iClicker Attendance

Please click on A if you are here:

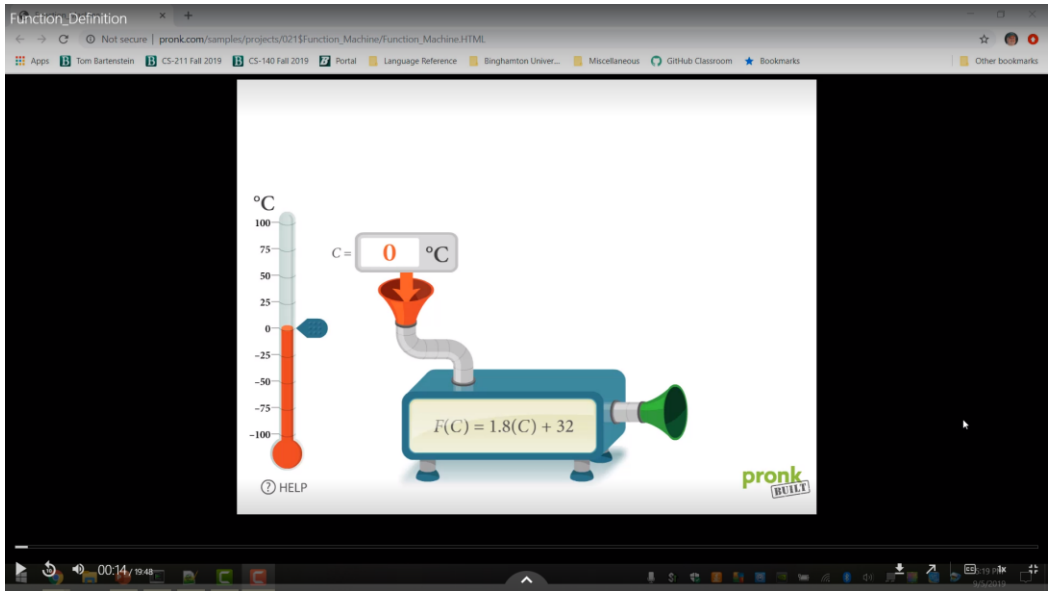
A. I am here today.

Data Conversion

software diversity in action

Function Definition Video Review

- First Line:
  - Return type
  - name
  - argument list
- Body
  - return statement
- Declaration vs. Definition
  - Right side up code



Program Design : Functions

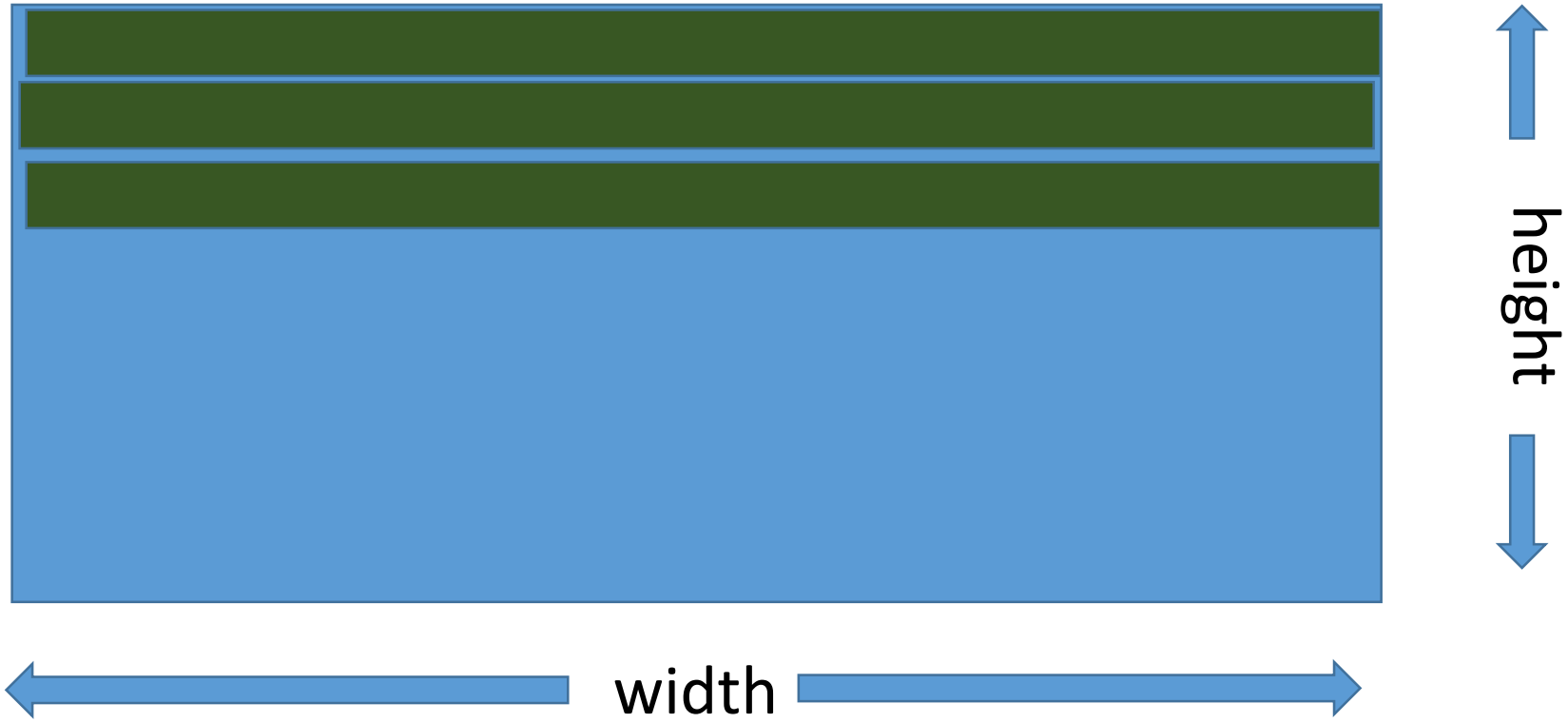
- Break up a big problem into smaller problems
- Encapsulate each smaller problem in a function
- Keep functional boundaries clear and well defined
  - Think in terms of parameters, processing, return value
  - Remember, each function can only return one value
- Example: Project installment 2 Verilog infrastructure

