ECS 36A, April 2, 2025

Example Program: °F to °C Table

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
          int fahr;                    /* fahrenheit temperature */
          int celsius;                 /* celsius temperature */
          register int lower = 0;      /* begin table here */
          register int upper = 300;    /* end table here */
          register int step = 20;      /* increment */

          printf("deg F\tdeg C\n");
          fahr = lower;
          while(fahr <= upper){
                    celsius = 5 * (fahr - 32) / 9;
                    printf("%d\t%d\n", fahr, celsius);
                    fahr += step;
          }
          exit(EXIT_SUCCESS);
}
```

Example Program: °F to °C Table


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

int main(void)
{
          int fahr;                    /* fahrenheit temperature */
          int celsius;                 /* celsius temperature */
          register int lower = 0;      /* begin table here */
          register int upper = 300;    /* end table here */
          register int step = 20;      /* increment */

          printf("deg F\tdeg C\n");
          fahr = lower;
          while(fahr <= upper){
                    celsius = 5 * (fahr - 32) / 9;
                    printf("%d\t%d\n", fahr, celsius);
                    fahr += step;
          }
          exit(EXIT_SUCCESS);
}
```

Example Program: °F to °C Table

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

int main(void)
{
          int fahr;                    /* fahrenheit temperature */
          int celsius;                 /* celsius temperature */
          register int lower = 0;      /* begin table here */
          register int upper = 300;    /* end table here */
          register int step = 20;      /* increment */

          printf("deg F\tdeg C\n");
          fahr = lower;
          while(fahr <= upper){
                    celsius = 5 * (fahr - 32) / 9;
                    printf("%d\t%d\n", fahr, celsius);
                    fahr += step;
          }
          exit(EXIT_SUCCESS);
}
```

Program declaration

Program end (could be return here too)

Example Program: °F to °C Table

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

int main(void)
{
        int fahr;                       /* fahrenheit temperature */
        int celsius;                    /* celsius temperature */
        register int lower = 0;         /* begin table here */
        register int upper = 300;       /* end table here */
        register int step = 20;         /* increment */

        printf("deg F\tdeg C\n");
        fahr = lower;
        while(fahr <= upper){
                celsius = 5 * (fahr - 32) / 9;
                printf("%d\t%d\n", fahr, celsius);
                fahr += step;
        }
        exit(EXIT_SUCCESS);
}
```

Program declaration

declarations

Program end (could be **return** here too)

Example Program: °F to °C Table

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

int main(void)
{
        int fahr;                       /* fahrenheit temperature */
        int celsius;                    /* celsius temperature */
        register int lower = 0;         /* begin table here */
        register int upper = 300;       /* end table here */
        register int step = 20;         /* increment */

