Still more choices with Mr C

ESC101: Fundamentals of Computing

Purushottam Kar

Announcements

- Institute holiday on August 22, 2018, Wednesday
    - No lecture, no lab on August 22
    - No extra lecture this week
- Extra lab for Wednesday batches B10, B11, B12, B14
    - Saturday, August 25, 2018, 2PM New Core Labs CC-01, CC-02
- Refer to course schedule calendar on website
  web.cse.iitk.ac.in/users/purushot/courses/esc/2018-19-a/material/schedule.pdf

Announcements

- Extra session for students facing trouble with English lectures but who are comfortable with Hindi
  - Saturday, August 25, 2018, 5PM-6:30PM, New Core Labs CC-02
  - Extra session to be held just after extra lab is over for B10, B11, B12, B14
- Students familiar with other Indian languages, please refer to document on website for names of admins
  web.cse.iitk.ac.in/users/purushot/courses/esc/2018-19-a/material/language.pdf

Announcements

- Major quiz next week – (syllabus till **Friday Aug 24**)
  - Wednesday, August 29, 2018, 12PM-12:50PM, L20 (i.e. lecture hour)
  - During lecture hours – don't be absent
  - **Bring your institute ID card** with you – will lose time if you forget
  - No minor quizzes during lab next week (August 27-August 30)
- Bring a **pencil, eraser and sharpener** with you
  - Answers to be written on question paper itself and returned back
  - If you make a mistake with pen – no extra question papers
  - If unsure, **first write answer with pencil** and **finally write it in pen**
  - We WONT HAVE EXTRA QUESTION PAPERS in case you spoil yours
  - We WONT HAVE PENCILS, ERASERS in case you forget

Revision – the two shades of if

Revision – the two shades of if

```
if( ... ){
    ...
}
```

Revision – the two shades of if

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

Revision – the two shades of if

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

Can put one or more statements here

Revision – the two shades of if

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

Can put one or more statements here

Can put one or more statements here

Revision – the two shades of if

```
if( … ){
    …
}
```

Can put one or more statements here

```
if( … ){
    …
}else{
    …
}
```

Can put one or more statements here

Can put one or more statements here

Revision – the two shades of if

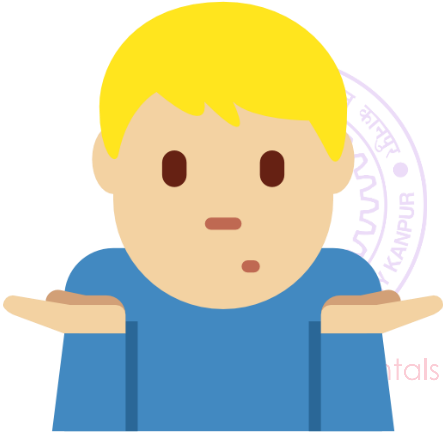
```
if( ... ){
    ...
}
```

Can put one or more statements here

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

Can put one or more statements here

Can put one or more statements here



Revision – the two shades of if

```
if( ... ){
    ...
}
```

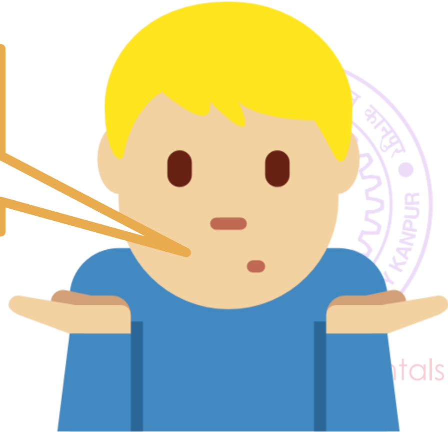
Can put one or more statements here

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

Can put one or more statements here

Can put one or more statements here

What all are valid statements?



Revision – the two shades of if

```
if( … ){
    …
}
```

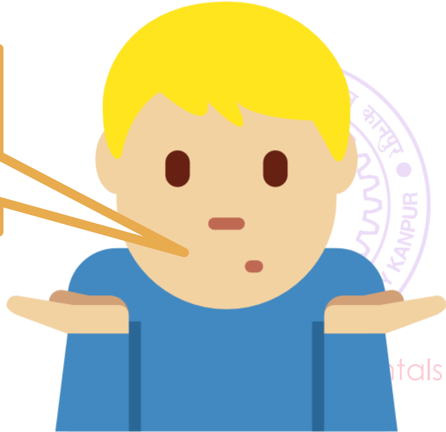
Can put one or more statements here

```
if( … ){
    …
}else{
    …
}
```

Can put one or more statements here

Can put one or more statements here

What all are valid statements?



Revision – the two shades of if

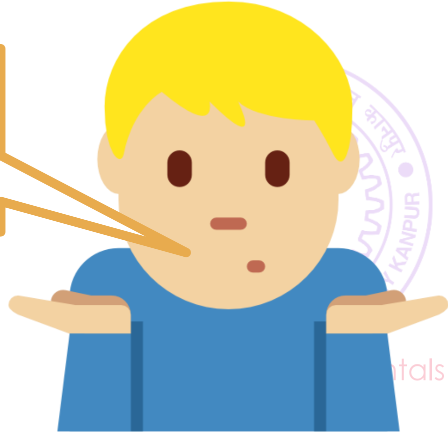
```
if( ... ){
    ...
}
```

Can put one or more statements here

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

Can put one or more statements here

Can put one or more statements here



What all are valid statements?

Revision – the two shades of if

```
if( ... ){
    ...
}
```

Can put one or more statements here

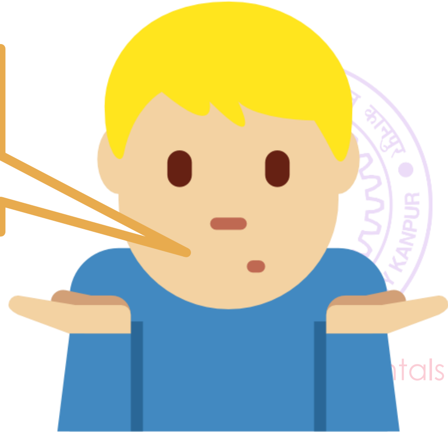
```
if( ... ){
    ...
}else{
    ...
}
```

Can put one or more statements here

Can put one or more statements here

- int a = 3, b;

What all are valid statements?



Revision – the two shades of if

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

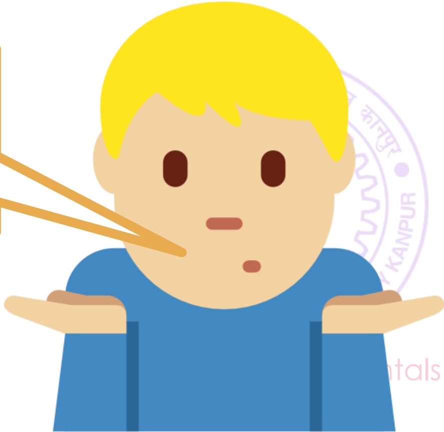
Can put one or more statements here

Can put one or more statements here

Can put one or more statements here

- int a = 3, b;
- b = a++;

What all are valid statements?



Revision – the two shades of if

```
if( … ){
    …
}
```

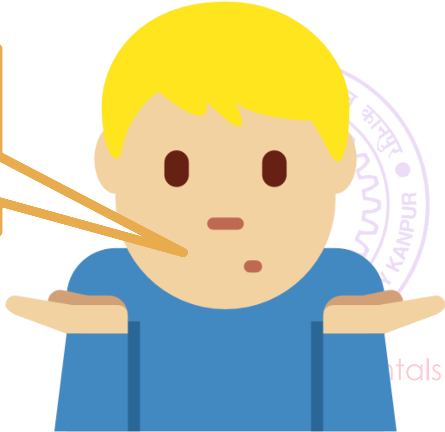
Can put one or more statements here

```
if( … ){
    …
}else{
    …
}
```

Can put one or more statements here

Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf(“%d”, b);



What all are valid statements?

