

Foundations of C Programming (Structured Programming) - Making Decisions

Outline

- C Statements
- Conditional Statements

C Statements

- In the most general sense, a **statement** is a part of your program that can be executed.
- An expression is a statement (**simple statement**).

```
a = a+1;
```

```
a--;
```

- A function call is also a statement (more about function call will be introduced later).

```
printf("%d", a);
```

- A compound statement consists of several expressions and statements
 - Conditional statements
 - Loop statements

C Statements

- C is a free form language, so you may type the statements in any style you feel comfortable:

```
a=  
a+  
1; a--;      // line breaks can be anywhere
```

- However, this style is not suggested

Conditional Statements

- A conditional statement allows us to control whether a program segment is executed or not.
- Two constructs
 - *if* statement
 - *if*
 - *if-else*
 - *if-else if*
 - *switch* statement

An Example

```
if (it is sunny) {  
    go to beach;  
    swim;  
}  
go to library;
```

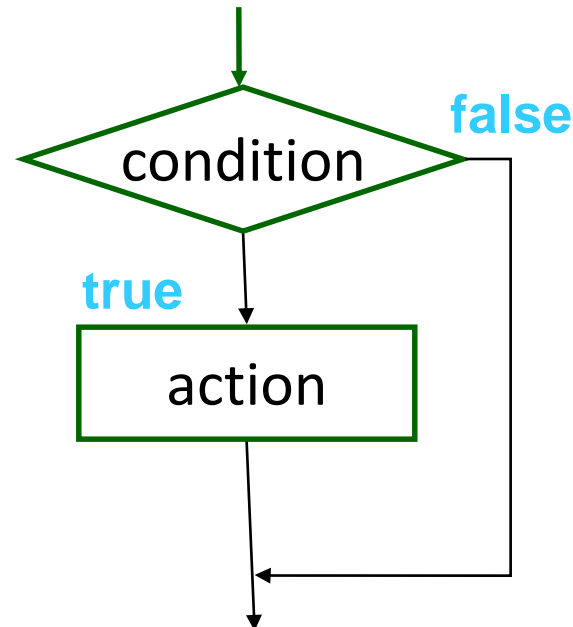
What does this mean?

The Basic If Statement

Syntax:

```
if (<condition>) {  
    <action>  
}
```

Flowchart:



- ◆ If the condition is **true** then execute the action.
- ◆ **action** is either a single statement or a group of statements within curly brackets.

An Example

```
/* program to read number and print out its absolute
   value */
#include<stdio.h>
int main(){
    int value;
    printf("Please enter an integer:");
    scanf("%d", &value);
    if(value < 0)
        value = -value;
    printf("The absolute value is %d.\n", value);
    return 0;
}
```

What does this program want to do?

Attention: You will lose marks in assignments or labs if there is no indentation in conditional statement.

An Example

```
/* program to read number and print out its absolute
   value */
#include<stdio.h>
int main() {
    int value;
    printf("Please enter an integer:");
    scanf("%d", &value);
    if(value < 0) {
        value = -value;
    }
    printf("The absolute value is %d.\n", value);
    return 0;
}
```

What is the problem if a pair { } are added?

Condition

- a condition (decision) can have one of two values:
 - **true** (corresponds to a **non-zero** value)
 - E.g., $10, 10 > 5$ all have the result `true` (1) if they are a condition.
 - **false** (corresponds to **zero** value)
 - E.g., $0, 10 < 5$ have the result `false` (0) if they are a condition

'=' and '=='

- Compare these two programs

```
int a;  
scanf("%d", &a);  
if (a == 10)  
    printf("a is 10");
```

```
int a;  
scanf("%d", &a);  
if (a = 10)  
    printf("a is 10");
```

If the input is 5, what are the outputs of these two programs?

If the input is 10, what are the outputs of these two programs?