        printf("deg F\tdeg C\n");
        fahr = lower;
        while(fahr <= upper){
                celsius = 5 * (fahr - 32) / 9;
                printf("%d\t%d\n", fahr, celsius);
                fahr += step;
        }
        exit(EXIT_SUCCESS);
}
```

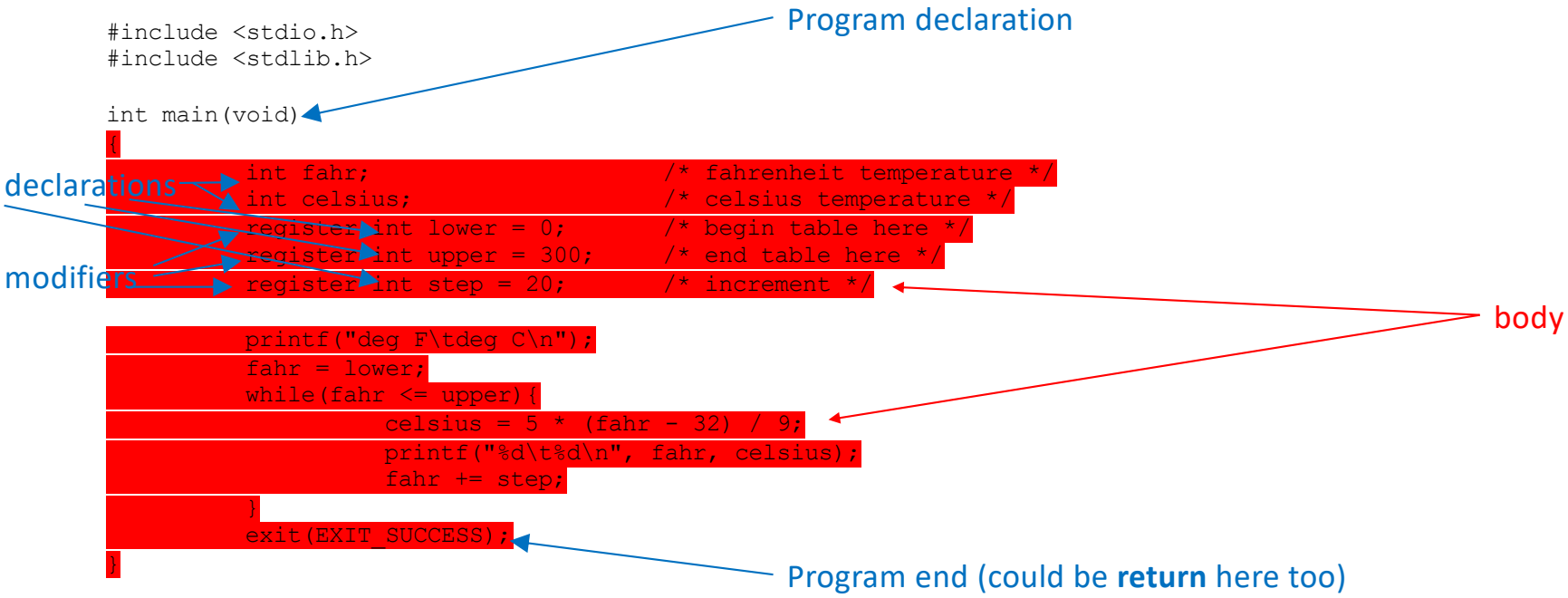
Program declaration

declarations

modifiers

Program end (could be **return** here too)

Example Program: °F to °C Table

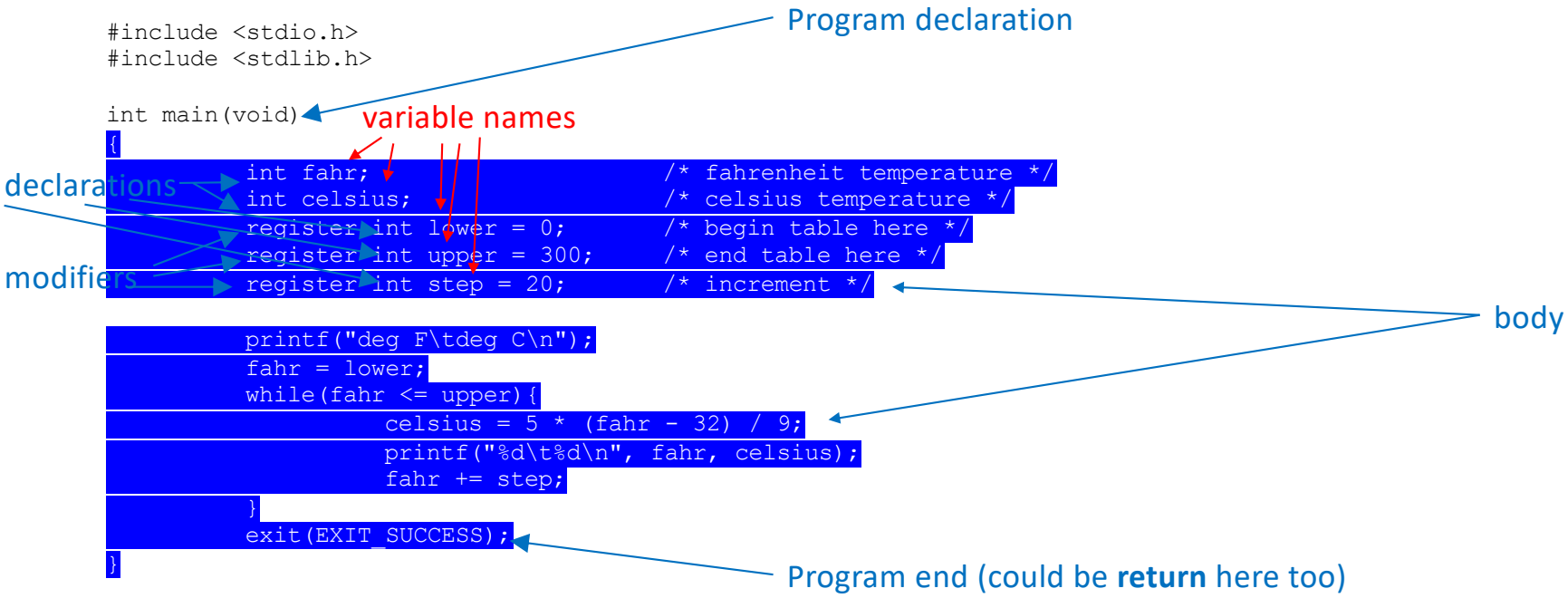



