I have a program that starts as follows:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct id {
  int i;
  double d;
} ID;

int main(int argc, char **argv)
{
  ID x, y;
  struct id z;
  ID *p;
  char s[20], *t;
```

Which of the following lines will not compile (answer all that apply)?

A. `"Fred" = s;`
B. `y->d = 3.14;`
C. `x = y;`
D. `p->i = x.i;`
E. `s = t;`
F. `s = argv[1];`
G. `s[30] = 'J';`
H. `p = &z;`
I. `x.i = y.d;`
J. `s[19] = x.i;`

Answers to Clicker Questions for CStuff-1 Lecture

COSC 360 - James S. Plank

```
A. "Fred" = s;     - This doesn't compile, because you can't set a string to anything.

B. y->d = 3.14;    - This doesn't compile because y is a struct and not a pointer.
                     You need to do "y.d".

C. x = y;          - This compiles fine.  It will copy y to x.  I don't like it, but it's legal.

D. p->i = x.i;     - This compiles fine.  Now, I'm not saying it's going to run correctly.
                     In particular, if we haven't initialized p, we'll be lucky to get a
                     segmentation violation with this.  However, it compiles fine.

E. s = t;          - This doesn't compile.  You can't set an array to anything.

F. s = argv[1];    - This doesn't compile either for the same reason.

G. s[30] = 'J';    - This will compile, but some compilers will give you a warning,
                     since you are accessing s out of bounds.  Some compilers won't care.

H. p = &z;         - Perfectly legal.

I. x.i = y.d;      - This is legal, too.  It will convert y.d to an integer.

J. s[19] = x.i;    - This is legal.  It will convert x.i to a one-byte integer.
```

On question J, a student in 2024 aske the following question:

> *I have a question about the last question for today's clicker. You said that assigning x.i to s[19] will typecast it to a "1 byte integer". I am not really sure what that means because to my understanding anything that is 1 byte is simply a char. How does the compiler know its an integer if it's one byte?*

I'm sorry for the confusion. A char is indeed a one-byte integer. It is either signed, in which case it ranges from -128 to 127, or it is unsigned, in which case it ranges from 0 to 255. Sometimes students get confused because chars represent characters, but those characters are just integers. Their character conversion is a standard called ASCII, which, for example, maps 'a' to 97 and 'A' to 65. They are not stored as characters, but they are stored as one-byte integers, so the string "Aa" is really stored in three one-byte integers: { 65, 97, 0 }.

In the clicker question, the variable s is declared as:

```
char s[20];
```

Which means that 20 bytes are allocated for s, and when you access s[0], that will access the first of the 20 bytes, and when you access s[19], that will access the last of the 20 bytes. When I say:

```
s[19] = x.i;
```

I am saying store x.i in the last of these 20 bytes. Since x.i is an integer, it is first converted to a char (one-byte integer) before being stored. So, if x.i = 0x123, then 0x23 will be stored in s[19].

I hope that helps answer your question -- JP