From specifications to algorithms to programs

- Specs – Precise statement of what the problem is about: What is a valid input? What is the target output?
- Algorithm – Sequence of steps to solve the problem; can be represented in textual, graphical (flowchart), or pseudocode form
- Program – the algorithm written in a programming language, ready for compilation and execution

Data types

- Some basic data types are integers, floating point numbers and characters

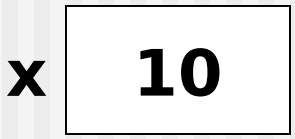
  -123 3.1416 ‘e’

- We can combine these basic types to form more complex types, e.g., a list of integers, or a string of characters

  {11, 13, 17} “Happy birthday!”

Operations on data: variables

- Data can be stored (and later retrieved) in variables
- Variables are named locations in the computer's memory




```
main()
{
   int  x = 10;
   printf("%d", x);
}
```

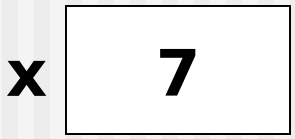
Stores 10 in the variable named x, then prints the contents of x

%d is the format code for a decimal integer

Operations on data: arithmetic

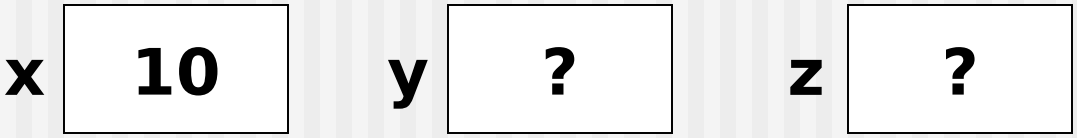
- Data can be entered via an input device
- Basic arithmetic (add +, subtract -, mult *, divide /, remainder %) can be performed

```
main()
{
   int  x;
   printf("enter any integer: ");
   scanf("%d", &x);
   printf("%d %d %d %d %d",
        x+2, x-2, x*2, x/2, x%2);
}
```



If the user inputs 7, the output is

9 5 14 3 1



Operations on data: assignment

- Longer arithmetic expressions can be used in the right side of assignment statements

```
main()
{
   int  x = 10, y, z;
   y = (2*x)+1;
   z = 2*(x+1);
   x = x+1;
   printf("%d %d %d", x, y, z);
}
```

Initially x=10, y=?, z=?
Then x=10, y=21, z=?
Then x=10, y=21, z=22
Then x=11, y=21, z=22

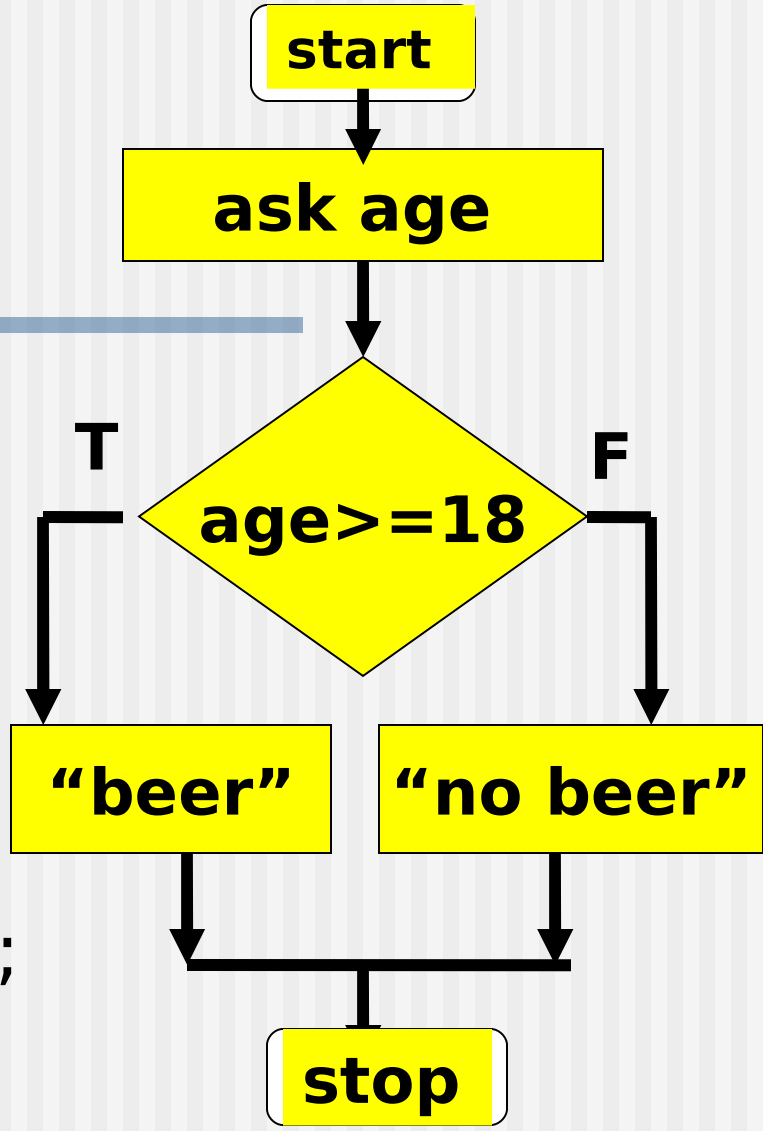
Outputs 11 21 22

Operations on data: comparisons

- Straight-line programs are kind of boring, lets try **if-else** statements to form branches
- We can **compare numbers** to form our conditions (equals? ==, less? <, greater? >, less than or equal? <=, greater than or equal? >=, NOT equal? !=)