```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int fahr;                       /* fahrenheit temperature */
        int celsius;                    /* celsius temperature */
        register int lower = 0;         /* begin table here */
        register int upper = 300;       /* end table here */
        register int step = 20;         /* increment */

        printf("deg F\tdeg C\n");
        fahr = lower;
        while(fahr <= upper){
                celsius = 5 * (fahr - 32) / 9;
                printf("%d\t%d\n", fahr, celsius);
                fahr += step;
        }
        exit(EXIT_SUCCESS);
}
```

Example Program: °F to °C Table




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

int main(void)
{
        int fahr;                   /* fahrenheit temperature */
        int celsius;                /* celsius temperature */
        register int lower = 0;     /* begin table here */
        register int upper = 300;   /* end table here */
        register int step = 20;     /* increment */

        printf("deg F\tdeg C\n");
        fahr = lower;
        while(fahr <= upper){
                celsius = 5 * (fahr - 32) / 9;
                printf("%d\t%d\n", fahr, celsius);
                fahr += step;
        }
        exit(EXIT_SUCCESS);
}
```

Variable Names

- Must contain only
  - letters (`ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz`)
  - digits (`0123456789`)
  - underscore (`_`)
- Must *not* begin with a digit
- Conventions
  - Don't begin names with an underscore
  - Use lower case (`userinput`)
  - To separate words, use camel case (`userInput`) or underscore (`user_input`)

Basic Types

- `int` (integer)
  - `long`, `short` modifiers: number of bits in the integer
  - `int` is often dropped after such a modifier
  - *only* guarantee is `long` ≥ `int` ≥ `short`
  - `signed`, `unsigned` modifiers: treat `int` as only non-negative or as negative
- `char` (character)
  - Holds 1 character (like '`A`' or '`^`')
  - Really, a very short `int`
  - Can be `signed` or `unsigned` (if treated as an integer)

Basic Types

- `float` (floating point number)
  - Not a real number
  - $\pm m \times 2^e$
  - m is mantissa; e is exponent
- `double` (double precision floating point number)
  - Not a real number
  - Holds about twice as many decimal places as a float

Changing Something to Another Type

- `x` an `int`, `y` a `float`; convert `y` to an `int`: `x = (int) y;`
  - Any fractional part in y is lost
  - So if `y` = 3.6, `x` = 3
- Go the other direction: convert `x` to a `float`: `y = (float) x;`
  - Fractional part of `y` is always 0
  - But the mantissa may be more than 1 integer . . .
  - So if `x` = 3, `y` = 3.0

Arithmetic Operators: +, −, *, /

- Addition, subtraction, multiplication work as you expect
  - 7 + 2 gives 9
  - 7 – 2 gives 5
  - 7 * 2 gives 14
  - If either operand is a double, the result is a double
  - If either operand is a float, the result is a float
  - Otherwise, as both are integers, so is their product
- Division does not:
  - 7 / 2 gives 3 (note it's an integer; the remainder is discarded)
  - 7 / 2.0 gives 3.5 (note it's a float; the remainder is represented as a fraction)

Arithmetic Operators: %

- Remainder: operands must be integers
  - Defined as `r`, where `n % p = r` means `n = ap + r` for some integer `a`
- Problem: what happens if `p` is negative? Look at `5 % -2`
  - `5 = (-2 × -2) + 1`, so `5 % -2` is `1`
  - `5 = (-3 × -2) - 1`, so `5 % -2` is `-1`
- So it's undefined in most versions of C
  - In latest version of C, the result takes on the sign of the dividend
  - Examples: `5 % -2 = 1`, but `-5 % 2 = -1`
  - ***Don't depend on this!!!***

Precedence

- Which operators do you evaluate first?

```
2 + 7 * 9
```

. . . is it `9 * 9 = 81`

. . . is it `2 + 63 = 65`

Precedence

- Which operators do you evaluate first?

```
2 + 7 * 9
```

. . . is it `9 * 9 = 81`

. . . is it `2 + 63 = 65`

Precedence

- Which operators do you evaluate first?

```
2 + 7 * 9
```

. . . is it `9 * 9 = 81`

. . . is it `2 + 63 = 65`

- In C:

`* / %` (multiplicative operators) have higher precedence than

`+ –` (additive operators)

Associativity

- Determines the order in which operators of the same precedence are executed
- Generally operators with the same precedence are evaluated left to right

```
3 * 7 % 2
```

... is it `21 % 2 = 1`

... is it `3 * 1 = 3`

Associativity

- Determines the order in which operators of the same precedence are executed
- Generally operators with the same precedence are evaluated left to right

```
3 * 7 % 2
```

... is it `21 % 2 = 1`

... is it `3 * 1 = 3`

Changing Precedence or Associativity

- Use parentheses
- What is in parentheses is done *first*
- So: `3 * 7 % 2 = 21 % 2 = 1`
- But: `3 * (7 % 2) = 3 * 1 = 3`

Changing Precedence or Associativity

- Use parentheses
- What is in parentheses is done *first*
- So: `3 * 7 % 2 = 21 % 2 = 1`
- But: `3 * (7 % 2) = 3 * 1 = 3`