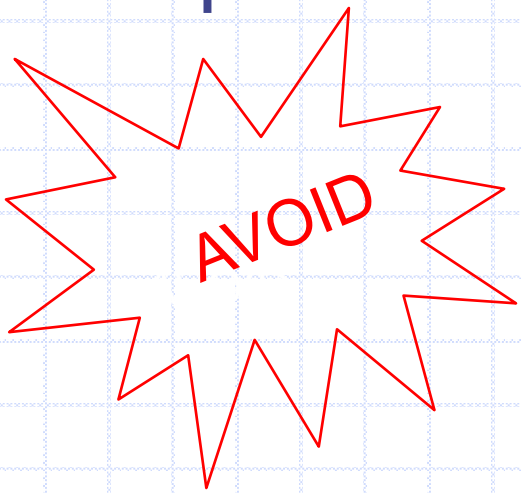
Scope of a variable in C

```
#include <stdio.h>
int main(){

 for (int i=1;i<=2;i++)
     printf("%d\n",i);

 return 0;
}
```

- Output?



Block scope of a variable

```
#include <stdio.h>
int main(){

 { //start block
 int i;
 for (i=1;i<=2;i++)
     printf("%d\n",i);
 } //end block

 return 0;
}
```

- Output?

  1

  2

Block scope of a variable

```
#include <stdio.h>
int main(){

 {
 int i;
 for (i=1;i<=2;i++)
     printf("%d\n",i);
 }
 printf("outside %d\n",i);

 return 0;
}
```

- Output?

Compiler error: 'i' undeclared

Block scope of a variable

```
#include <stdio.h>
int main(){
 int i;
 for (i=1;i<=2;i++){
     printf("%d\n",i);
     int j=0;
     printf(“j=%d\n",j+1);
 }

 return 0;
}
```

- Output?

1

j=1

2

j=1

Back to Break

- Used for exiting a loop forcefully
- Example Program:

Read 100 integer inputs from a user. Print the sum of inputs until a negative input is found (Excluding the negative number) or all 100 inputs are exhausted.

```
int value;
int sum = 0;
int i;
for (i = 0; i < 100; i++) {
     scanf("%d", &value);
     if (value < 0) {
          //-ve number: no need to go
          // around the loop any more!!
          break;
     }
     sum = sum + value;
}
printf("%d", sum);
```

To break or not to!

- Use of break sometimes can simplify exit condition from loop.
- However, it can make the code a bit harder to read and understand.
- Tip: if the loop terminates in **at least two ways** which are sufficiently different and requires substantially different processing then consider the use of termination via **break** for one of them.

Continue

- Used for skipping an iteration of a loop
- The loop is NOT exited.
- Example Program:

Read 100 integer inputs from a user.
Print the sum of only positive inputs.

```
int sum = 0;
int i, value;
for (i = 0; i < 100; i++) {
     scanf("%d", &value);
     if (value < 0) {
          //-ve number: no need to add it
          // to the sum. Go ahead and
          // check the next input.
          continue;
     }
     sum = sum + value;
}
printf("%d", sum);
```

Break and Continue

- if there are nested loop: break and continue apply to the nearest enclosing loop only.

```
for (i = 0; i < 100; i++) {
  for (j = 0; j < 100; j++) {
     if (...) break;
  }
  ...
}
```

Continue and Update Expr

- Make sure continue does not bypass update-expression for loops
  - Specially for while and do-while loops

```
i = 0;
while (i < 100) {
   scanf("%d", &value);
   if (value < 0) continue;
   sum = sum + value;
   i++;
}
```

i is never incremented
potentially infinite loop!!

Continue and Update Expr

- Correct Code:

```
i = 0;
while (i < 100) {
   i++;
   scanf("%d", &value);
   if (value < 0) continue;
   sum = sum + value;
}
```

Continue and Update Expr

- Correct Code:

```
i = 0;
while (i < 100) {
  scanf("%d", &value);
  if (value < 0) {
     i++;
     continue;
  }
  sum = sum + value;
  i++;
}
```

Class Quiz: How many times the loop is executed?

```
int a = 10 - 6;
while (a < 10) {
    if (a = 5)  {
            printf("%d\n", a);
    }
    a=a+1;
}
```

Output
5
5
5
...

A common bug

Probable intention:

```
int a =10 - 6;
while (a < 10) {
    if (a == 5)  {
        printf("%d", a);
    }
    a=a+1;
}
```

Output
5

Assignment Operator =

- The value of assignment expression is same as the value of its RHS

LHS = RHS

- It also has the side effect of updating the "box" of LHS

x = 5 + 23	Value is 28 x updated to 28
y = 12; x = y = 5 + 23	(x = (y = 5 + 23)) right associative
(x=5) + (y=3)	Result of + is 8 x becomes 5, y becomes 3 (eventually!)

Ternary operator ?:

- Select among values of two expressions based on a condition

```
condition ? true_expr : false_expr
```

- Both expressions must be of compatib type.
  - The expression is called ternary expression.

Condition

value if condition is True

value if condition is False

```
int abs;
int val;
scanf ("%d", val);
if (val < 0)
    abs = -val;
else
    abs = val;
printf("%d", abs);
```

```
int abs;
int val;
scanf ("%d", val);
abs = (val < 0) ? –val : val;
printf("%d", abs);
```

```
int val;
scanf ("%d", val);
printf("%d", (val < 0) ? –val : val);
```

ESC101: Introduction to Computing

f(unction)