'=' and '=='

- Compare these two programs

```
int a;  
scanf("%d", &a);  
if (a == 10)  
    printf("a is 10");
```

```
int a;  
scanf("%d", &a);  
if (a = 10)  
    printf("a is 10");
```

Input: 5

Output:

Input: 10

Output: a is 10

Input: 5

Output: a is 10

Input: 10

Output: a is 10

An Example

```
int a;  
scanf("%d", &a);  
if (a <= 10 && a >= 5)  
    printf("a is between 5 and 10");
```

Use Float in Condition

- A trap
 - Be careful when use != and == with float numbers.

```
float f1 = 1.23, f2 = 1.25;

printf("%f\n", f1 - 1.23);
if (f1 == 1.23) printf("f1 == 1.23 OK\n");
if (f2 == 1.25) printf("f2 == 1.25 OK\n");
```

Output:

```
0.000000
f2 == 1.25 OK
```

May never be true in a machine due to the float data expression

An Example

Sorting two numbers:

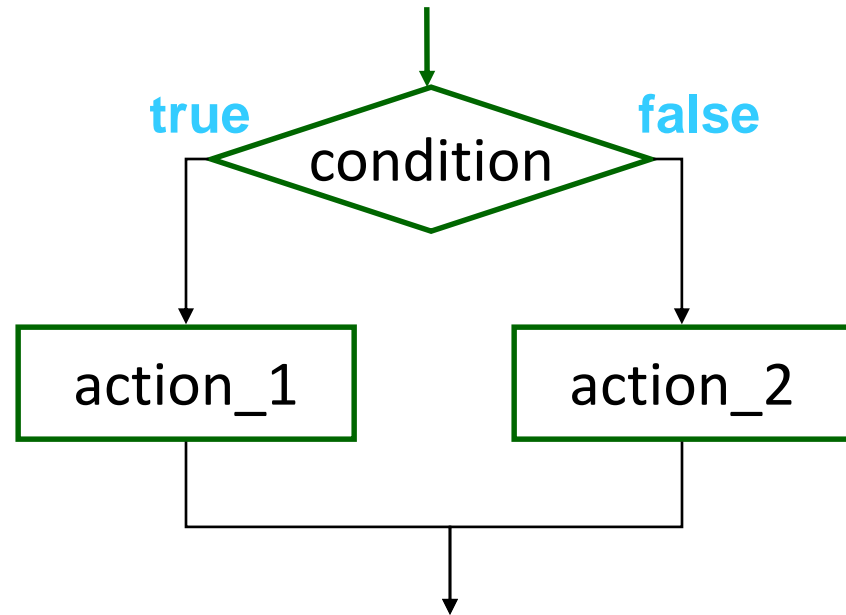
```
int value1;  
int value2;  
int temp;  
printf("Enter two integers:");  
scanf("%d%d",&value1,&value2);  
if(value1 > value2){  
    temp = value1;  
    value1 = value2;  
    value2 = temp;  
}  
printf("The input in sorted order: %d %d ",  
value1,value2);
```

The Basic If – else Statement

Syntax:

```
if (<condition>) {  
    <action_1>  
}  
else {  
    <action_2>  
}
```

Flowchart:



- ◆ if the condition is **true** then execute **action_1**; otherwise, execute **action_2**.
- ◆ **action_1** and **action_2** are either a single statement or a group of statements within curly brackets.

An Example

```
if (it is sunny){  
    go to beach;  
    swim;  
}  
go to library;
```

```
if (it is sunny){  
    go to beach;  
    swim;  
}  
else  
    go to library;
```

What is the difference between these code segments?