Revision – the two shades of if

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);

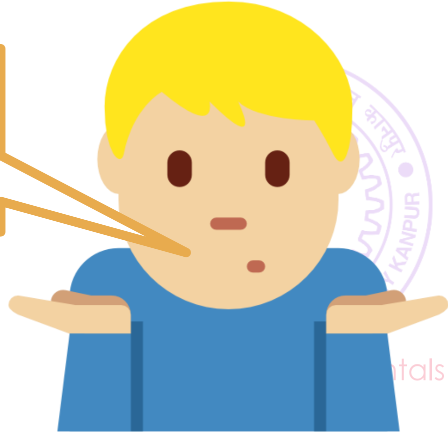
```
if( ... ){
    ...
}
```

Can put one or more statements here

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

Can put one or more statements here

Can put one or more statements here



What all are valid statements?

Revision – the two shades of if

```
if( ... ){
    ...
}
```

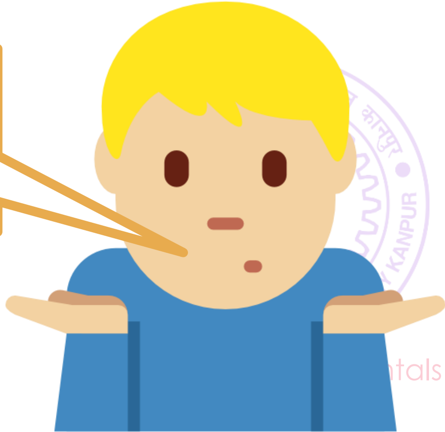
Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);
- etc. etc.

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

Can put one or more statements here

Can put one or more statements here



What all are valid statements?

Revision – the two shades of if

```
if( … ){
    …
}
```

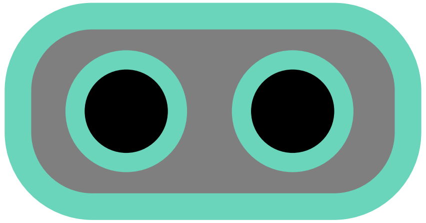
Can put one or more statements here

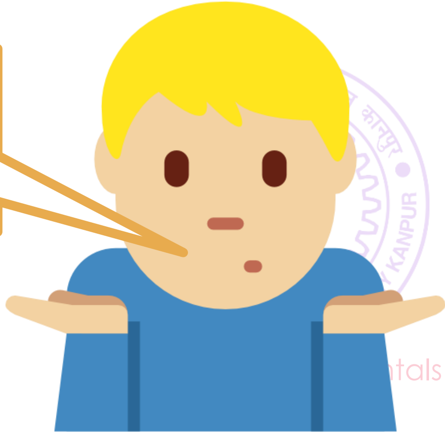
```
if( … ){
    …
}else{
    …
}
```

Can put one or more statements here

Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);
- etc. etc.





What all are valid statements?

Revision – the two shades of if

```
if( … ){
    …
}
```

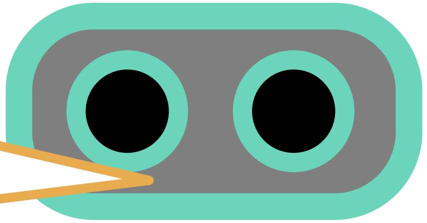
Can put one or more statements here

```
if( … ){
    …
}else{
    …
}
```

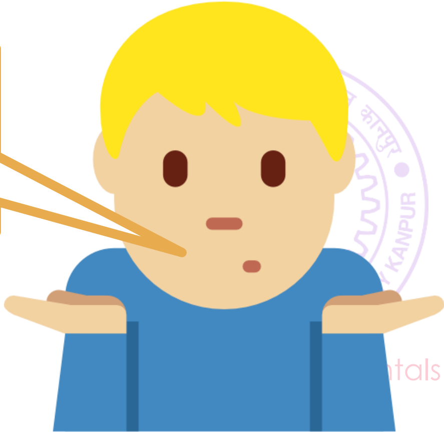
Can put one or more statements here

Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);
- etc. etc.



Don’t forget
if(…){ … } is itself a
single valid statement



What all are valid statements?

Revision – the two shades of if

```
if( … ){
    …
}
```

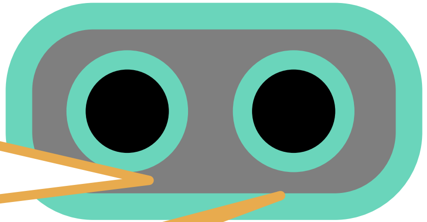
Can put one or more statements here

```
if( … ){
    …
}else{
    …
}
```

Can put one or more statements here

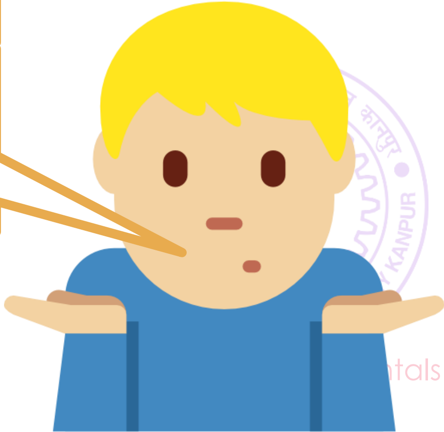
Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf("%d", b);
- scanf("%f", &r);
- etc. etc.



Don't forget if(…){ … } is itself a single valid statement

if(…){ … }else{ … } is also a single valid statement



What all are valid statements?

Revision – the two shades of if

```
if( ... ){
    ...
}
```

Can put one or more statements here

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

Can put one or more statements here

Can put one or more statements here

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);
- etc. etc.

Don’t forget if(…){ … } is itself a single valid statement

if(…){ … }else{ … } is also a single valid statement

What all are valid statements?

Yes, we can nest if-else statements

Revision – the tw

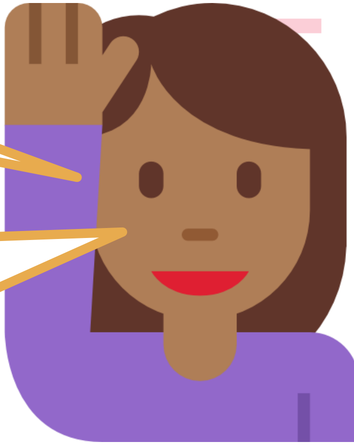
```
if( ... ){
    ...
}
```

Can put one or more statements here

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

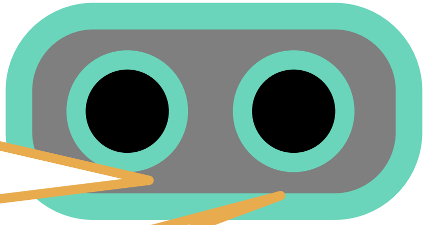
Can put one or more statements here

Can put one or more statements here



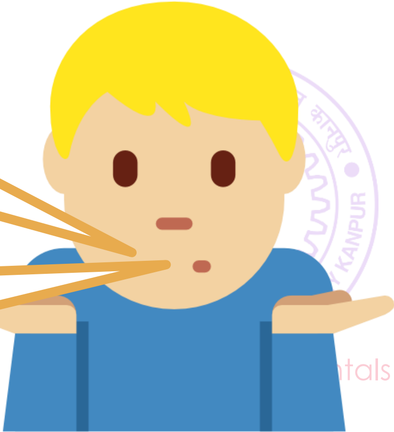
Oops, thanks for the reminder ☺

- int a = 3, b;
- b = a++;
- printf(“%d”, b);
- scanf(“%f”, &r);
- etc. etc.



Don’t forget if(…){ … } is itself a single valid statement

if(…){ … }else{ … } is also a single valid statement



What all are valid statements?

