EXPRESSIONS

- Expressions combine operands (variables, constants) and operators to produce new values.
  - e.g.: `x + count * (i + 4)`
- A constant expression is an expression that involves only constants.
  - A variable can be initialized using a constant expression.
  - e.g: `int total = 2+3*4;`

PRACTICE WITH RELATIONAL EXPRESSIONS

```
int a = 1, b = 2, c = 3;
```

Expression	True/False	Expression	True/False
`a < c`		`(a + b) >= c`	
`b <= c`		`(a + b) == c`	
`c <= a`		`a != b`	
`a > b`		`(a + b) != c`	
`b >= c`			

Is it equal to?
Not equal to?

ARITHMETIC EXPRESSIONS: TRUE OR FALSE

- Arithmetic expressions evaluate to numeric values.
- An arithmetic expression that has a value of zero is FALSE.
- An arithmetic expression that has a value other than zero is TRUE.

PRACTICE WITH ARITHMETIC EXPRESSIONS

```
int     a = 1,    b = 2,   c = 3;
float   x = 3.33, y = 6.66;
```

Expression	Numeric Value	True/False
a + b	3	T
b-2*a	0	F
c-b-a	0	F
c-a	2	T
y-x	3.33	T
y-2*x	0.0	F

EXPRESSIONS: WHAT IS THE VALUE OF *i* ????

- The operands used in an expression should be ideally of same type. The result of the expression will be of same type as the operand's type.

```
int i;
i = 3/2;  /* what will be value of i ? */
```

- Automatic type conversion is done some times when the operands are of different types.

AUTOMATIC TYPE CONVERSIONS

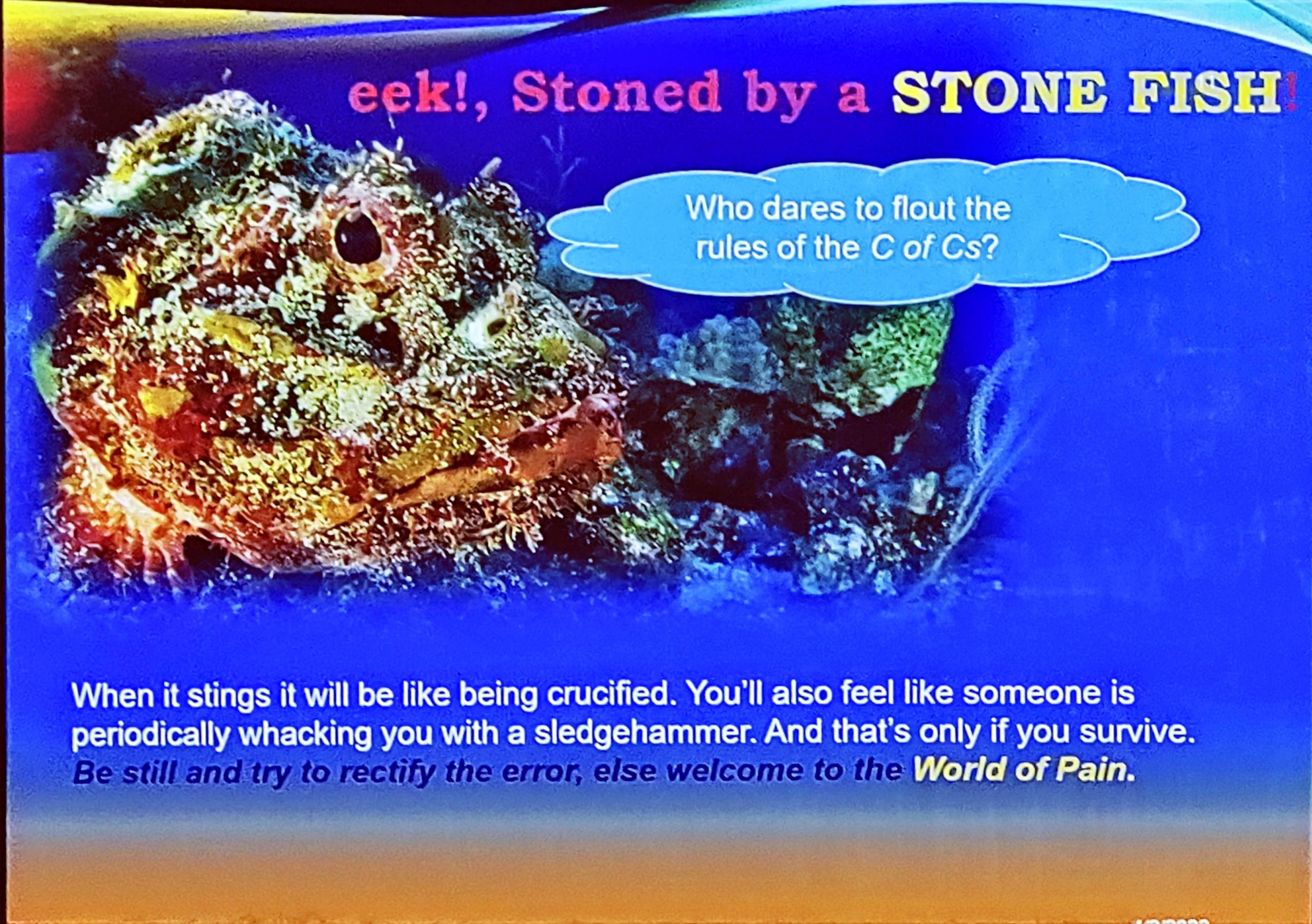
- A narrower type is converted to a wider type.
  - In 3 + 4.0, 3 is converted to float 3.0 (Compiler dependent)

