# C Review

## Question

How many bytes does it take to store the following string:

```
char string[ ] = {“0123456789”};
```

(Assuming ascii characters (char))

# C Review

## Answer

11 bytes to store the string:

```
char string[ ] = {“0123456789”};
```

Strings are NULL terminated, thus the length is N+1 where N is the number of characters.

# C Review

## Primary Data Types

| Variable Type | Keyword | Bytes Required | Range |
|---|---|---|---|
| Character | char | 1 | -128 to 127 |
| Unsigned character | unsigned char | 1 | 0 to 255 |
| Integer | int | 2 | -32768 to 32767 |
| Short Integer | short int | 2 | -32768 to 32767 |
| Long Integer | long int | 4 | -2,147,483,648 to 2,147,438,647 |
| Unsigned Integer | unsigned int | 2 | 0 to 65535 |
| Unsigned Short integer | unsigned short int | 2 | 0 to 65535 |
| Unsigned Long Integer | unsigned long int | 4 | 0 to 4,294,967,295 |
| Float | float | 4 | 1.2E-38 to |
| Double | double | 8 | 2.2E-308 to |
| Long Double | long double | 10 | 3.4E-4932 to 1.1E+4932 |

Note: This is assuming a 16-bit machine since the *int* size is 2 bytes. C is not always portable across machines. To make sure your code is portable, define your own data types as uint16, int32, etc. where these can be explicitly set to the correct data types in a header with #define statements.

# C Review

## Question

If i=5, what is y?

```
y = 2 * ++i;
```

# C Review

Answer

If i=5, what is y?

```
y = 2 * ++i;
```

y = 12  (i is incremented before the statement is evaluated)

# C Review

## Question

If i=5, what is y?

```
y = 2 * i--;
```

# C Review

## Answer

If i=5, what is y?

```
y = 2 * i--;
```

y = 10  (The statement is evaluated, then i is decremented)

# C Review

## The Conditional Expression

```
if (exp1)
    exp2;
else
    exp3;
```

```
exp1 ? exp2 : exp3;
```

Examples:

max = (a>b) ? a : b;

absy = (y<0) ? –y : y;

# C Review

## Program Flow

```
while (exp){
    statements;
}
```

Statement repeats until exp is false (zero).

# C Review

## Program Flow

```
for (initialize; test; update){
    statements;
}
```

comma operator – allows you to include more than one initialization or update expression in a single for loop.

```
for (i=0, x=10;   i<20;   i+=2, x*=3){
    statements;
}
```

# C Review

## Program Flow

```
do{
    statements;
}while(exp);
```

Exit Condition Loop. The loop must be executed at least once, the statement portion is repeated until the expression becomes false.

Note the semicolon at the end of the *while* expression.

# C Review

## Program Flow Equivalence

Make a *for* loop like a *while* loop

```
for( ; test; ){
    statements;
}
```

```
while(test){
    statements;
}
```

# C Review

## Program Flow Equivalence

Make a *while* loop like a *for* loop

```
init;
while(test){
    statements;
    update;
}
```

```
for(init; test; update){
    statements;
}
```

# C Review

## Program Flow

```
if(exp){
    statement1;
}else{
    statement2;
}
```

In this example you don’t need the braces, {,},
but what happens when you go back and add more statements and don’t add the braces?
It’s better to start with the braces.

# C Review

## Program Flow

Using Braces you know exactly what is happing

```
if(score < 1000)
    bonus = 0;
else if (score < 1500)
    bonus = 1;
else if (score < 2000)
    bonus = 2;
else if (score < 2500)
    bonus = 4;
else
    bonus = 6;
```

Better Code

It's more readable

```
if(score < 1000){
    bonus = 0;
}else{
    if (score < 1500){
        bonus = 1;
    }else{
        if (score < 2000){
            bonus = 2;
        }else{
            if (score < 2500){
                bonus = 4;
            }else{
                bonus = 6;
            }
        }
    }
}
```

# C Review

## Question

If number = 10, what does the following code print?

```
if (number > 6)
      if (number < 12 )
            printf(“You are close \n”);
else
      printf(“Sorry, you lose a turn! \n”);
```

# C Review

Answer

If number = 10, what does the following code print?

```
if (number > 6)
      if (number < 12 )
            printf(“You are close \n”);
else
      printf(“Sorry, you lose a turn! \n”);
```

You are close.

# C Review

## Question

If number = 5, what does the following code print?

```
if (number > 6)
      if (number < 12 )
            printf(“You are close \n”);
else
      printf(“Sorry, you lose a turn! \n”);
```

# C Review

Answer

If number = 5, what does the following code print?

```
if (number > 6)
      if (number < 12 )
            printf("You are close \n");
else
      printf("Sorry, you lose a turn! \n");
```

*Nothing is printed*

The rule is an *else* goes with the most recent *if* unless braces indicate otherwise.