Yes, we can nest if-else statements

Finding the smaller of two numbers

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

```
int a = 5, b = 3, min;
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

```
int a = 5, b = 3, min;
min = (a < b)? a : b;
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf("Minimum is %d",min);
```

```
int a = 5, b = 3, min;
min = (a < b)? a : b;
printf("Minimum is %d",min);
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

Called a *ternary conditional* operator

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

```
int a = 5, b = 3, min;
min = (a < b)? a : b;
printf(“Minimum is %d”,min);
```

Finding the smaller of two numbers

Will learn a cute shortcut for very simple if-else statements

Called a *ternary conditional* operator

Just a shortcut, can be implemented exactly using if-else

```
int a = 5, b = 3, min;
if(a < b)
    min = a;
else
    min = b;
printf(“Minimum is %d”,min);
```

```
int a = 5, b = 3, min;
min = (a < b)? a : b;
printf(“Minimum is %d”,min);
```

Ternary Conditional expression

Ternary Conditional expression

General form

Ternary Conditional expression

General form

(relational expression)? expression1 : expression2

Ternary Conditional expression

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

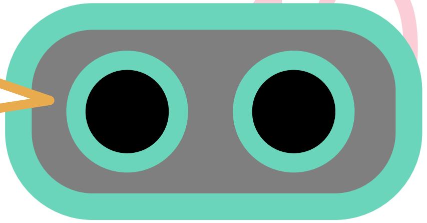
Ternary Conditional expression

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Ternary Conditional ex



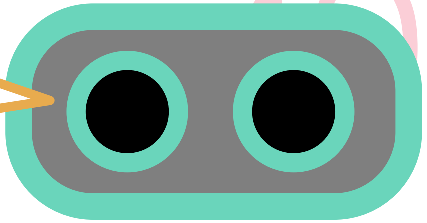
All expressions generate values

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Ternary Conditional ex



All expressions generate values

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

Ternary Conditional expression

All expressions generate values

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

Ternary

Good idea to put brackets
Less confusion, less chance
of error, easy to read

All expressions
generate values

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

Ternary

Good idea to put brackets
Less confusion, less chance
of error, easy to read

All expressions
generate values

General form

(relational expression)? expression1 : expression2

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

```
if((t >= 22) && (t <= 27))
    c = t + 1;
else
    c = t + 2;
```

Ternary O xpression

Good idea to put brackets
Less confusion, less chance
of error, easy to read

All expressions
generate values

General form

(relational expressio 2

Exactly same output

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

```
if((t >= 22) && (t <= 27))
    c = t + 1;
else
    c = t + 2;
```

Ternary O x

Good idea to put brackets
Less confusion, less chance
of error, easy to read

All expressions
generate values

General form

(relational expression)? expressio 2

Exactly same output

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

```
if((t >= 22) && (t <= 27))
    c = t + 1;
else
    c = t + 2;
```

Ternary

Good idea to put brackets
Less confusion, less chance of error, easy to read

All expressions generate values

General form

(relational expression)? expressio 2

Exactly same output

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

```
if((t >= 22) && (t <= 27))
    c = t + 1;
else
    c = t + 2;
```

expression1 and expression2 can be arithmetic, relational or even ternary (nested ternary)

Ternary C x

Good idea to put brackets
Less confusion, less chance of error, easy to read

All expressions generate values

General form

(relational expression)? expressio 2

Exactly same output

If relational expression evaluates to true (1 or non-zero) then the value of expression1 is calculated and generated, otherwise value of expression 2 is calculated, generated

Usually used in a statement along with assignment step

```
c = ((t >= 22) && (t <= 27))? (t + 1) : (t + 2);
```

```
if((t >= 22) && (t <= 27))
    c = t + 1;
else
    c = t + 2;
```

expression1 and expression2 can be arithmetic, relational or even ternary (nested ternary)

Avoid deep nesting – don't want shortcut to cause more difficulty

BODMAS table has more members

B embers

Operator Name	Symbol/Sign	Associativity
Bracket, Post increment/decrement	(), ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	Right
Multiplication/division/ remainder	*, /, %	Left
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	**? :**	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right

B embers

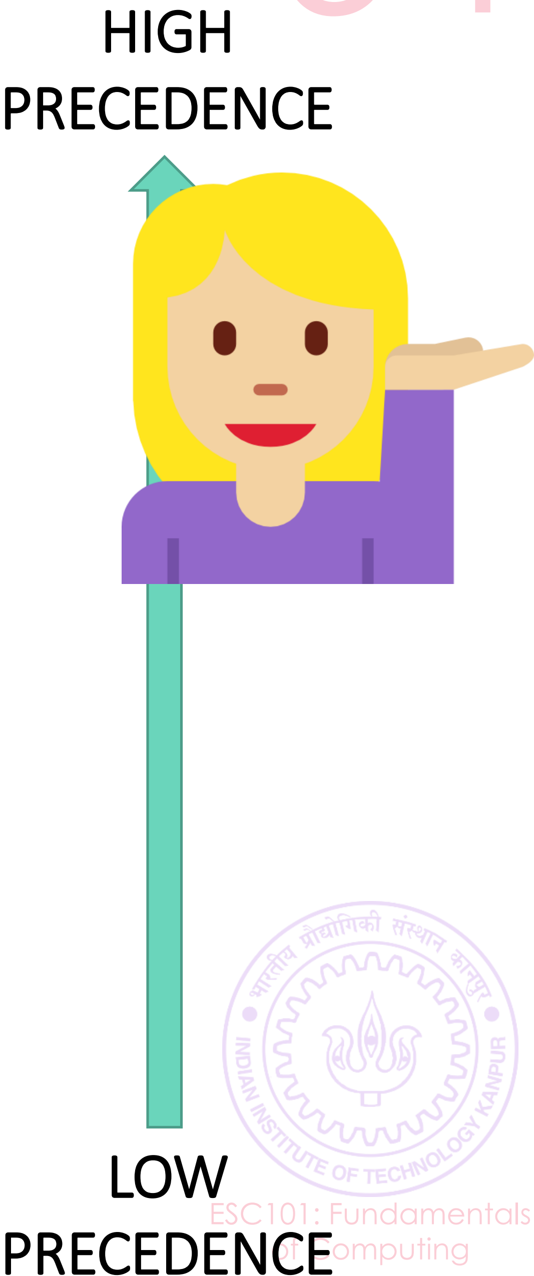
Operator Name	Symbol/Sign	Associativity
Bracket, Post increment/decrement	(), ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	Right
Multiplication/division/ remainder	*, /, %	Left
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	**? :**	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right

HIGH PRECEDENCE

LOW PRECEDENCE

B embers

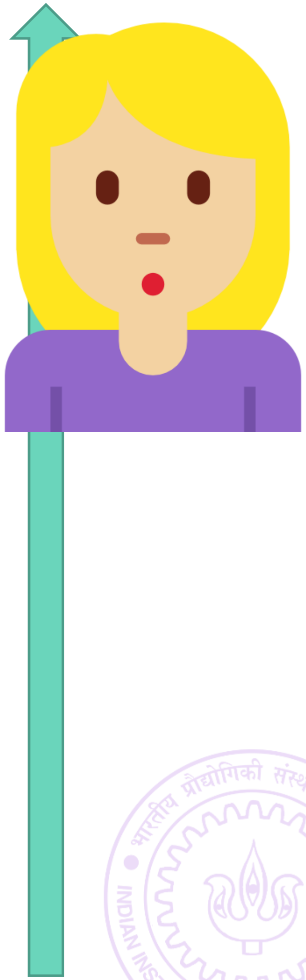
Operator Name	Symbol/Sign	Associativity
Bracket, Post increment/decrement	(), ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	Right
Multiplication/division/ remainder	*, /, %	Left
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	**? :**	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right



B embers

Operator Name	Symbol/Sign	Associativity
Bracket, Post increment/decrement	(), ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	Right
Multiplication/division/ remainder	*, /, %	Left
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	**? :**	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right

HIGH PRECEDENCE

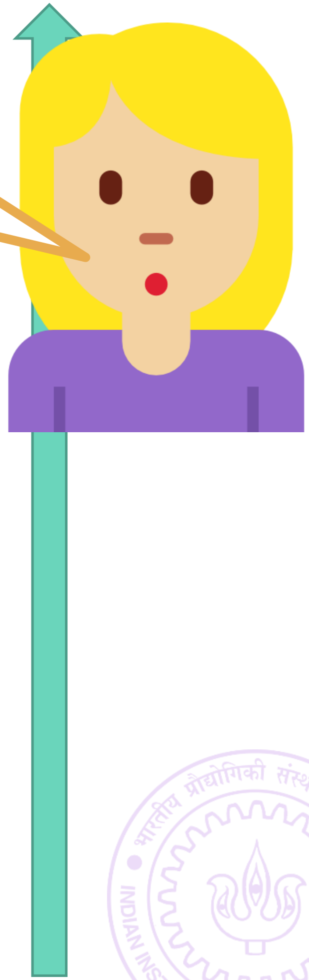


LOW PRECEDENCE

B embers

Operator Name	Symbol/Sign	Associativity
Bracket, Post increment/decrement	(), ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	
Multiplication/division/ remainder	*, /, %	
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	**? :**	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right