Operations on data

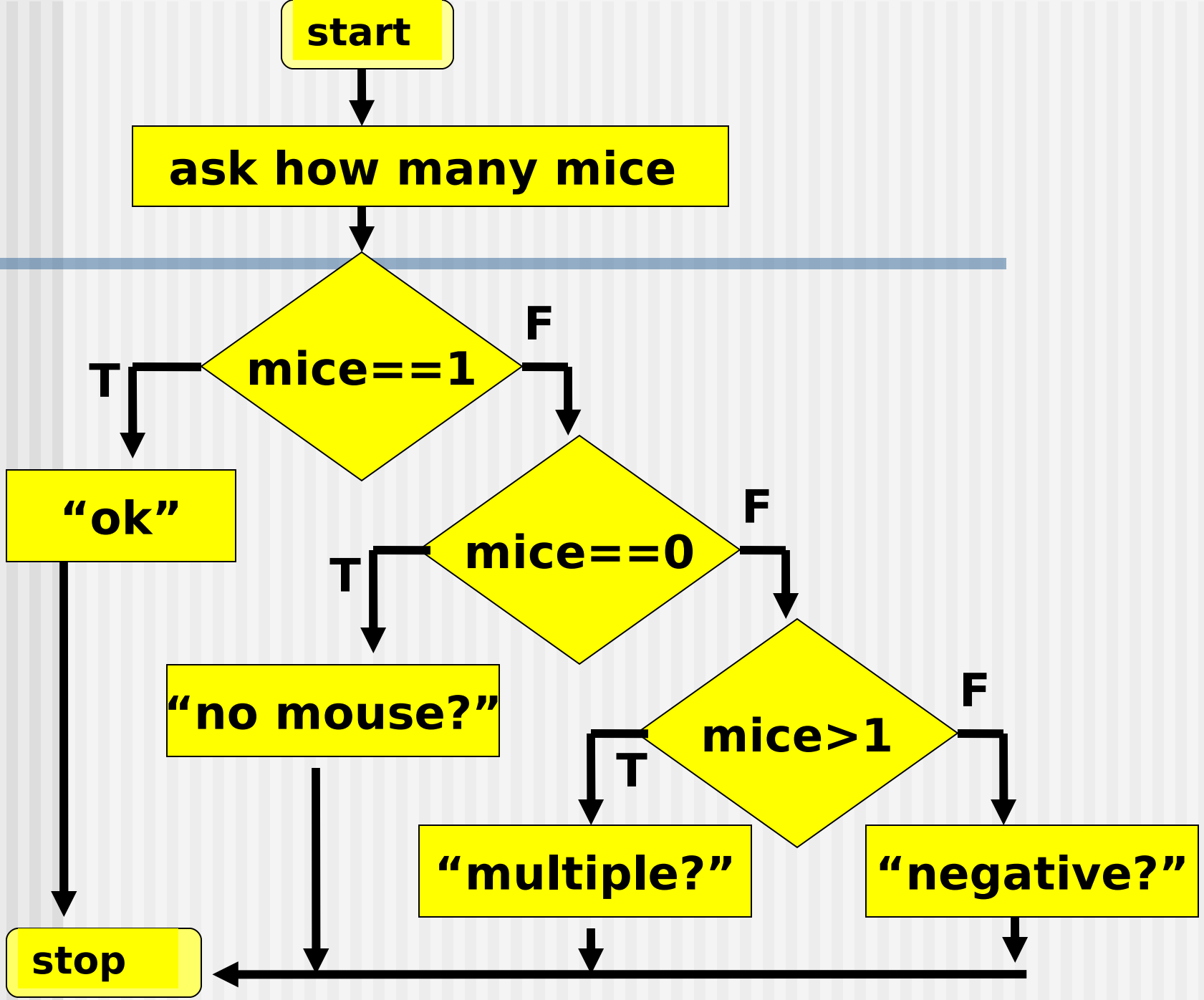
```
…
int  age;
printf(“enter your age: ”);
scanf(“%d”, &age);
if ( age>=18 ) {
   printf(“want some beer?”);
}
else {
   printf(“sorry… can’t offer you beer”);
}
```