A good reason to be liberal with the braces!!!
Make your intention explicit!

# C Review

## Question

If number = 15, what does the following code print?
(Now with the code indented the way it should be)

```
if (number > 6)
      if (number < 12 )
            printf(“You are close \n”);
      else
            printf(“Sorry, you lose a turn! \n”);
```

# C Review

Answer

If number = 15, what does the following code print?
(Now with the code indented the way it should be)

```
if (number > 6)
      if (number < 12 )
            printf(“You are close \n”);
      else
            printf(“Sorry, you lose a turn! \n”);
```

Sorry, you lose a turn!

## C Review

Braces will force you to make your intention explicit.

```
if (number > 6){
      if (number < 12 ){
            printf(“You are close \n”);
      }else{
            printf(“Sorry, you lose a turn! \n”);
      }
}
```

# C Review

The *break* statement

break; // exit the loop which is being executed

Can be used in the following loops & statements:
for
while
do-while
switch

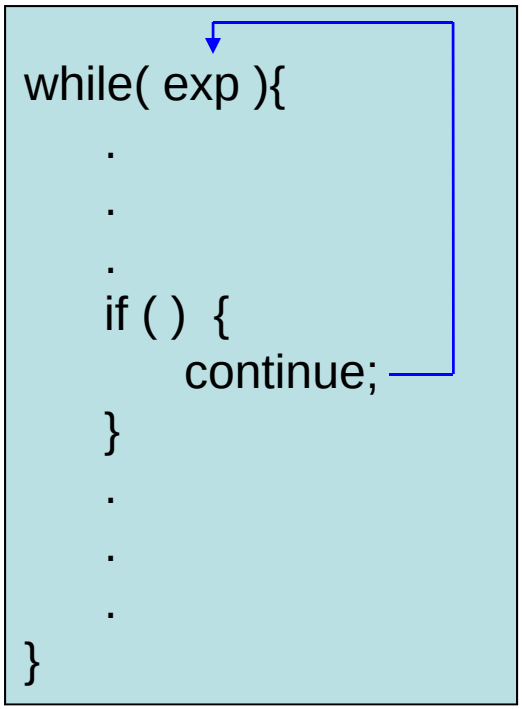
```
while( (ch=getchar()) != EOF){
      f1(ch);
      if (ch == '\n' )  break;
      f2(ch);
      f3(ch);
}
nextf();
```

In nested loops, break exits from the innermost loop only.