HIGH PRECEDENCE



LOW PRECEDENCE

Print the name of day of the week

Print the name of day of the week

Print the name of day of the week

```
if(n == 1)
```

Print the name of day of the week

```
if(n == 1)

    printf(“Monday”);
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
```

Print the name of day of the week

```
if(n == 1)
   printf(“Monday”);
else if(n == 2)
   printf(“Tuesday”);
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
    printf(“Tuesday”);
else if(n == 3)
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
    printf(“Tuesday”);
else if(n == 3)
    printf(“Wednesday”);
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
    printf(“Tuesday”);
else if(n == 3)
    printf(“Wednesday”);
else if(n == 4)
```

Print the name of day of the week

```
if(n == 1)
   printf(“Monday”);
else if(n == 2)
   printf(“Tuesday”);
else if(n == 3)
   printf(“Wednesday”);
else if(n == 4)
   printf(“Thursday”);
```

Print the name of day of the week

```
if(n == 1)
   printf(“Monday”);
else if(n == 2)
   printf(“Tuesday”);
else if(n == 3)
   printf(“Wednesday”);
else if(n == 4)
   printf(“Thursday”);
else if(n == 5)
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
    printf(“Tuesday”);
else if(n == 3)
    printf(“Wednesday”);
else if(n == 4)
    printf(“Thursday”);
else if(n == 5)
    printf(“Friday”);
```

Print the name of day of the week

```
if(n == 1)
   printf(“Monday”);
else if(n == 2)
   printf(“Tuesday”);
else if(n == 3)
   printf(“Wednesday”);
else if(n == 4)
   printf(“Thursday”);
else if(n == 5)
   printf(“Friday”);
else if(n == 6)
```

Print the name of day of the week

```
if(n == 1)
    printf(“Monday”);
else if(n == 2)
    printf(“Tuesday”);
else if(n == 3)
    printf(“Wednesday”);
else if(n == 4)
    printf(“Thursday”);
else if(n == 5)
    printf(“Friday”);
else if(n == 6)
    printf(“Saturday”);
```

Print the name of day of the week

```
if(n == 1)
   printf(“Monday”);
else if(n == 2)
   printf(“Tuesday”);
else if(n == 3)
   printf(“Wednesday”);
else if(n == 4)
   printf(“Thursday”);
else if(n == 5)
   printf(“Friday”);
else if(n == 6)
   printf(“Saturday”);
else if(n == 7)
```

Print the name of day of the week

```
if(n == 1)
    printf("Monday");
else if(n == 2)
    printf("Tuesday");
else if(n == 3)
    printf("Wednesday");
else if(n == 4)
    printf("Thursday");
else if(n == 5)
    printf("Friday");
else if(n == 6)
    printf("Saturday");
else if(n == 7)
    printf("Sunday");
```

Print the name of day of the week

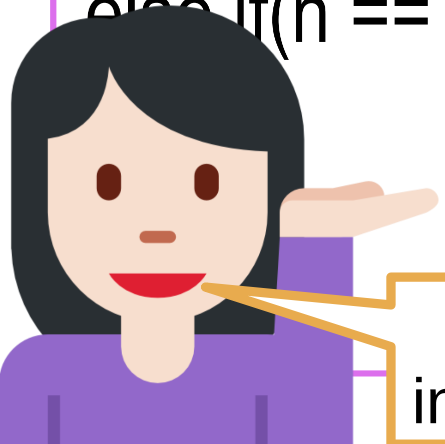
```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```



Sometimes not indenting looks neater

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Sometimes not indenting looks neater

Print the name of day of the week

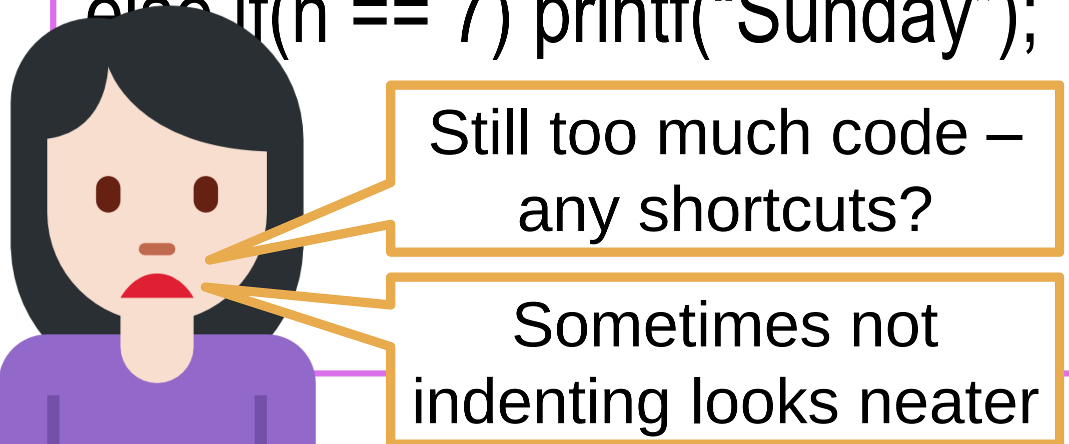
```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

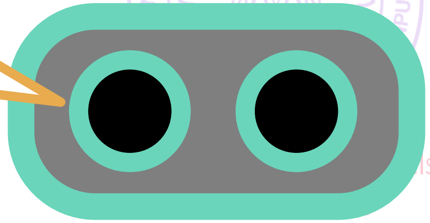

Still too much code – any shortcuts?
Sometimes not indenting looks neater

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

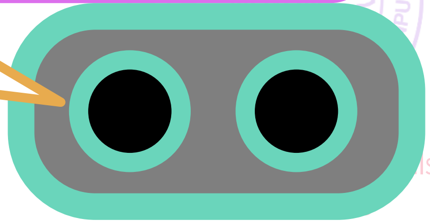
Print the name of day of the week

```
if(n == 1) printf("Monday");
else if(n == 2) printf("Tuesday");
else if(n == 3) printf("Wednesday");
else if(n == 4) printf("Thursday");
else if(n == 5) printf("Friday");
else if(n == 6) printf("Saturday");
else if(n == 7) printf("Sunday");
```

```
switch(n){
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

Print the name of day of the week

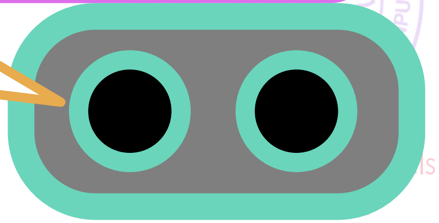
```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

```
switch(n){
    case 1: printf(“Monday”); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

The switch statement



Print the name of day of the week

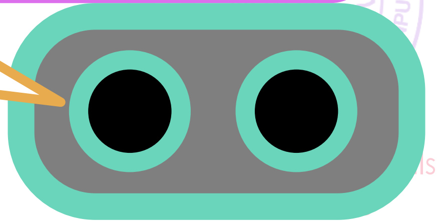
```
if(n == 1) printf("Monday");
else if(n == 2) printf("Tuesday");
else if(n == 3) printf("Wednesday");
else if(n == 4) printf("Thursday");
else if(n == 5) printf("Friday");
else if(n == 6) printf("Saturday");
else if(n == 7) printf("Sunday");
```

```
switch(n){
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

The switch statement



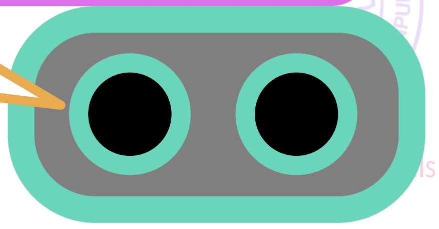
Print the name of day of the week

```
if(n == 1) printf("Monday");
else if(n == 2) printf("Tuesday");
else if(n == 3) printf("Wednesday");
else if(n == 4) printf("Thursday");
else if(n == 5) printf("Friday");
else if(n == 6) printf("Saturday");
else if(n == 7) printf("Sunday");
```

```
switch(n){
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    case 3: printf("Wednesday"); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

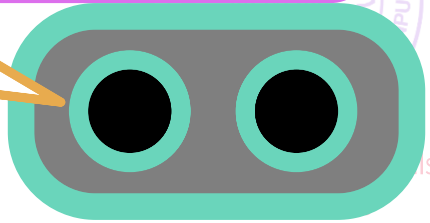
Print the name of day of the week

