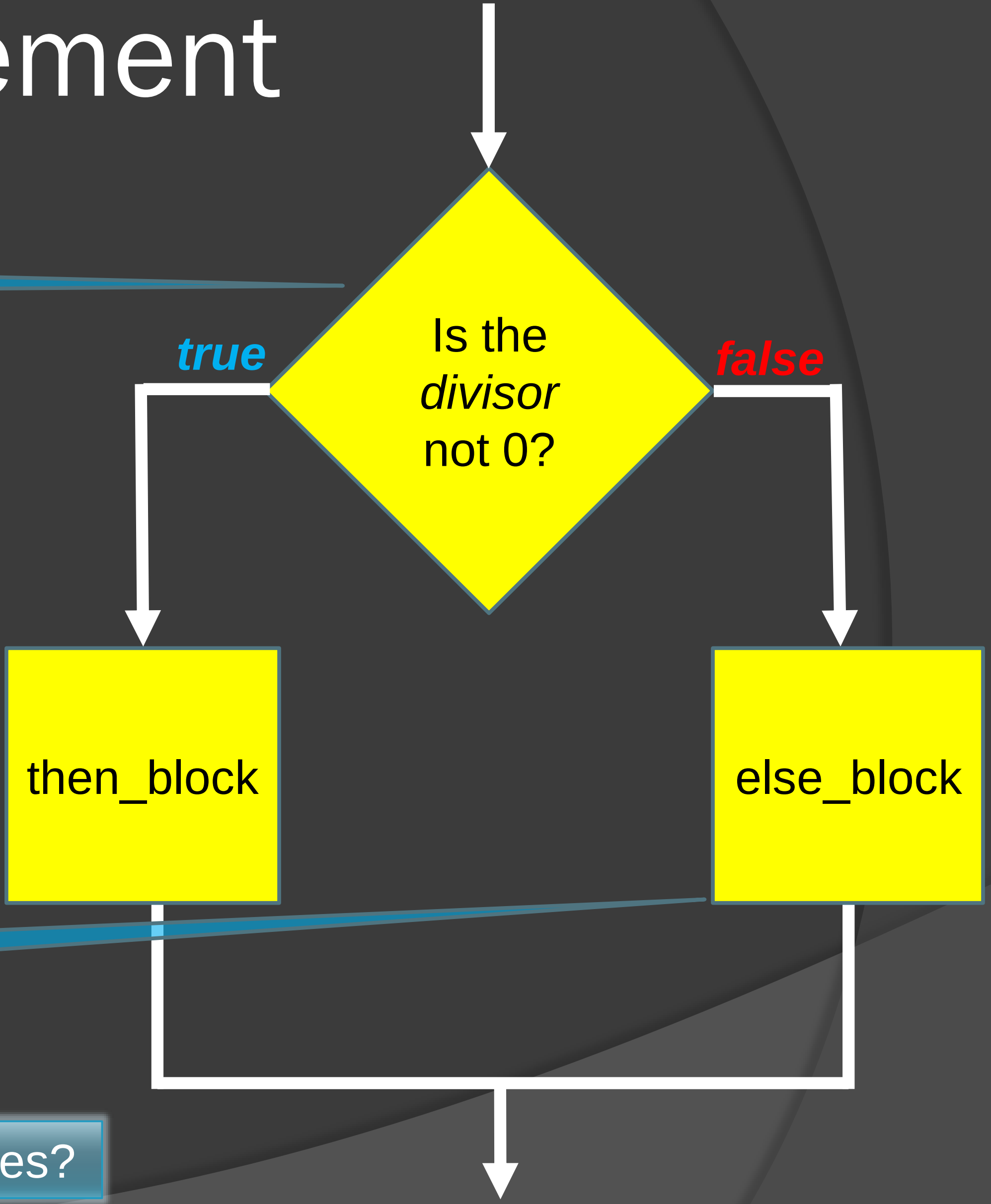
```
if (i > j)
{
    max = i;
}
else
{
    max = j;
}
```

```
max = i;
if (j > max)
{
    max = j;
}
```

- *Is the* `else` *needed here?*

if-else statement

```
if (test)
{
    then_block
}
else
{
    else_block
}
```



What if you want more alternatives?