- Fancy name: Functional Decomposition

Demo

- We are working on CAD for house plans. Our customer has asked us to write a function, given the length and width of a wall in inches, figure out how many linear feet of board to the nearest foot will we need to cover the wall if the boards are 8 inches wide?

Big Picture



Need ceil(*height*/8) boards, each *width* wide.

Need ceil(*boards* * *width*/12) linear feet

Function Arguments?

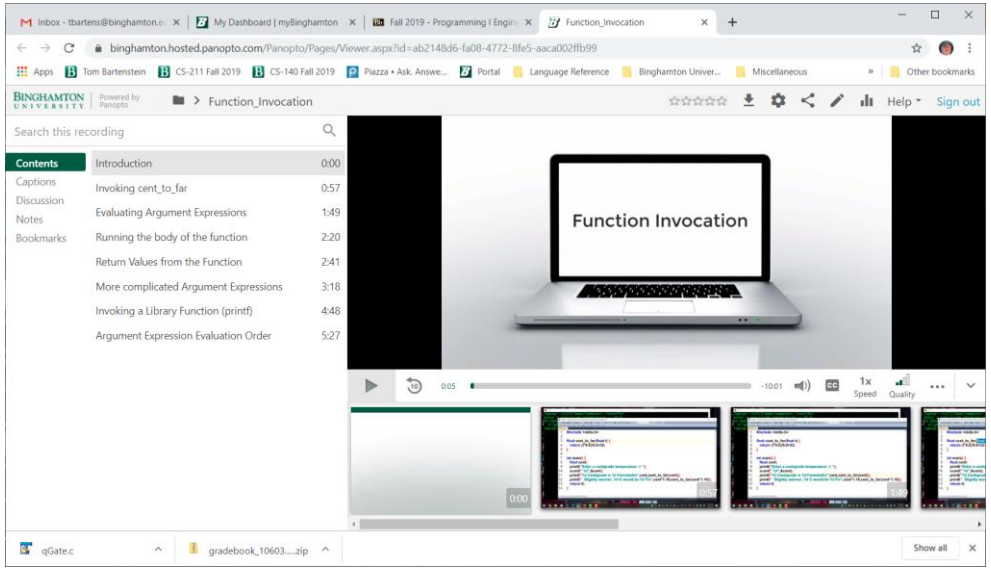
- What values is the user willing to give us?
- What do we need to figure out the answer?
- What is the data type of each argument?
- What name will you use for each argument?
  - Easy to type and remember

Note: When you implement your function, you may change your mind. That's OK.