```
if(n == 1) printf("Monday");
else if(n == 2) printf("Tuesday");
else if(n == 3) printf("Wednesday");
else if(n == 4) printf("Thursday");
else if(n == 5) printf("Friday");
else if(n == 6) printf("Saturday");
else if(n == 7) printf("Sunday");
```

```
switch(n){
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    case 3: printf("Wednesday"); break;
    case 4: printf("Thursday"); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

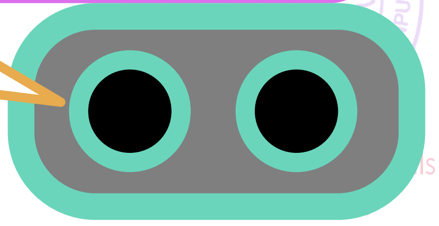
Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
```



The switch statement

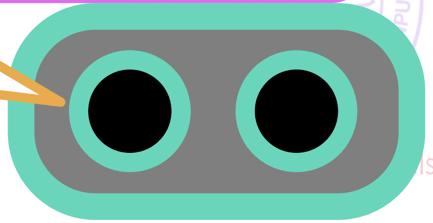
Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

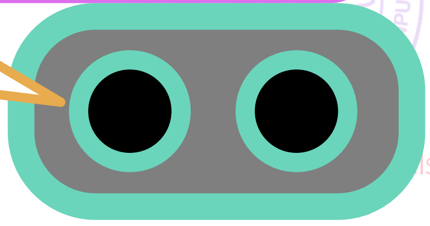
Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

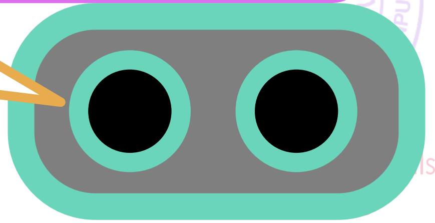
Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
}
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater



The switch statement

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

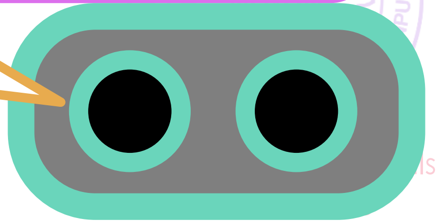
```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
}
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

This whole *block* is one valid statement

The switch statement



Print the name of day of the week

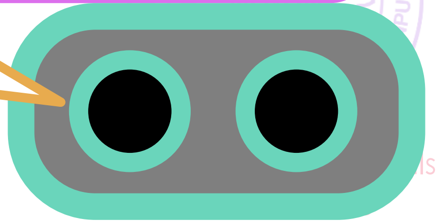
```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n == 7) printf(“Sunday”);
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
}
```

Still too much code – any shortcuts?

Sometimes not indenting looks neater

This whole *block* is one valid statement



The switch statement

Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n =
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
}
```

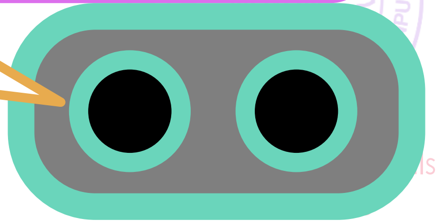
Just like if-else block is a single statement!

Still too much code – any shortcuts?

Sometimes not indenting looks neater

This whole *block* is one valid statement

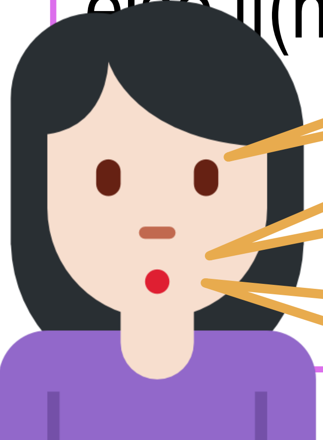
The switch statement



Print the name of day of the week

```
if(n == 1) printf(“Monday”);
else if(n == 2) printf(“Tuesday”);
else if(n == 3) printf(“Wednesday”);
else if(n == 4) printf(“Thursday”);
else if(n == 5) printf(“Friday”);
else if(n == 6) printf(“Saturday”);
else if(n =
```

```
switch(n){
    case 1: printf(“Monday”); break;
    case 2: printf(“Tuesday”); break;
    case 3: printf(“Wednesday”); break;
    case 4: printf(“Thursday”); break;
    case 5: printf(“Friday”); break;
    case 6: printf(“Saturday”); break;
    case 7: printf(“Sunday”); break;
}
```



Just like if-else block is a single statement!

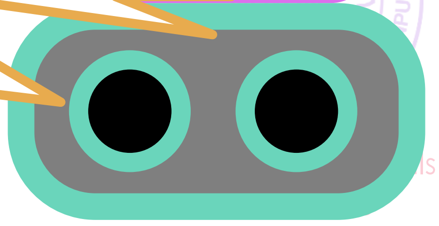
Still too much code – any shortcuts?

Sometimes not indenting looks neater

This whole *block* is one valid statement

Yes, can use switch inside if,else

The switch statement



The structure of a switch statement

The structure of a switch statement

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The structure of a switch statement

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

The structure of a switch statement

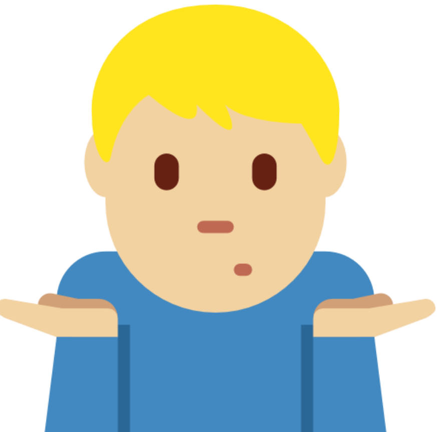
Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets



The structure of a switch statement

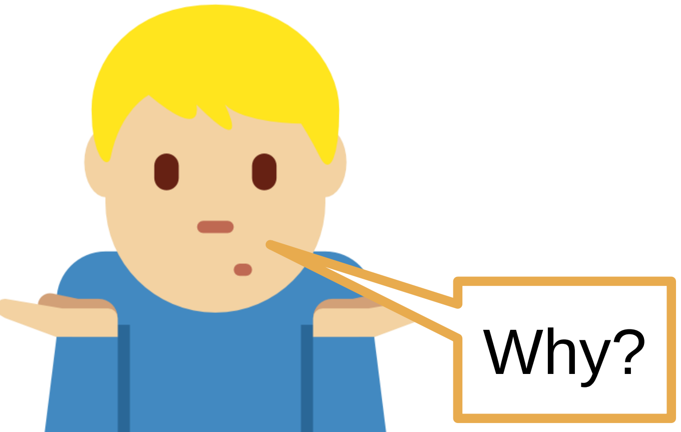
Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets



The structure of a switch statement

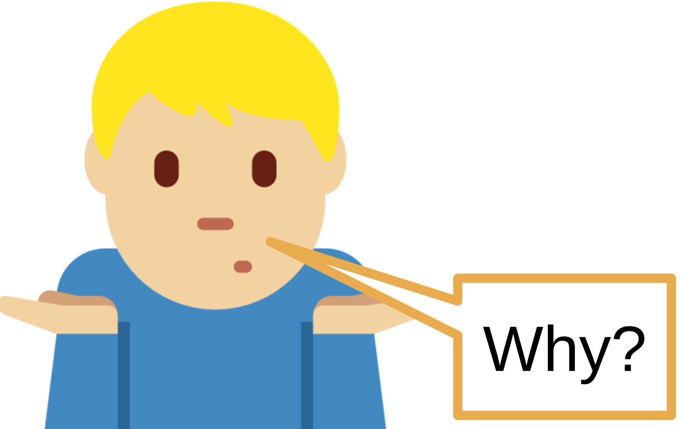
Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

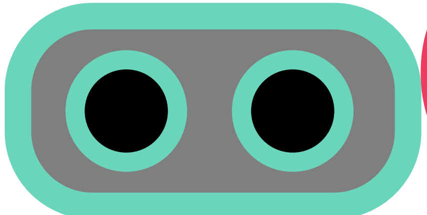
Double, float expressions banned

For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets





The structure of a switch statement

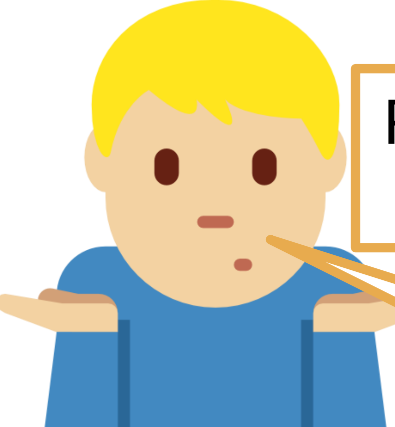
Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

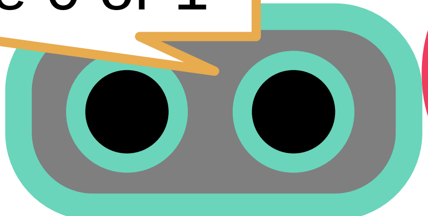
```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets



Why?