An Example

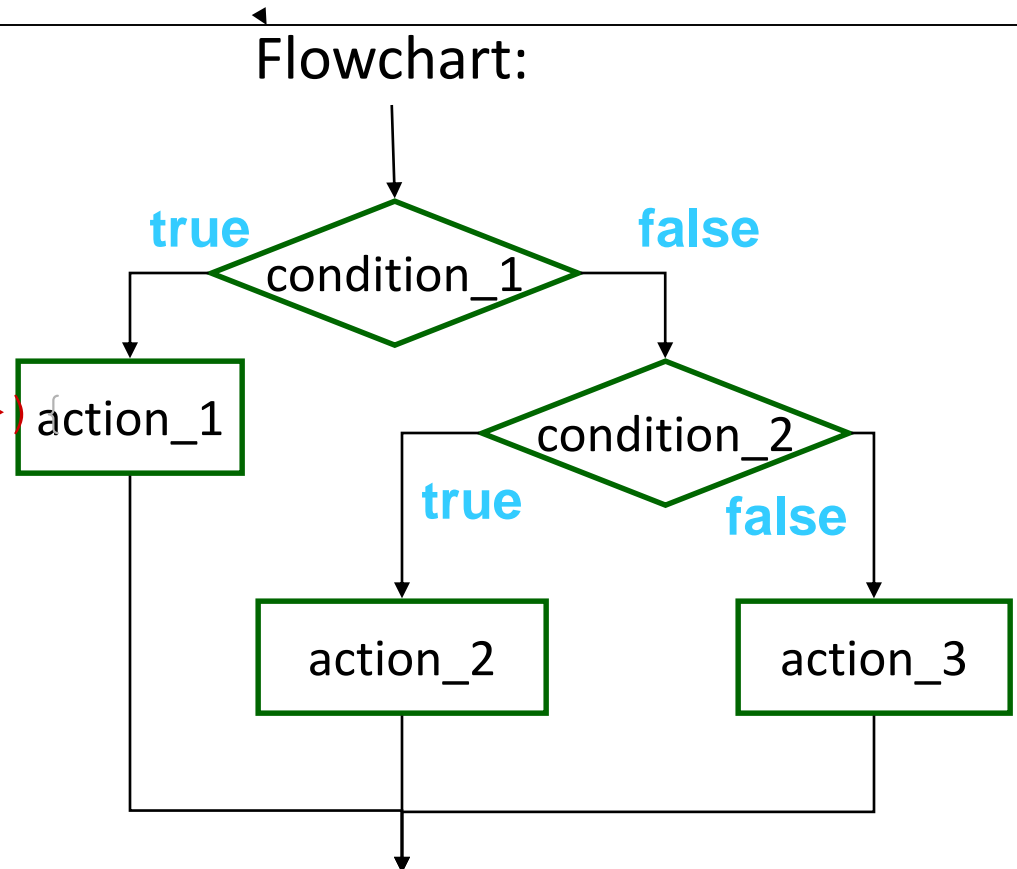
```
int value1;
int value2;
int larger;
printf("Enter two integers: ");
scanf("%d%d",&value1,&value2);
if(value1 > value2)
    larger = value1;
else
    larger = value2;
printf("Larger of inputs is: %d.\n",larger);
```

The Basic If – else – if Statement

Syntax:

```
if (<condition_1>) {  
    <action_1>  
}  
else if (<condition_2>) {  
    <action_2>  
}  
else {  
    <action_3>  
}
```

Flowchart:



Class Exercise

A calculator (for only +, - * and /):

```
char op;
int x, y;
scanf("%d%c%d",&x, &op, &y);
if(op == '+')
    printf("%d + %d = %d", x, y, x + y);
else if(op == '-')
    printf("%d - %d = %d", x, y, x - y);
else if(op == '*')
    printf("%d * %d = %d", x, y, x * y);
else if(op == '/')
    printf("%d / %d = %d", x, y, x / y);
else printf("Invalid operator!");
```

Can you draw a flowchart for this program?