Return Type?

- What type of data does your user expect you to return?
- Does the data type reflect the precision of the inputs?

Function Invocation Video Review



- Argument Expression Evaluation
- Copy arguments to parameters
- Transfer control to called function
- "Replace" invocation with returned result
- Function "side-effects" Other than just calculating return value
  - Read or write to/from the terminal or a file
  - Update global variables
  - manage memory
- "void" functions are all about side-effects

Data Conversion

iClicker Question

- What gets printed?

```
int x=2;  float y=3.1;
y=y*x;
printf("y=%f\n",y);
```

A. No value – compiler error.
B. y=3.1
C. y=6
D. y=6.2

iClicker Question

- What gets printed?

```
int x=2;  float y=3.1;
x=y*x;
printf("x=%f\n",x);
```

A. No value – compiler error.
B. x=6
C. x=6.0
D. x=6.2

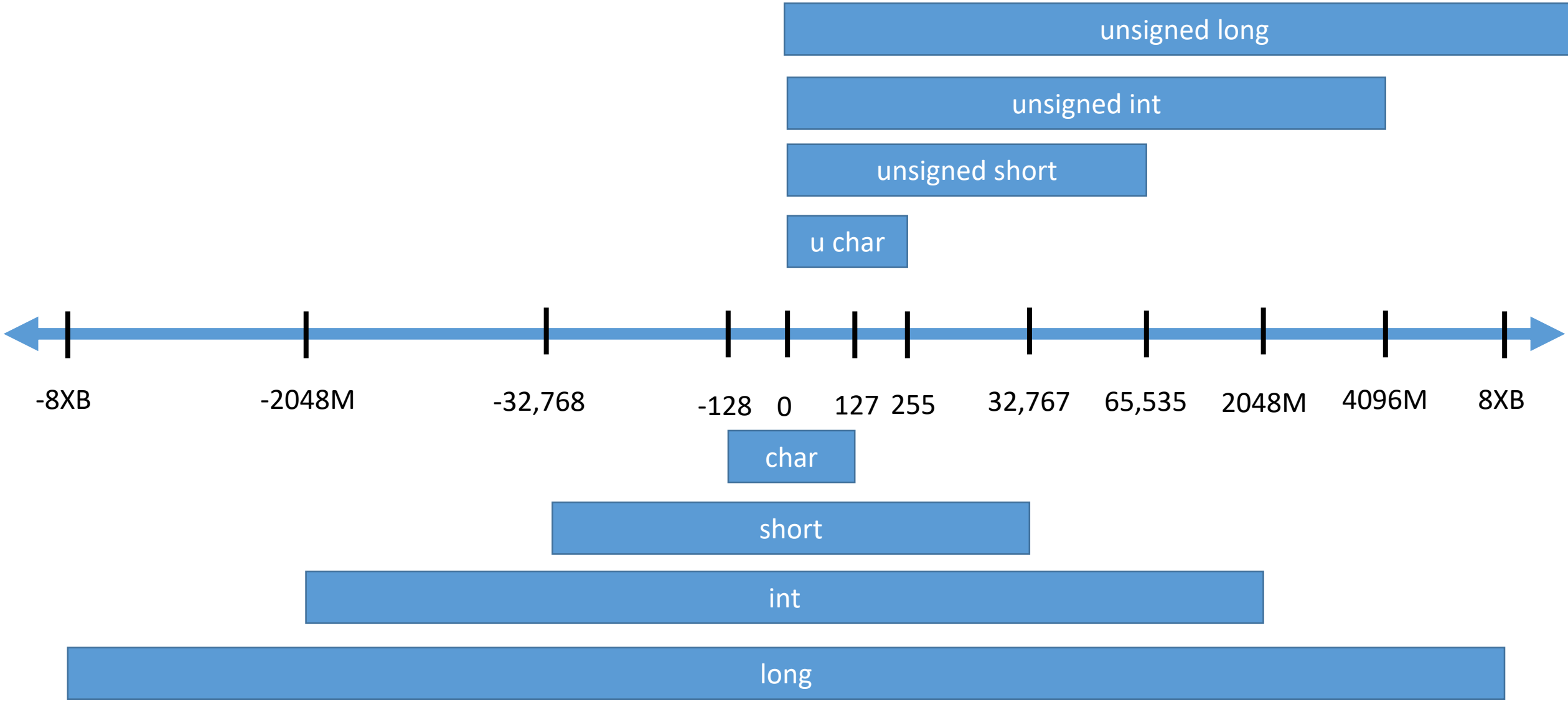
What happens when types are mixed?

