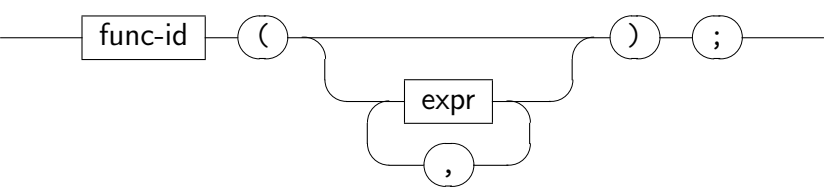
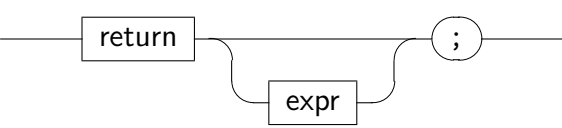


Function Call and Return Statements

function-statement



return-statement



Procedure: Multiple Function Output

- ❑ A function can only return at most one value
- ❑ Example: Given a time duration (in seconds), compute the equivalent number of hours, minutes and seconds
- ❑ Solution #1: do-it-yourself!

```
#include <stdio.h>
int main(void) {
    int t, h, m, s;

    printf("Enter duration (secs): ");
    scanf("%d", &t);

    h = t/3600;
    m = (t%3600)/60;
    s = t%60;

    printf("Duration: %d:%d:%d\n", h,m,s);

    return 0;
}
```

Procedure: Program Output

- ❑ Useful for complex printing tasks

```
#include <stdio.h>
void printTriangle(int n);
void printRowOfStars(int n);
int main(void) {
    int n, i;

    printf("Enter n: ");
    scanf("%d", &n);

    for (i = n; i >= 1; i--) {
        printTriangle(i);
    }
    return 0;
}

void printTriangle(int n) {
    int i, j;
    for (i = n; i > 0; i--) {
        printRowOfStars(i);
    }
    return;
}

void printRowOfStars(int n) {
    int i;
    for (i = 1; i <= n; i++) {
        printf("*");
    }
    printf("\n");
    return;
}
```

Procedure: Multiple Function Output

- ❑ Solution #2: Get a procedure to do it. Does this work?

```
#include <stdio.h>
void splitTime(int t, int h, int m, int s);
int main(void) {
    int t, h, m, s;

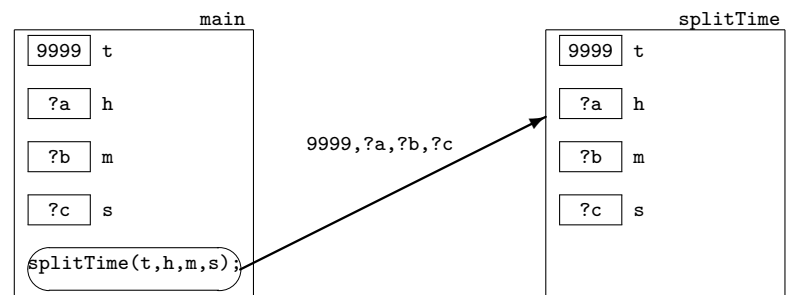
    printf("Enter duration (secs): ");
    scanf("%d", &t);

    splitTime(t,h,m,s);
    printf("Duration: %d:%d:%d\n", h,m,s);
    return 0;
}

void splitTime(int t, int h, int m, int s) {
    h = t/3600;
    m = (t%3600)/60;
    s = t%60;
    return;
}
```

Pass-by-Value Revisited

- Recall pass-by-value



Variable Access Across Functions

- To enable function output via the parameters, use the “into” way of variable access (e.g. `scanf`)

```
int main(void) {
    int t, h, m, s;

    printf("Enter duration (secs): ");
    scanf("%d", &t);

    splitTime(t,&h,&m,&s);

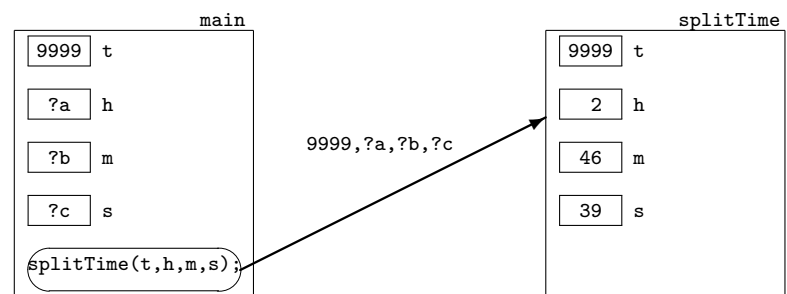
    printf("Duration: %d:%d:%d\n", h,m,s);

    return 0;
}
```

- How to define parameters of the `splitTime` function?