After Class Exercise

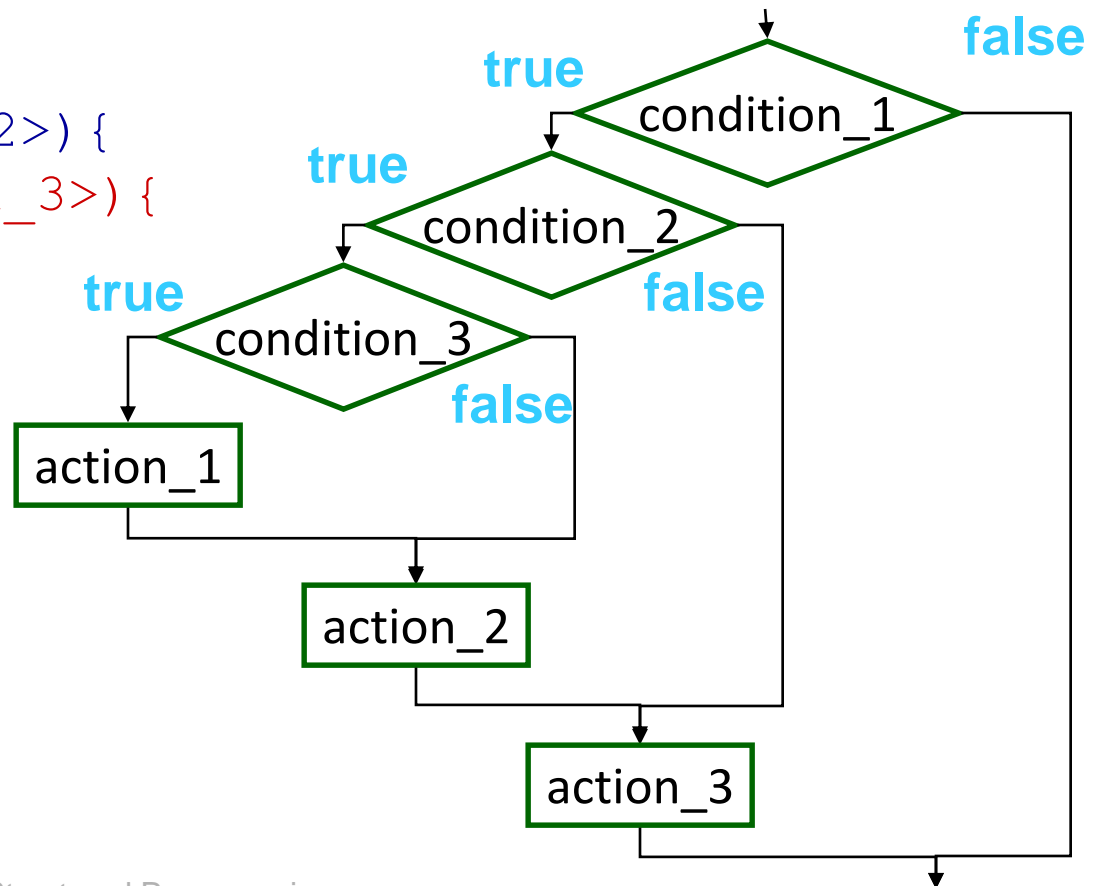
```
#include <math.h>
#include <stdio.h>
int main() {
    double a, b, c, discriminant, root1, root2, realPart, imagPart;
    printf("Enter coefficients a, b and c: ");
    scanf("%lf %lf %lf", &a, &b, &c);
    discriminant = b * b - 4 * a * c; // condition for real and different roots
    if (discriminant > 0) {
        root1 = (-b + sqrt(discriminant)) / (2 * a);
        root2 = (-b - sqrt(discriminant)) / (2 * a);
        printf("root1 = %.2lf and root2 = %.2lf", root1, root2);
    } // condition for real and equal roots
    else if (discriminant == 0) {
        root1 = root2 = -b / (2 * a);
        printf("root1 = root2 = %.2lf;", root1);
    } // if roots are not real
    else {
        realPart = -b / (2 * a);
        imagPart = sqrt(-discriminant) / (2 * a);
        printf("root1 = %.2lf+%.2lfi and root2 = %.2f-%.2fi", realPart, imagPart, realPart, imagPart);
    }
    return 0;
}
```

What is the purpose of this program?