Relational expressions generate value 0 or 1



The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

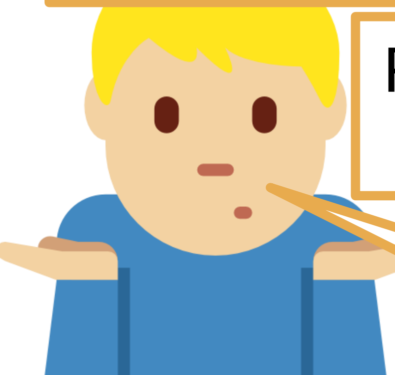
For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

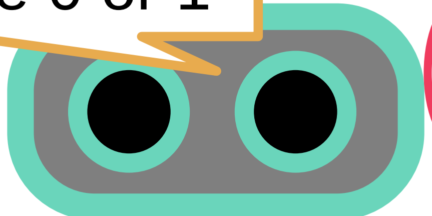
Careful about brackets

I'll give a warning but interpret 0, 1 as int

Relational expressions generate value 0 or 1



Why?



The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

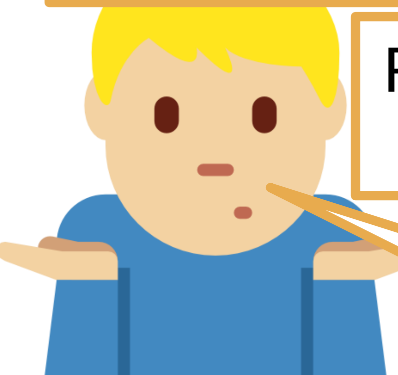
Labels must be integer constants e.g. 2, -50, 12

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

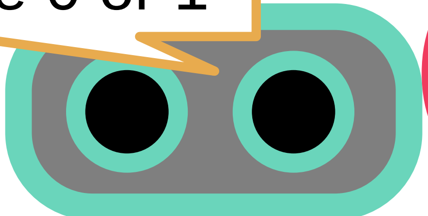
Careful about brackets

I’ll give a warning but interpret 0, 1 as int

Relational expressions generate value 0 or 1



Why?



The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

case a+2: wrong label

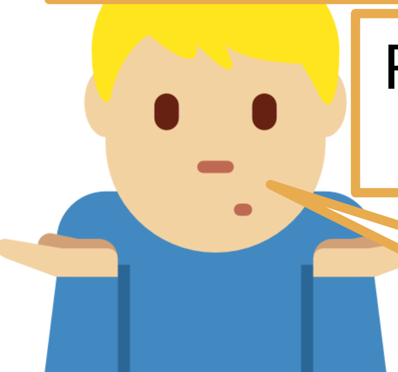
Labels must be integer constants e.g. 2, -50, 12

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

I'll give a warning but interpret 0, 1 as int

Relational expressions generate value 0 or 1



Why?

The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

case a+2: wrong label

Labels must be integer constants e.g. 2, -50, 12

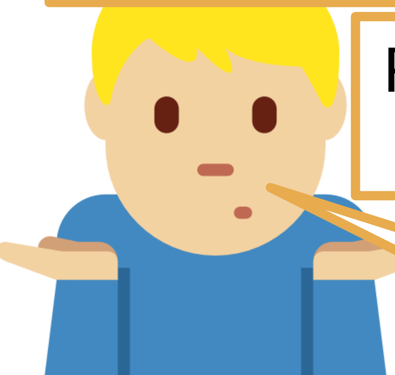
Labels must not repeat

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

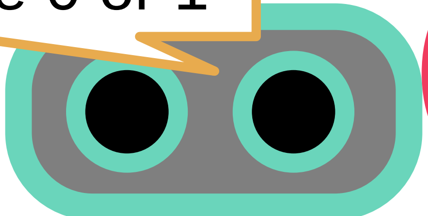
I'll give a warning but interpret 0, 1 as int

Relational expressions generate value 0 or 1

Careful about brackets



Why?



The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

case a+2: wrong label

Labels must be integer constants e.g. 2, -50, 12

Labels must not repeat

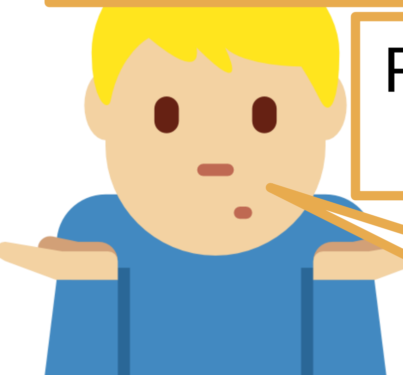
Can put any number of statements here, math formulae, printf, if-else, another switch (nested)

I’ll give a warning but interpret 0, 1 as int

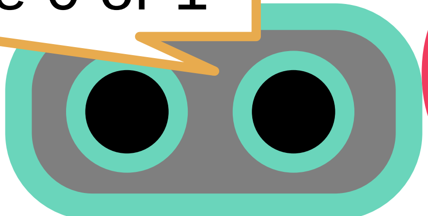
```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets

Relational expressions generate value 0 or 1



Why?



The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

case a+2: wrong label

Labels must be integer constants e.g. 2, -50, 12

Labels must not repeat

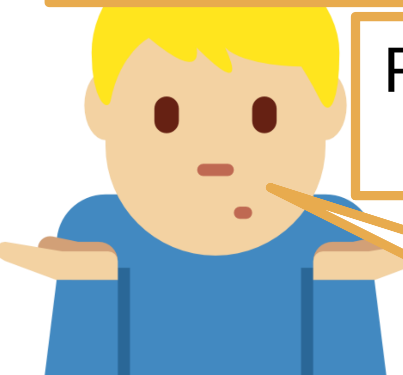
Can put any number of statements here, math formulae, printf, if-else, another switch (nested)

I'll give a warning but interpret 0, 1 as int

??

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

Careful about brackets



Relational expressions generate value 0 or 1

Why?

The structure of a switch statement

Must be an integer expression, e.g a, b+2, c*3 where a,b,c are int

Double, float expressions banned

For relational expressions, Mr C will warn but work

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

case a+2: wrong label

Labels must be integer constants e.g. 2, -50, 12

Labels must not repeat

Can put any number of statements here, math formulae, printf, if-else, another switch (nested)

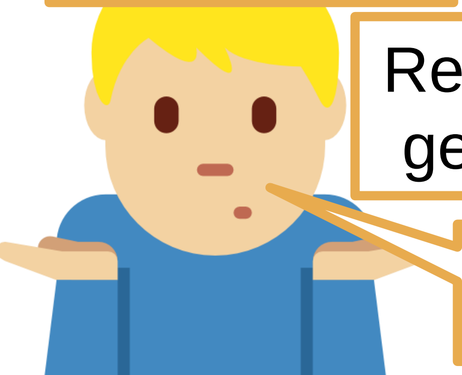
??

??

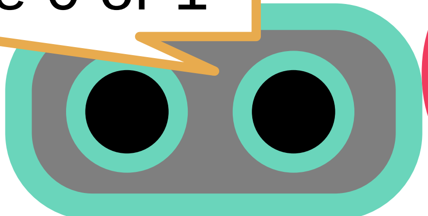
I'll give a warning but interpret 0, 1 as int

Relational expressions generate value 0 or 1

Careful about brackets



Why?



The working of a switch statement

The working of a switch statement

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The working of a switch statement

First value v of the integer expression calculated

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

If label matches, execute statements next to it till break is encountered

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

If label matches, execute statements next to it till break is encountered

In case no label matches execute statements next to default (if no default, do nothing)

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

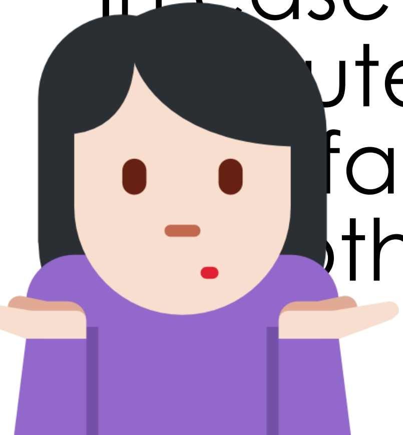
The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

If label matches, execute statements next to it till break is encountered

In case no label matches ute statements next fault (if no default, thing)