Many alternatives

- The variable `age` contains a person’s age:
  - If 1 ≤ `age` ≤ 12, print “You are a child”
  - If 13 ≤ `age` ≤ 19, print “You are a teen”
  - If 20 ≤ `age` ≤ 39, print “You are getting old”
  - If 40 ≤ `age`, print “You are getting over the hill”
- How many `if-else` statements do we need?

How about?

This is undesirable!

```
if (1 <= age && age <= 12)     // Line 1
{
   cout << “You are a child” << endl;
}
if (13 <= age && age <= 19)    // Line 2
{
   cout << “You are a teenager” << endl;
}        Backwards
if (age <= 20 && age <= 39)    // Line 3
{
   cout << “You are getting old” << endl;
}
if (40 <= age)                 // Line 4
{
   cout << “You are over the hill” << endl;
}
```

Mutual Exclusion

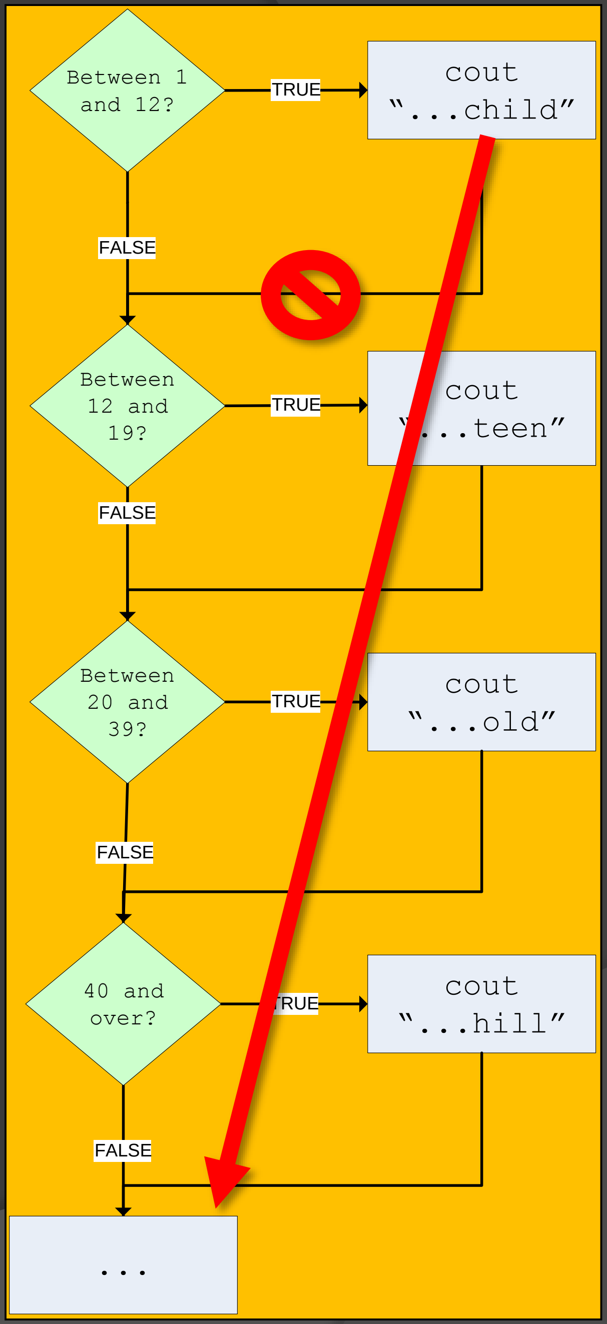
- Given variable `age`, only ONE `if` statement will execute its code block
  - None of the others will execute their code block

  - If 1 ≤ `age` ≤ 12, print "You are a child"
  - If 13 ≤ `age` ≤ 19, print "You are a teen"
  - If 20 ≤ `age` ≤ 39, print "You are getting old"
  - If 40 ≤ `age`, print "You are getting over the hill"

- E.g., if the variable `age` is 6 then the first `if` statement will be satisfied and the rest of the `if` statements will be checked but not execute their code block, which is simply a waste!