Nested if Statements

- Nested means that one complete statement is inside another
- Flowchart:

```
if (<condition_1>) {  
  if (<condition_2>) {  
    if (<condition_3>) {  
      <action_1>  
    }  
    <action_2>  
  }  
  <action_3>  
}
```



Examples: Are They the Same?

```
if (membership) {  
    if (age < 18)  
        fee = fee * 0.5;  
    if (age >= 18)  
        fee = fee * 0.8;  
}
```

```
if (membership) {  
    if (age < 18)  
        fee = fee * 0.5;  
}  
if (age >= 18)  
    fee = fee * 0.8;
```

```
if (membership) {  
    if (age < 18)  
        fee = fee * 0.5;  
    else  
        fee = fee * 0.8;  
}
```

```
if (membership)  
    if (age < 18)  
        fee = fee * 0.5;  
    else  
        fee = fee * 0.8;
```

"Dangling Else" Problem

```
if (membership) {  
    if (age < 18)  
        fee = fee * 0.5;  
    if (age >= 18)  
        fee = fee * 0.8;  
}
```

||

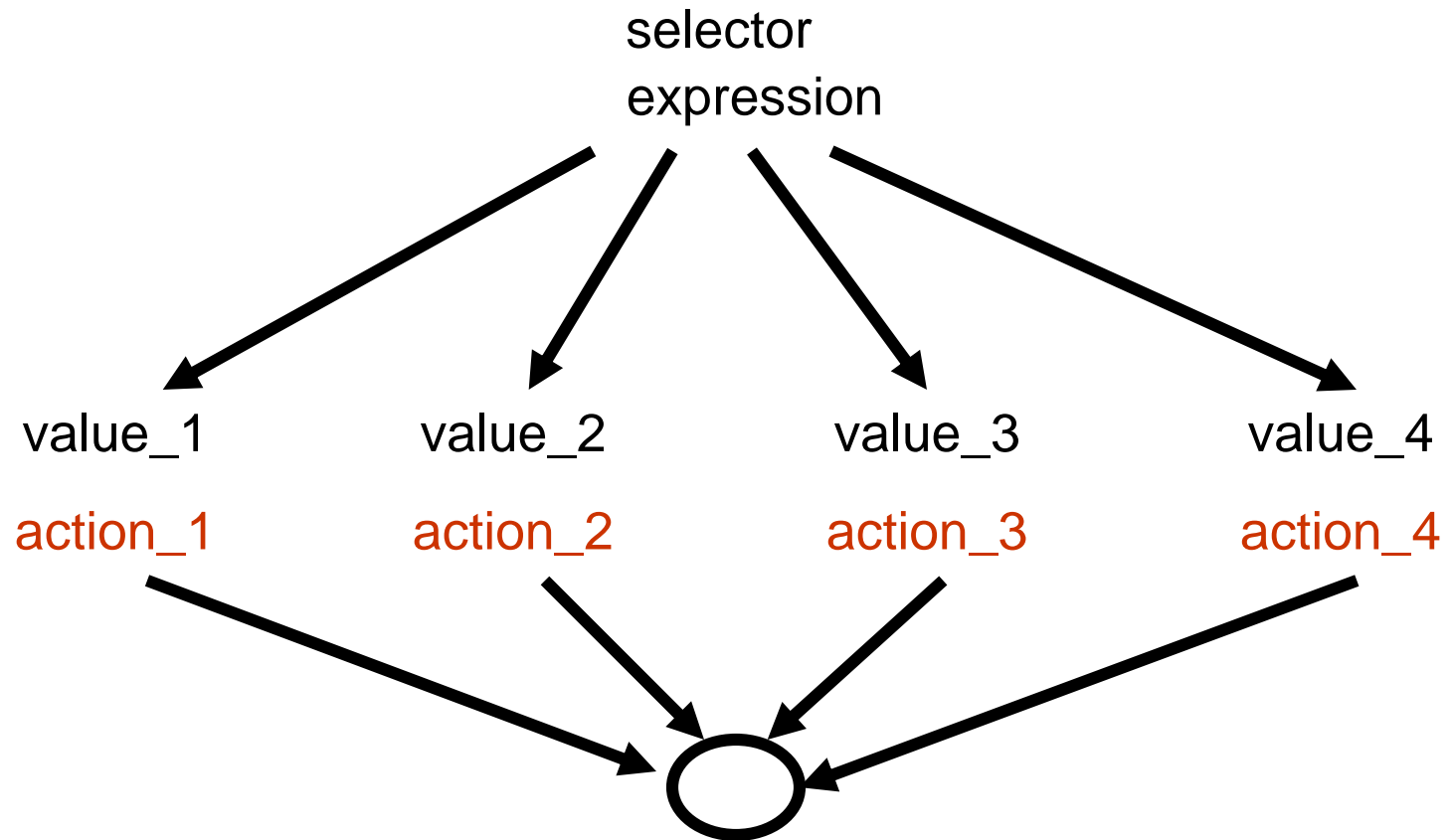
```
if (membership) {  
    if (age < 18)  
        fee = fee * 0.5;  
    else  
        fee = fee * 0.8;  
}
```