- Expressions that might lose information, like assigning a longer integer type to a shorter, may draw a warning, but they are not illegal.

```
float a, b, z;
int  i, sum[5];
a=3.01
z = 5.1;

i = a+1;                    // i ← 4 ← 4.01 ← 3.01+1
sum[z] = 3;
b = sum[z] + a;             // b ← 6.01 ← 3+3.01
```

eek!, Stoned by a STONE FISH!



When it stings it will be like being crucified. You'll also feel like someone is periodically whacking you with a sledgehammer. And that's only if you survive. ***Be still and try to rectify the error, else welcome to the World of Pain.***

The Stone fish codones this first error and provides a clue to recrify the error!

In spite of repeated warnings, a careless student (Roll No. 200123024.2, from IITG, name withheld) stepped on me the other day while sCuba diving in the Sea of Cs and entered the *Heaven of Pain*!

```
float a, b,z;
int  i, sum[5];
a=3.01, z = 5.1;
i = a+1;                    // i ← 4 ← 4.01 ← 3.01+1
sum[z] = 3;
b = sum[z] + a;             // b ← 6.01 ← 3+3.01
```

sum[5.1]

EXPRESSIONS

- What will be the values of i, j and k?

```
float i, j;
int k;
i = 3/2;
j = 3.0/2;
k = 3.0/2.0;
```

1.0 Converted to float

1.5 2 converted to float

1 Answer converted to integer

(1 or 1.5 or 2 ...?)

EXPLICIT TYPE CONVERSIONS (TYPE CASTING)

- You can force the type to be converted.
  - `f = (float) 3;  /* has value 3.0 */`

- Syntax : **(type-name) expression**
  - **`float f;`**
  - **`f = (float) 3/2;`**
    - **`/*  3 → 3.0  because of explicit type conversion`**
    - **`3.0/2  → 3.0/2.0  because of automatic conversion`**
    - **`So, f gets value 1.5  */`**

EXPLICIT TYPE CONVERSIONS (TYPE CASTING)

- You can force the type to be converted.
  - f = (float) 3; /* has value 3.0 */

- Syntax : (type-name) expression
  - float f;
  - f = (float) 3/2;
    - /* 3 → 3.0 because of explicit type conversion
    - 3.0/2 → 3.0/2.0 because of automatic conversion
    - So, f gets value 1.5 */

    - double d = 3.8;
      int i;
      i = (int) d; /* value of d itself is not changed */
      /* the value of (int) d is 3 */

DIVING SOLO IN THE REEFS OF *SEA OF Cs*

```
#include <stdio.h>int
int main()
{
        float a, b;
         int  sum[6];
         a=3.01;
        sum[5] = 3;
        b = sum[5] + a;
        sum[5] = b;
        printf("%d\n", sum[5]);
        printf("%f\n", b);
        printf("%f\n", sum[b]);
        return 0;
}
```

6
6.01

ASSIGNMENT OPERATORS AND EXPRESSIONS

- Most binary operators of the form

  variable = variable op expression

  can thus be written as

  variable op= expression

- So what would this mean?

```
x *= y+1;
```

ASSIGNMENT OPERATORS

= += -= *= /= %=

Statement	Equivalent Statement
a = a+2;	a += 2;
a = a-3;	a -= 3;
a = a*2;	a *= 2;
a = a/4;	a /= 4;
a = a%2;	a %= 2;
b = b+(c+2);	b += c + 2;
d = d*(e-5);	d *= e - 5;

CONDITIONAL EXPRESSIONS

- variable = cond_expr ? expr1 : expr2
  cond_expr is evaluated first
  If TRUE (non zero) then variable = expr1
  else variable = expr2

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

Try this...

```
x = 5 ? 0:1;  /* what is value of x */
```

0 or 1 or 5?

INCREMENT AND DECREMENT OPERATORS

- ++ (increment unary operator)
- -- (decrement unary operator)
- x++ means x = x + 1
- ++x also means x = x + 1

- NB: Increment and decrement operators *can only be applied to variables, NOT to Constants or Expressions*. Thus,

  5++ is not allowed because it means

  5 = 5+1

  y = x++ ; is same as y = x; x = x+1;
  - Post increment : Use and then increment

  y = ++x ; means x = x+1; y = x;
  - Pre increment : Increment and then use

USE WITH CARE

INCREMENT AND DECREMENT OPERATORS

- Check-1

```
int j, k, m;
j = 5;
k = j++;
m =  ++(j + k);
```

- Check-2

```
int a[5];
int j = 0;
a[++j] = 4;
```

Box jellyfish, named for their body shape, have tentacles covered in biological booby traps known as nematocysts - tiny darts loaded with lethal poison.