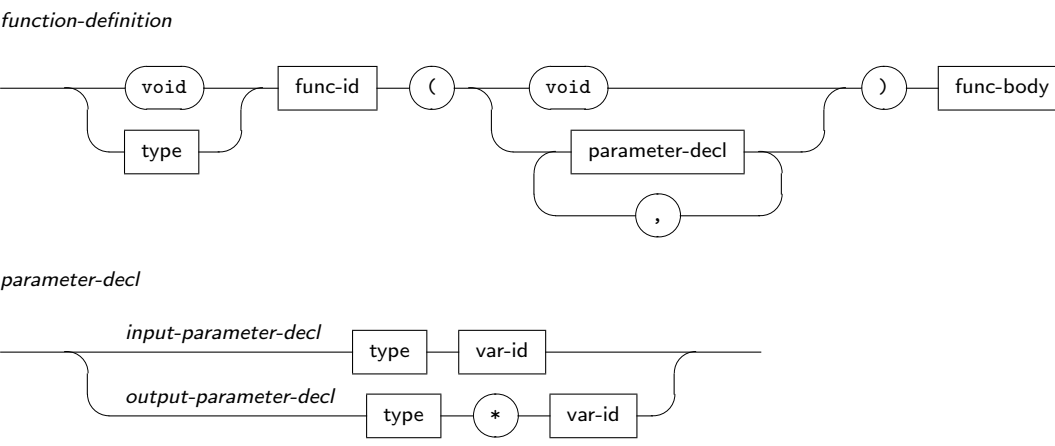
Pass-by-Value Revisited

- Just before `splitTime` function returns



- What happens to the variables in the `main` function?

Function Output Parameter



Function Output Parameter

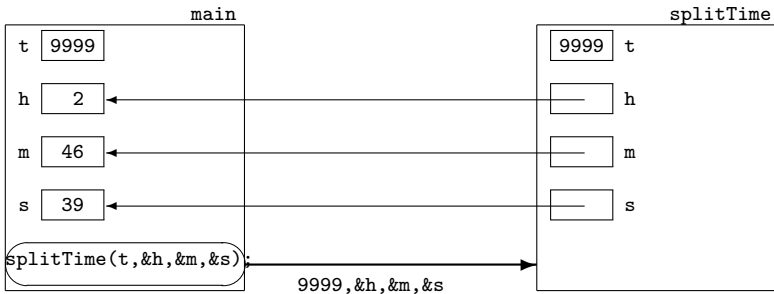
- Output parameter declared with * in the function header
- Output parameter accessed using * in the function body

```
/*
  Function splitTime takes t in seconds, splits and
  outputs the hours, minutes and seconds through the
  output parameters h, m and s.

  Precondition: t >= 0
*/
void splitTime(int t, int *h, int *m, int *s) {
    *h = t/3600;
    *m = (t%3600)/60;
    *s = t%60;
    return;
}
```

Function Output Parameter

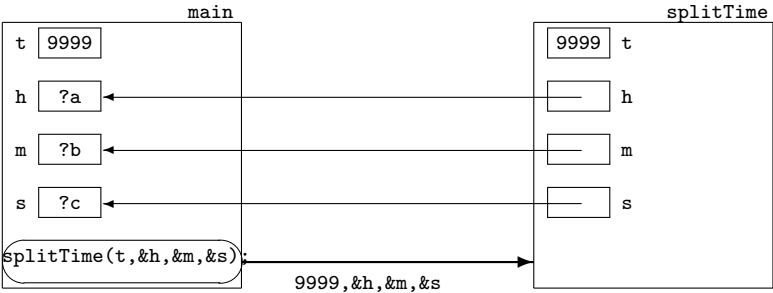
- Just before splitTime function returns



- Values are “returned” through the function output parameters; does not violate lexical scoping

Function Output Parameter

- Calling splitTime with function output parameters



- Passing &h, &m and &s to splitTime gives it access to variables h, m and s in main