- Mixed Type Expressions
  ```
  int x; float y; x=y*x;
  ```
- Assignment Statements
  ```
  int x; float y; x=y*3.0;
  ```
- Argument Evaluation
  ```
  int myfn(float x); int y=myfn(3);
  ```
- Explicit Casting
  ```
  int x=7; float y = ((float)x)/3;
  ```

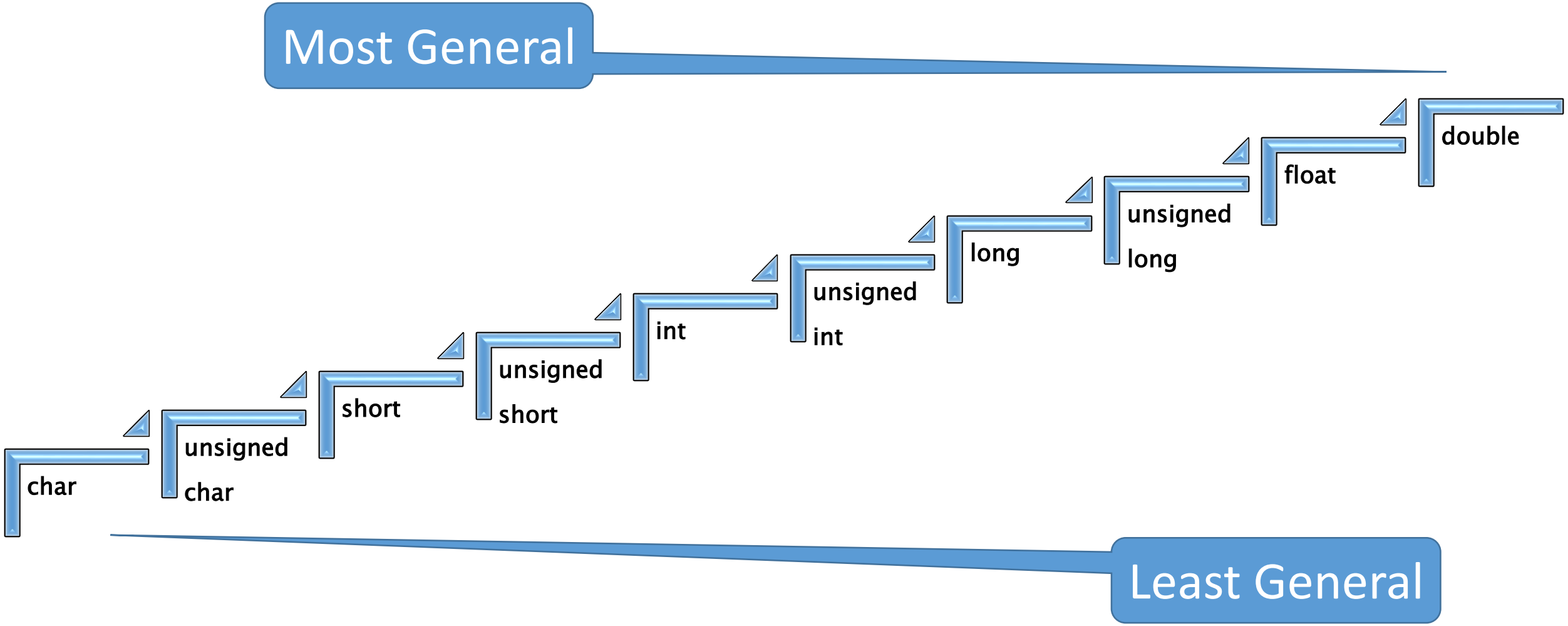
C Automatic type conversion rules

- In an expression, C converts all components in that expression to the most “general” type, and then evaluates the expression using that general type
- In an assignment (or argument evaluation), C converts the value of the expression to the type of the receiver
- C converts expressions with a valid explicit cast

Generality of Integer Types



Generality of Numeric Types



Converting Signed vs Unsigned

- Bits stay exactly the same, but the bits are INTERPRETTED differently

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

```
char x = -22;
unsigned char y=x;
printf("y=%d\n",y);
// prints y=234
```

```
unsigned char w = 234;
char z=w;
printf("z=%d\n",z);
// prints z=-22
```