Mutual Exclusion

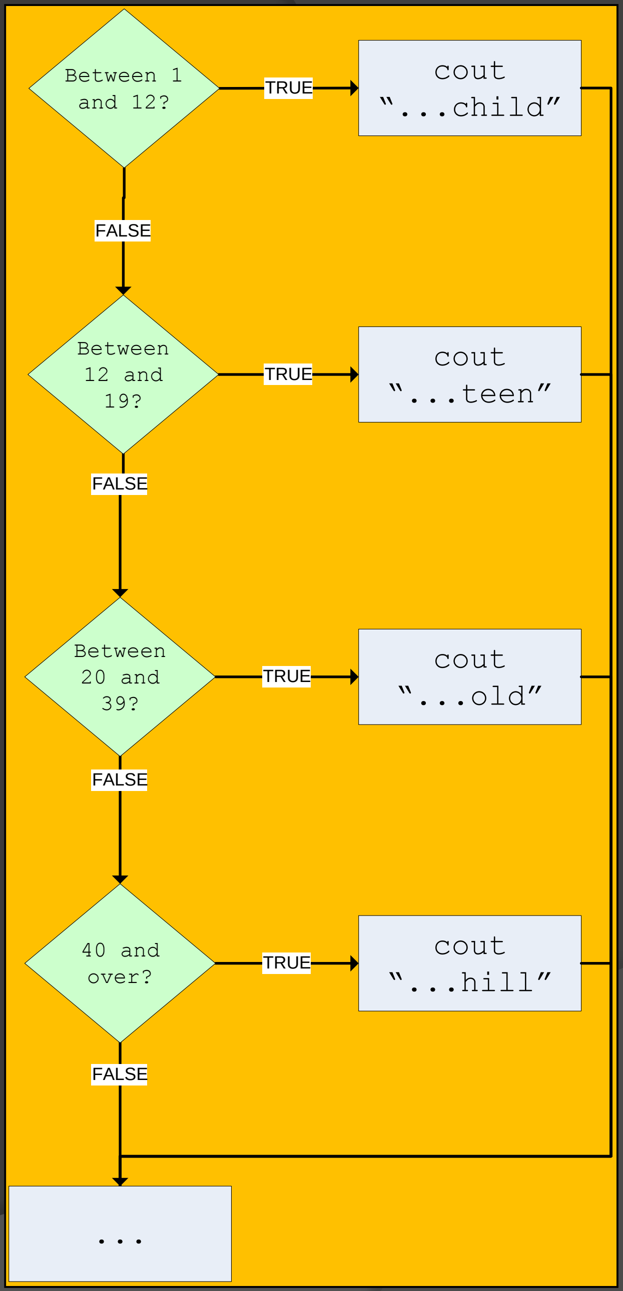
- Once one condition is satisfied why bother checking the remaining conditions?
  - If the age is a child then skip past all of the remaining if statements
- Otherwise, it is a waste of time



Mutual Exclusion

- Only check next condition if *the previous conditions all failed so far*

- Once a *condition is true*:
  - Execute its corresponding code block
  - Exit out of the selection structure



The **else-if**

- Remember: ***Each else statement belongs to a matching if statement***

- An `else` keyword may be followed immediately by an `if` statement, called an **`else-if`**

  - This *properly* handles *mutual exclusion*
  - There is no limit to the number of `else-if`’s you chain together

Example

- Given a variable `temp` with a temperature value, report
  - "cold", if the temperature is below 32 degrees
  - "nice", if the temperature is between 32 and 80 (inclusive of 32)
  - "hot", if the temperature is 80 or over

else-if example

- Three *mutually exclusive* alternatives using the **else-if**

```
if (temp < 32)
{
   cout << "It is cold" << endl;
}
else if ~~(temp >= 32~~ ~~&&~~ temp < 80) // Inefficient!!
{
   cout << "It is nice" << endl;
}
else ~~if~~ ~~(temp >= 80)~~               // Unnecessary!!
{
   cout << "It is hot" << endl;
}
```

else-if example

- Three *mutually exclusive* alternatives using the **else-if**

```
if (temp < 32)
{
   cout << "It is cold" << endl;
}
else if (temp < 80) // Here, we already know temp ≥ 32
{
   cout << "It is nice" << endl;
}
else                   // Here, we already know temp ≥ 80
{
   cout << "It is hot" << endl;
}
```

ifExample.cpp

```
...
int main()
{
  int a(0), b(0), c(0);

  cout << "Enter a, b, c: ";
  cin >> a >> b >> c;

  if (a < b)
  {
    if (b < c) { cout << "b < c" << endl; }
    else { cout << "b >= c" << endl; }
  }
  else if (a < c)
  { cout << "a < c" << endl; }
  else
  { cout << "a >= c" << endl; }

  return 0;
}
```