```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;

    ...
    case labelk: ... break;
    default: ... break;
}
```

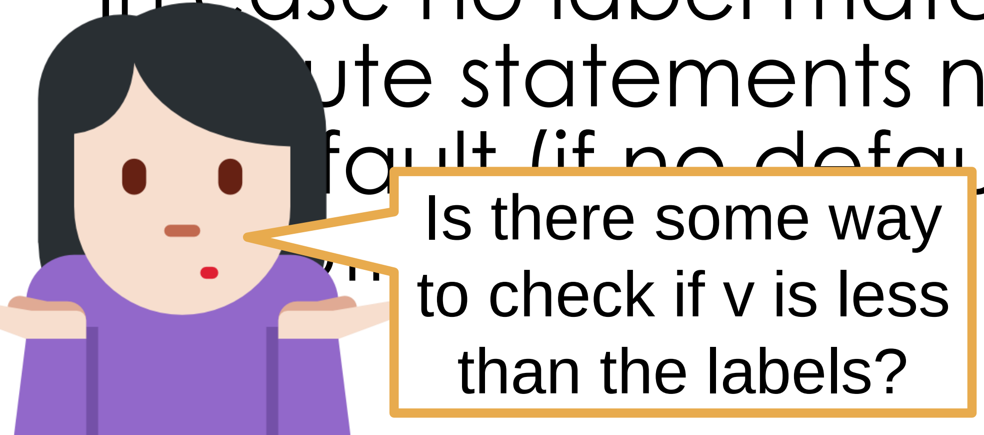
The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

If label matches, execute statements next to it till break is encountered

In case no label matches
ute statements next
fault (if no default,



```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```

The working of a switch statement

First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

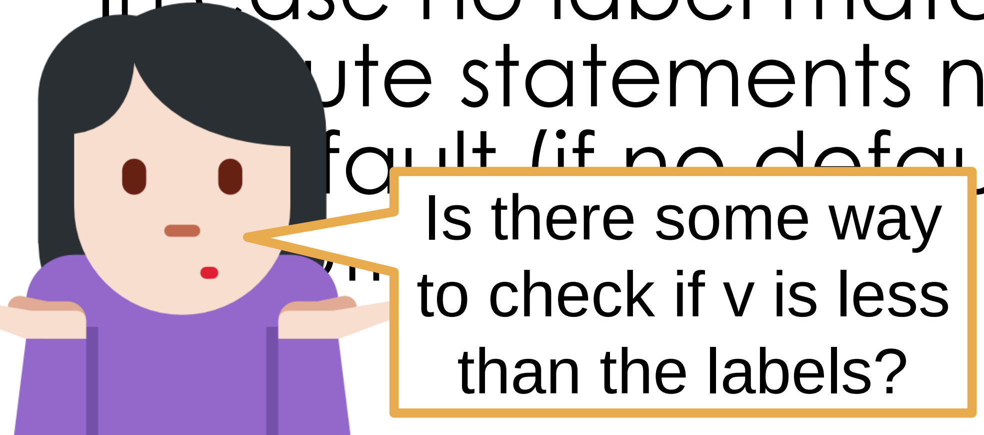
If label matches, execute statements next to it till break is encountered

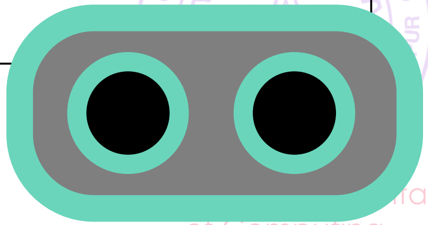
In case no label matches

ute statements next

fault (if no default,

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
    default: ... break;
}
```





The working of a switch statement

First value v of the integer expression calculated

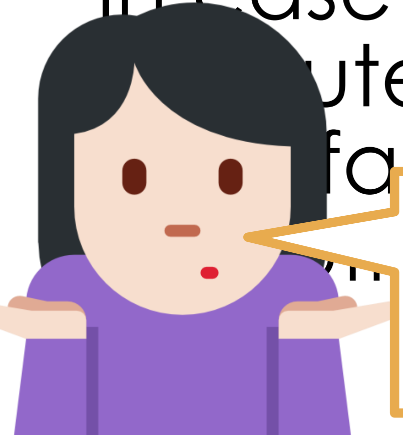
v is compared to all labels see if it is equal to any one

If label matches, execute statements next to it till break is encountered

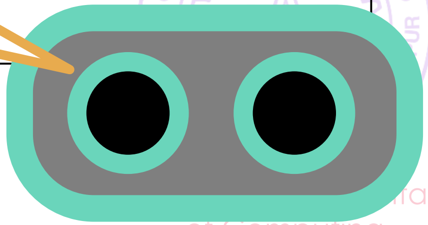
In case no label matches ute statements next fault (if no default,

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
}
```

Switch-case is a shortcut that only checks for equality and that too only with integers



Is there some way to check if v is less than the labels?



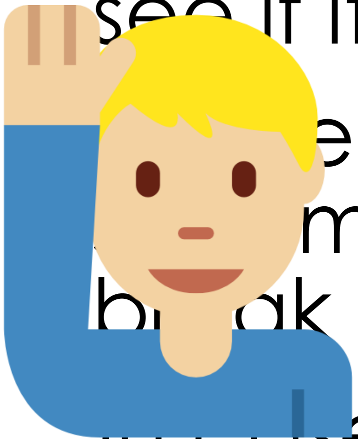
The working of a switch statement

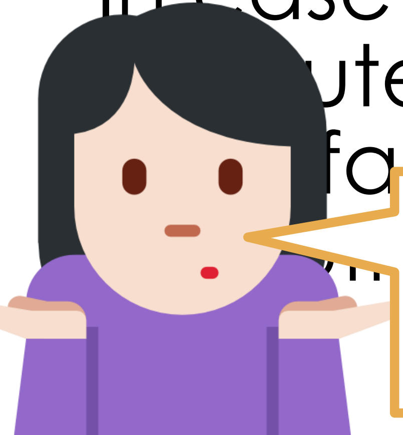
First value v of the integer expression calculated

v is compared to all labels see if it is equal to any one

el matches, execute ments next to it till b ak is encountered

se no label matches ute statements next fault (if no default,

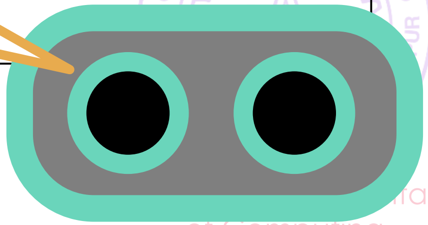




Is there some way to check if v is less than the labels?