Converting Signed vs Unsigned

- Bits stay exactly the same, but the bits are INTERPRETTED differently

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

```
char x = -22;
unsigned char y=x;
printf("y=%d\n",y);
// prints y=234
```

```
unsigned char w = 234;
char z=w;
printf("z=%d\n",z;
// prints z=-22
```

Demo – Sign Conversion

```
int main(int argc,char **argv) {

        int sint=atoi(argv[1]);
        unsigned int uint=sint;

        printf("sint=%10d = 0B%s = 0x%x\n",sint,bitString(sint),sint);
        printf("uint=%10u = 0B%s = 0x%x\n",uint,uBitString(uint),uint);
        return 0;
}
```

Changing Integer Size

- Truncate or Pad on left with sign bit

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0

```
char x = -22;
short y=x;
```

```
short w = -22;
char z=w;
```

Changing Integer Size (unsigned)

- Truncate or Pad on left with sign bit (=0)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	0	1	0	1	0

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0

```
unsigned char x = 234;
unsigned short y=x;
```

```
unsigned short w = 234;
unsigned char z=w;
```

Demo – widthConv.c

```
int main(int argc, char **argv) {
        assert(argc>1);

        int n32 = atoi(argv[1]);
        long n64=n32;
        short n16=n32;
        char n08=n32;

        unsigned int u32=n32;
        unsigned long u64=u32;
        unsigned short u16=u32;
        unsigned char u08=u32;

        printf("n64=%10ld = 0x%016lx\n",n64,n64);
        printf("u64=%10lu = 0x%016lx\n",u64,u64);
        printf("n32=%10d = 0x%08x\n",n32,n32);
        printf("u32=%10u = 0x%08x\n",u32,u32);
        printf("n16=%10d = 0x%04hx\n",n16,n16);
        printf("u16=%10u = 0x%04hx\n",u16,u16);
        printf("n08=%10d = 0x%02hhx\n",n08,n08);
        printf("u08=%10u = 0x%02hhx\n",u08,u08);
        return 0;
}
```

Integer to Float

- Add .0 and convert to nearest floating point representation

```
int x = 1331254215;
float y=x;
printf("y=%f\n",y);
// prints y=1331254272.000000
```

Float to Integer

- Truncate at the decimal point (round towards zero)

```
float w=-374289.74112;
int z=w;
printf("z=%d\n",z);
// prints z=-374289
```

Demo – floatConv.c

```
int main(int argc, char **argv) {
        assert(argc>1);

        int i32=atoi(argv[1]);
        float f32=atof(argv[1]);
        int fi32=f32;
        float if32=i32;

        printf("i32=%d f32=%f fi32=%d if32=%f\n",i32,f32,fi32,if32);
        return 0;
}
```

Conversion Errors

- When C converts a negative signed number to a positive number

```
char x = -1; unsigned char y = x; printf("y = %d\n",y);
```

- When C converts a wide number to a smaller width, but the number doesn't fit

```
short x=260; char y = x; printf("y = %d\n",y);
```

- When C truncates decimals

```
float x=2.7; int y=x;; printf("y = %d\n",y);
```

Integer Division Pitfall

```
int atBats = atoi(argv[1]);
int hits = atoi(argv[2]);

float battingAverage = (hits/ atBats) * 1000.0;

printf("Everybody has a zero batting average?\n");
```

iClicker Question

- What gets printed?

```
unsigned int width=+8;
signed int leftX = -13;
if ((leftX < 0) && (leftx + width) > 0)  {
        printf("Rectangle crosses y axis\n");
}
```

A. No value – compiler error.
B. Rectangle crosses y axis
C. Nothing

Explicit Casting

- Programmer tells C explicitly to perform conversion
- "cast" prefix operator: (*type*)*expression*
  - Causes expression to be evaluated and then converted to the specified type
- Needed when the programmer knows better than the compiler!
- Note that casting is pretty high in operator precedence
  - after parens, but before any mathematical operations

```
int battingAverage = ((float) hits / atBats) * 1000;
```

Resources

- Programming in C, Chapter 3, 13 (pp 325-328)
- Wikipedia Type Conversion: https://en.wikipedia.org/wiki/Type_conversion
- C Tutorial – Cast operator: http://www.crasseux.com/books/ctutorial/The-cast-operator.html#The%20cast%20operator