# C Review

The *continue* statement

continue; // skip the rest of the loop statements

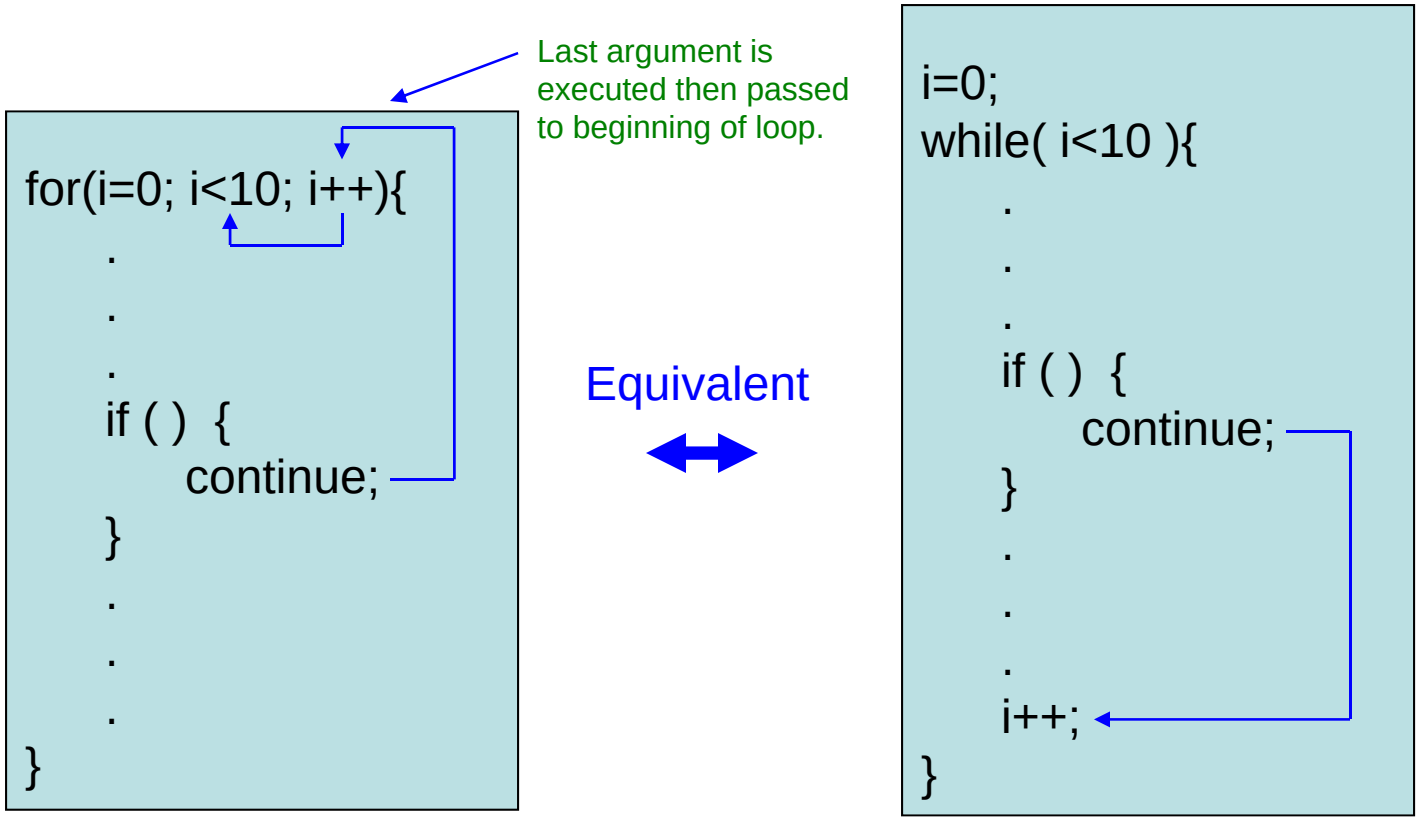



Useful for speeding up loops.

# C Review

## The *continue* statement in a for loop

continue; // skip the rest of the loop statements




Note: *The continue in the while loop will not do this*. It is only shown for illustration purposes, to show what the while loop would have to do to be equivalent to the for loop.

# C Review

The *goto* statement

Are you serious???!!!

Don’t use this.  It will make your code unreadable.

# C Review

## The *switch* statement

```
switch( c ) {
      case 'A':
            capa++;
      case 'a':
            lettera++;
      default :
            total++;
}
```

If c is equal to 'A‘, all three statements of the **switch** body in this example are executed since a **break** statement does not appear before the following case.

Execution control is transferred to the first statement (capa++;) and continues in order through the rest of the body.

If c is equal to 'a', lettera and total are incremented.

Only total is incremented if c is not equal to 'A' or 'a'.

# C Review

## The *switch* statement

```
switch( i ) {
      case -1: n++; break;
      case 0 : z++; break;
      case 1 : p++; break;
}
```

In this example, a **break** statement follows each statement of the **switch** body.

The **break** statement forces an exit from the statement body after one statement is executed.

If i is equal to –1, only n is incremented. The **break** following the statement n++; causes execution control to pass out of the statement body, bypassing the remaining statements.

Similarly, if i is equal to 0, only z is incremented;

if i is equal to 1, only p is incremented.

The final **break** statement is not strictly necessary, since control passes out of the body at the end of the compound statement, but it is included for consistency.

# C Review

## The *switch* statement

```
switch( c ) {
      case 'a' :
      case 'b' :
      case 'c' :
      case 'd' :
      case 'e' :
      case 'f'  :
            hexcvt(c);
      default :
}
```

In this example, if *constant-expression* equals any letter between 'a' and 'f', the hexcvt() function is called (as well as the default case).

**Note:** switch can't be use with floats, nor can it be used effectively if a variable must fall into a range e.g.
if( integer < 1000 && integer > 2)

# C Review

## A *switch* problem – what does *break* break?

```
network code(  ) {
      switch( line ) {
            case THING1 :
                  doit1( );
                  break;
            case THING2 :
                  if( x == STUFF ){
                        do_first_stuff();
                        if ( y== OTHER_STUFF)
                              break;
                        do_later_stuff();
                  } // coder mean to break to here…..
                  initialize_modes_pointer();
                  break;
            default:
                  processing();
      } // …. but actually broke to here!
      use_modes_pointer();
}
```

The programmer wanted to break out of the *if* statement, forgetting that *break* actually gets you out of the nearest enclosing iteration or *switch* statement. Here it broke out of the *switch* and executed the call to use_modes_pointer() without the proper pointer initialization.

This caused the first major network problem in AT&T's 114 year history. On the afternoon of Jan 15, 1990 AT&T's network became in large part unusable for about nine hours. The code was running on a model 4ESS Central Office Switching System.