=

```
if (membership)  
    if (age < 18)  
        fee = fee * 0.5;  
    else  
        fee = fee * 0.8;
```

else matches to the *if* closest to it

Multi-way Selection: Switch Statement



Switch Statement

Syntax:

```
switch (<selector expression>) {  
    case <value_1>: <action_1>;  
                    break;  
    case <value_2>: <action_2>;  
                    break;  
    case <value_3>: <action_3>;  
                    break;  
    default: <action_4>;  
}
```

Meaning:

- Evaluate selector expression.
- Match case label.
- Execute sequence of statements of matching label.
- If **break** encountered,
go to end of the **switch** statement;
otherwise continue execution.

Switch Statement

```
switch (< selector expression>) {  
    case <value_1>: <action_1>;  
                    break;  
    case <value_2>: <action_2>;  
                    break;  
    case <value_3>: <action_3>;  
                    break;  
    default: <action_4>;  
}
```

- **Attentions**

- The expression must be an integral expression.
- The value following each **case** label must be a constant.
- No two **case** labels have the same value.
- Two **case** labels may be associated with the same statements.
- Remember to include the **break** statement at the end of each case.
- The **default** label is not compulsory.
- There can be only one **default** label, and it is usually last.

An Example

```
char op;
int x, y;
scanf("%d%c%d",&x, &op, &y);
if(op == '+')
    printf("%d + %d = %d", x, y, x + y);
else if(op == '-')
    printf("%d - %d = %d", x, y, x - y);
else if(op == '*')
    printf("%d * %d = %d", x, y, x * y);
else if(op == '/')
    printf("%d / %d = %d", x, y, x / y);
else printf("Invalid operator!");
```

Can we rewrite this program using switch statement?

An Example

```
char op;
int x, y;
scanf("%d%c%d",&x, &op, &y);
switch (op) {
    case '+': printf("%d + %d = %d", x, y, x + y);
               break;
    case '-': printf("%d - %d = %d", x, y, x - y);
               break;
    case '*': printf("%d * %d = %d", x, y, x * y);
               break;
    case '/': printf("%d / %d = %d", x, y, x / y);
               break;
    default:  printf("Invalid operator!");
}
}
```

Class Exercise

```
switch((int)score/10){  
    case 10:  
    case 9: printf("Grade = A\n");  
            break;  
    case 8: printf("Grade = B\n");  
            break;  
    case 7: printf("Grade = C\n");  
            break;  
    case 6: printf("Grade = D\n");  
            break;  
    default: printf("Grade = F\n");  
}
```

- What is the output of this program if score is 95?
- What if all the "break" are removed?
- How can we use if-else if statement to rewrite this program?

Summary

- Make decisions in a program
 - if
 - if-else
 - if-else if
 - switch
- Switch statements can be used at some fixed values
- Pay attention to the difference between = and ==
- Be careful in deciding the usage of < and <=, > and >=
- Decision statements are important in programming