What is the output on input:

```
1 2 3

3 2 1

1 3 2

2 1 3

3 1 2

2 3 1
```

Your Turn (I'll do this one with you)

- Given three integers `i` , `j`, and `k` input from the user, write C++ code that assigns the integer variable `max` to the **larger** of `i` , `j`, and `k`

Possible Solution

```
...
int main()
{
  int i(0), j(0), k(0), max;

  cout << "Enter i, j, k: ";
  cin >> i >> j >> k;

  max = i;
  if (j > max)
    max = j;
  else if (k > max)
    max = k;

  . . .

  return 0;
}
```

Your Turn

- Given a the value in the variable `age`, write C++ code that outputs the correct response. Use mutual exclusion.

  - If `age` ≤ 14, print “You are too young to drive”
  - If `age` is 15, print “You can get a learner’s permit”
  - If `age` is between 16 and 25 (inclusive), print “You pay more for insurance”
  - If `age` is over 25, print “You can drive”

The `switch` Statement

- The switch statement is an alternative to the **`if-else`** chain (but not in all circumstances)

```
switch (expression)
{
   case constant-value1:
       ...
   case constant-value2:
       ...
   default: // optional
       ...
}
```

(See zyBook Text for more information)

CERR AND EXIT()

Error Handling

- The **if** and **if-else** statements are used to detect and handle errors

```
if (divisor == 0)
{
    // Stop the program, we cannot go any further!
}
quotient = dividend/divisor;
```

Error Handling

- The **`exit`** function quits the program immediately

```
if (divisor == 0)
{
    cout << "Cannot complete division" << endl;
    exit(10); // The value 10 is returned
              // to the operating system
}
quotient = dividend/divisor;
```

The **`exit`** function

- You need to indicate its library:

```
#include <cstdlib>
```

- To help you debug, use a different number with each exit statement
  - Keep track of your numbers by matching them up with the errors they represent

```
exit(10);
exit(20);
exit(30);
```

The `cerr` statement

- The `cout` statement sends output to the user’s window
    - Related to the *stdout* “stream” (standard out)
- But what if we want to send the message to another location, e.g. a file?
    - You may want to look at it later

```
if (divisor == 0)
{
    cout << "Cannot complete division" << endl;
    exit(10);
}
quotient = dividend/divisor;
```

The `cerr` statement

- Use `cerr` when you want to report your error messages
    - Related to the *stderr* “stream” (standard error)

```
if (divisor == 0)
{
    cerr << "Cannot complete division" << endl;
    exit(10);
}
quotient = dividend/divisor;
```

cerrExample.cpp

```
#include <cstdlib>   // File cstdlib contains exit()
...
int main()
{
  double x(0.0);

  cout << "Enter non-negative value: ";
  cin >> x;

  if (x < 0)
  { // Use cerr instead of cout.  Use exit instead of return.
    cerr << "Error: Illegal negative value: " << x << endl;
    exit(20);
  }

  cout << "sqrt(" << x << ") = " << sqrt(x) << endl;

  return 0;
}
```

cerrExample2.cpp

```
...
int main()
{
  double x(0.0);

  cout << "Enter non-negative value: ";
  cin >> x;

  if (x < 0)
  { // Change to default value.
    cerr << "Warning: Illegal negative value: " << x << endl;
    cerr << "Changing " << x << " to " << -x << endl;

    x = -x;
  }

  cout << "sqrt(" << x << ") = " << sqrt(x) << endl;

  return 0;
}
```

Error Handling

- **`cerr`** instead of `cout`
  - Messages can be sent to a different place than `cout`
  - Forces messages to be printed immediately
- Function **`exit`** instead of `return`
  - Quits the program and returns control to the operating system
  - Frees up resources associated with the program
  - “return” returns control to any calling program/function