Example: Swapping Variable Contents

- Using main function to swap the contents of two variables

```
#include <stdio.h>

int main(void) {
    int x, y, temp;

    printf("Enter x and y: ");
    scanf("%d%d", &x, &y);

    temp = x;
    x = y;
    y = temp;

    printf("x=%d; y=%d\n", x, y);
    return 0;
}
```

Example: Swapping Variable Contents

- Using a swap function

```
#include <stdio.h>

void swap(int *x, int *y);

int main(void) {
    int x, y;

    printf("Enter x and y: ");
    scanf("%d%d", &x, &y);

    swap(&x,&y);

    printf("x=%d; y=%d\n", x, y);

    return 0;
}
```

```
void swap(int *x, int *y) {
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;

    return;
}
```

17 / 24

Using only the main function

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

int main(void) {
    int n=0;
    double data, sum=0, sumSq=0, mean, stdev;

    scanf("%lf", &data);
    while (data >= 0) {
        sum = sum + data;
        sumSq = sumSq + (data * data);
        n++;
        scanf("%lf", &data);
    }

    if (n > 0) {
        mean = sum/n;
        stdev = sqrt((sumSq - (sum*sum/n))/(n));
        printf("mean=%f; stdev=%f\n", mean, stdev);
    } else {
        printf("No data\n");
    }

    return 0;
}
```

19 / 24

Example: Finding Mean and Standard Deviation

- Example: Given $n (\geq 0)$ non-negative floating-point values, find the mean μ and standard deviation σ

$$\mu = \frac{\sum x_i}{n}$$
$$\sigma = \sqrt{\frac{\sum (x_i^2) - \frac{(\sum x_i)^2}{n}}{n}}$$

- Finding μ and σ requires the sum $\sum x_i$, as well as sum of squares $\sum (x_i^2)$
- Use sentinel-controlled input to read values

18 / 24

Modularizing readData

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

void readData(int *n, double *sum, double *sumSq);

int main(void) {
    int n=0;
    double sum=0, sumSq=0, mean, stdev;

    readData(&n,&sum,&sumSq);

    if (n > 0) {
        mean = sum/n;
        stdev = sqrt((sumSq - (sum*sum/n))/(n));
        printf("mean=%f; stdev=%f\n", mean, stdev);
    } else {
        printf("No data\n");
    }

    return 0;
}
```

20 / 24

Modularizing readData

readData reads values until a sentinel (< 0) and outputs n (no. of values read), sum and sumSq (sum of square values)

Precondition: none

Postcondition: n >= 0, sum >= 0, sumSq >= 0

```
*/
void readData(int *n, double *sum, double *sumSq) {
    double data;

    *n = 0; *sum = 0; *sumSq = 0;
    scanf("%lf", &data);
    while (data >= 0) {
        *sum = *sum + data;
        *sumSq = *sumSq + (data*data);
        (*n)++;

        scanf("%lf", &data);
    }

    return;
}
```

21 / 24

Modularizing findStats

/*
findStats outputs the mean and stdev of n values
given sum and sumSq (sum of square values).

Precondition: n > 0

```
*/
void findStat(int n,
              double sum, double sumSq,
              double *mean, double *stdev) {
    *mean = sum / n;
    *stdev = sqrt((sumSq - (sum*sum)/n)/(n));

    return;
}
```

23 / 24

Modularizing findStats

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

void readData(int *n, double *sum, double *sumSq);
void findStat(int n, double sum, double sumSq,
              double *mean, double *stdev);

int main(void)
{
    int n=0;
    double sum=0, sumSq=0, mean, stdev;

    readData(&n,&sum,&sumSq);

    if (n > 0) {
        findStat(n,sum,sumSq,&mean,&stdev);
        printf("mean=%f; stdev=%f\n", mean, stdev);
    } else {
        printf("No data\n");
    }

    return 0;
}
```

22 / 24

Lecture Summary

- Application of user-defined functions as value-returning functions or procedures
- Use of function output parameters to simulate “multiple return values”
 - Are value-returning functions that return a single value no longer necessary?
 - How about function compositions?

```
double cos(double x);
double sqrt(double x);
```

allows the following expression to be defined:

```
sqrt((b * b) + (c * c) -
      (2 * b * c * cos(alpha * PI / 180)));
```

24 / 24