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
}
```

Switch-case is a shortcut that only checks for equality and that too only with integers



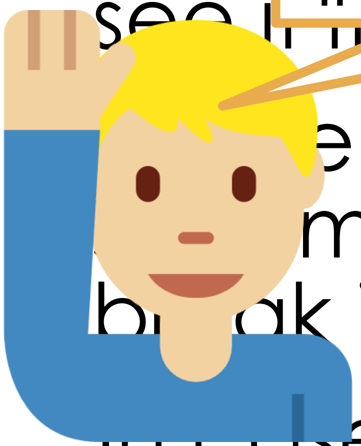
The working of a switch statement

First v... of the ind... expr...

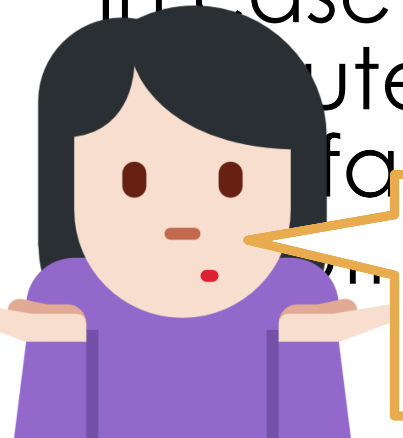
v is c... s see i... qual to any one

...el matches, execute ...ments next to it till b...ak is encountered

...se no label matches ...ute statements next ...fault (if no default, ...)



If we want to check for inequality or else work with float etc, we can always write if-else statements ourselves

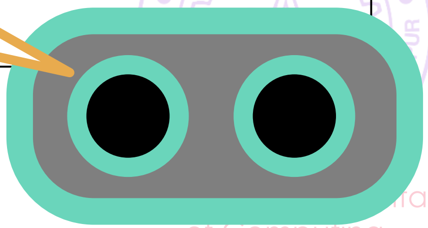


Is there some way to check if v is less than the labels?

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;

    ...
    case labelk: ... break;
}
```

Switch-case is a shortcut that only checks for equality and that too only with integers



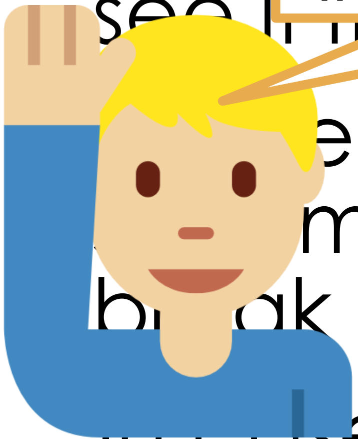
The working of a switch statement

First v
expr

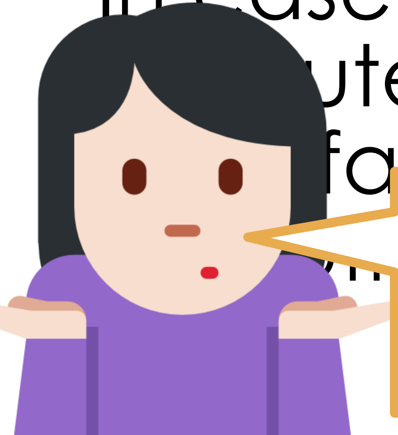
v is c s
see i equal to any one

el matches, execute
ments next to it till
b ak is encountered

se no label matches
ute statements next
fault (if no default,



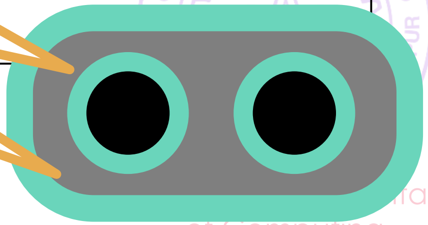
If we want to check for inequality or else work with float etc, we can always write if-else statements ourselves



Is there some way to check if v is less than the labels?

```
switch(integer expression){
    case label1: ... break;
    case label2: ... break;
    ...
    case labelk: ... break;
}
```

Switch-case is a shortcut that only checks for equality and that too only with integers



Exactly

The default case

The default case

The English word default can mean failure to fulfil a promise (*bank loan default*)

The default case

The English word default can mean failure to fulfil a promise (*bank loan default*)

… or it can mean a rule that applies when no other rule applies (*by default, Saturday is a holiday unless cruel instructor* *schedules a lecture*)

The default case

The English word default can mean failure to fulfil a promise (*bank loan default*)

… or it can mean a rule that applies when no other rule applies (*by default, Saturday is a holiday unless cruel instructor* *schedules a lecture*)

In switch case, whatever we write in default is executed if none of the labels match – used to handle incorrect input

The default case

The English word default can mean failure to fulfil a promise (*bank loan default*)

… or it can mean a rule that applies when no other rule applies (*by default, Saturday is a holiday unless cruel instructor* *schedules a lecture*)

In switch case, whatever we write in default is executed if none of the labels match – used to handle incorrect input

Can put the default case anywhere, not necessary at end

The default case

The English word default can mean failure to fulfil a promise (*bank loan default*)

… or it can mean a rule that applies when no other rule applies (*by default, Saturday is a holiday unless cruel instructor* *schedules a lecture*)

In switch case, whatever we write in default is executed if none of the labels match – used to handle incorrect input

Can put the default case anywhere, not necessary at end

Need not put default case at all. If we don’t put a default case, Mr C will do nothing if no labels match

The break statement

The break statement

The switch case statement behaves in a funny manner

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

That is why no brackets needed

```
case 2:{...}break;
```

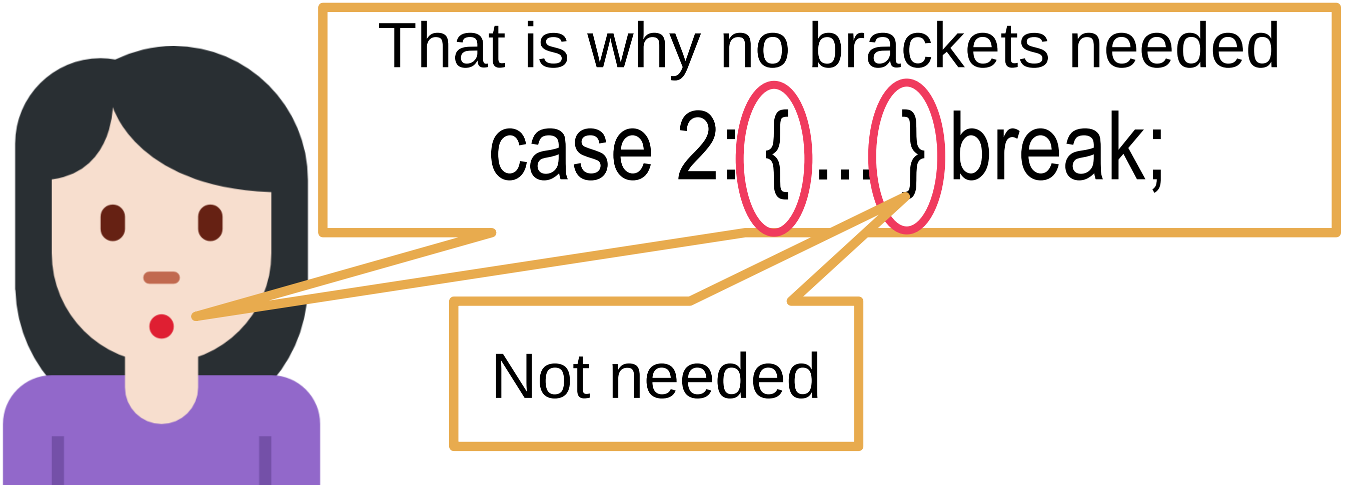
The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺



That is why no brackets needed

case 2:{...}break;

Not needed

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

That is why no brackets needed

case 2: { … } break;

Not needed

The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

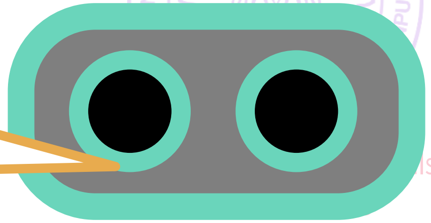
Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

That is why no brackets needed

case 2: { ... } break;

Not needed

Yes, the break; statement tells me when to stop



The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

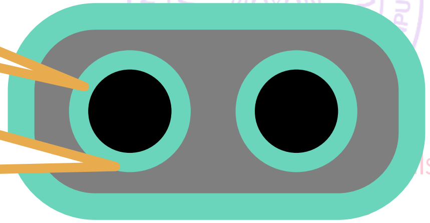
That is why no brackets needed

case 2:{...}break;

Not needed

Although this may seem strange now, this can be used very beautifully

Yes, the break; statement tells me when to stop



The break statement

The switch case statement behaves in a funny manner

Mr C finds the label that matches (else default if none match) but keeps executing all statements (**even those of other labels and default**) till encounters a *break;*

This behaviour is called *fall-through*

Once *break;* is encountered, Mr C claims he is done with the switch statement – *break;* stops Mr C's fall ☺

That is why no brackets needed

case 2:{...}break;

Not needed

Although this may seem strange now, this can be used very beautifully

Let us see an example

Yes, the break; statement tells me when to stop