Multiple branching: comparing ints

```
int     mice;  /* no. of mice connected to the PC */
printf("how many mice do you have?");
scanf("%d", &mice);
if (mice == 1) printf("ok... just right\n");
else if (mice == 0) printf("no mouse?\n");
else if (mice > 1) printf("multiple mice??\n");
else printf("eeeks...negative mice?!?!\n");
```

Note: we can omit the pairs of curly {braces} here, but be sure to include them when a branch consists of more than one statement


start
ask how many mice
mice==1
T
F
“ok”
mice==0
T
F
“no mouse?”
mice>1
T
F
“multiple?”
“negative?”
stop

Exercise

- Study the code fragment below. Draw the flowchart. Is it doing essentially the same thing as our earlier code? Why or why not?

```
int     mice;  /* no. of mice connected to the PC */
printf("how many mice do you have?");
scanf("%d", &mice);
if (mice == 1) printf("ok... just right\n");
if (mice == 0) printf("no mouse?\n");
if (mice > 1) printf("multiple mice??\n");
else printf("eeeks...negative mice?!?!\n");
```

Programming tips

- Use **meaningful** variable names to help document your programs:
  - **x, y, z** are valid names but they do not mean much
  - **mice** and **age** in our examples are better names
  - In C, variable names must start with a letter and may be followed by more letters or digits (the under_score may also be used)
  - C is **case-sensitive** so be careful when you type: **age**, **Age**, **AGE**, and **aGe** can all represent different variables/memory locations

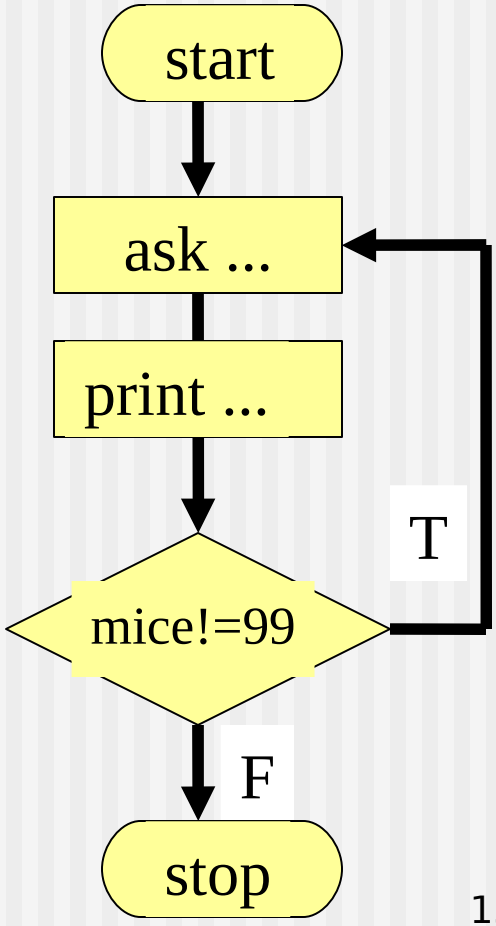
Programming tips

- Improve program layout
  - Use indentation to indicate which parts of the code go together (e.g., statements in an if-branch block should all be indented together)
  - Add extra spaces, extra lines to avoid crowding
  - Use English comments to help explain unclear code **/* comment */** or **// comment**

Loops

- A simple application of loops is to allow multiple inputs for code testing

```
...
int mice;
do {
    // ask how many mice
    ... scanf("%d", &mice);
    // print appropriate message
    ... printf("......");
} while ( mice != 99 );
// repeat for most inputs
```



Loops

```
int mice;
do {
    // ask how many mice
    printf("how many mice do you have?");
    scanf("%d", &mice);
    // print appropriate message
    if (mice == 1) printf("ok... just right\n");
    else if (mice == 0) printf("no mouse?\n");
    else if (mice > 1) printf("multiple mice??\n");
    else printf("eeeks...negative mice?!?!\n");
} while ( mice != 999 );   // repeat for most inputs
printf("bye!\n");
```

Exercise

- Design an algorithm (flowchart and pseudocode) then write a complete C program for the ff task:

  Keep asking the user for a secret four-digit password. If the user guesses the correct password within six tries, congratulate her. Otherwise print a warning message on unauthorized access and stop.

  Input: a sequence of up to six numbers

  Output: a congratulatory message (if the secret password is guessed within six tries) or a warning message (if user is unable to guess within six tries)

Sample solution ☺

```
/* Sample solution to the password problem */
#include <stdio.h>

main()
{
        int password = 1024;  /* secret password hidden from the user, can be encrypted if necessary */
        int guess;
        int tries =0, maxtries = 6;

        do {
                tries = tries+1;
                printf("what is the 4-digit password? ");
                scanf("%d", &guess);
        } while ((guess != password) && (tries < maxtries));

        if (guess==password)
                printf("congrats. You guessed the password in %d tries\n", tries);
        else
                printf("security alert... unauthorized access prohibited\n");
}
```