Program Control 1

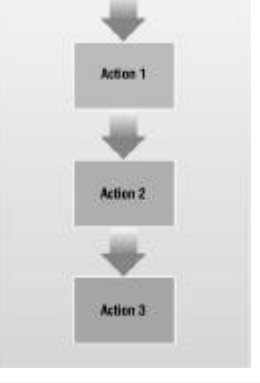
Control Structures

- Sequential execution
  - Statements executed one after the other in the order written
- Transfer of control
  - When the next statement executed is not the next one in sequence
  - Overuse of **goto** statements led to many problems

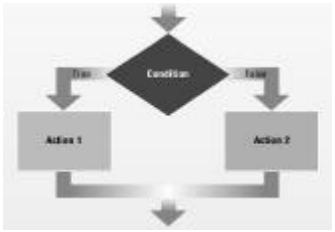
Control Structures

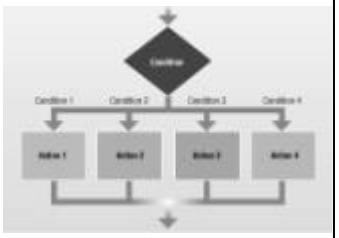
- All programs written in terms of 3 control structures
  - Sequence structures: Built into C. Programs executed sequentially by default
  - Selection structures: C has three types: if, if/else, and switch
  - Repetition structures: C has three types: while, do/while and for

Sequence Structure

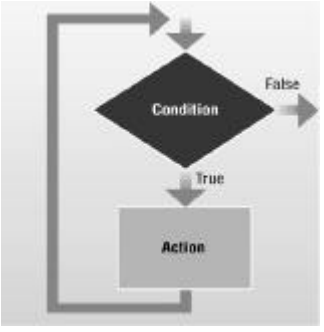


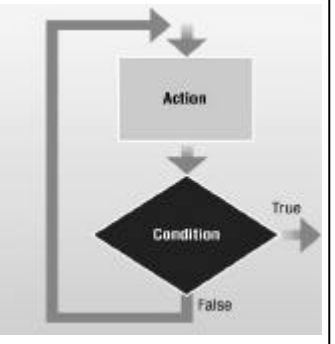
Selection Structure





Repeating Structure





The if/else Selection Structure

- Simple Structure in C:
  - if (grade >= 60)
    - printf("Passed\n");
  - else
    - printf("Failed\n");

The if/else Selection Structure

- Ternary conditional operator (?:)
  - Takes three arguments (condition, value if true, value if false)
  - Our pseudocode could be written:

    printf("%s\n", grade >= 60 ? "Passed" : "Failed");
  - Or it could have been written:

    grade >= 60 ? printf(“Passed\n”) : printf(“Failed\n”);

The if/else Selection Structure

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
  printf( "Passed.\n" );
else {
  printf( "Failed.\n" );
  printf( "You must take this course again.\n" );
}
```

  - Without the braces, the statement

```
printf( "You must take this course again.\n" );
```

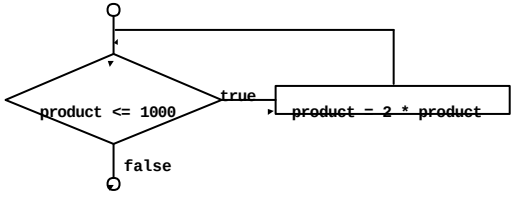
  - would be executed automatically

The while Repetition Structure

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains true
  - while loop repeated until condition becomes false

The while Repetition Structure

- Example:
  - int product = 2;
  - while (product <= 1000)
    product = 2 * product;



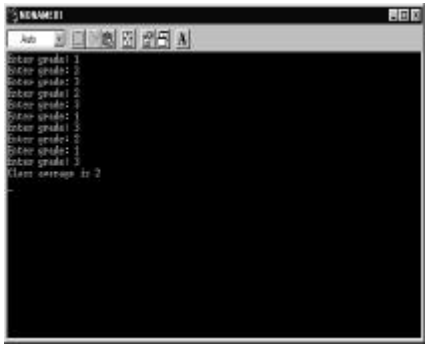
Example

```
/* 2 A: Class average program with counter-controlled repetition */
#include<stdio.h>
#include<conio.h>
int main()
{
   int counter, grade, total, average;
    /* initialization phase */
   total = 0;
   counter = 1;

   /* processing phase */
   while ( counter <= 10 ) {
     printf( "Enter grade: " );
     scanf( "%d", &grade );
     total = total + grade;
     counter = counter + 1;
   }
```

Example (cont')

```
    /* termination phase */
    average = total / 10;
    printf( "Class average is %d\n", average );
    getch ();
    return 0;   /* indicate program ended successfully */
}
```


Output Result



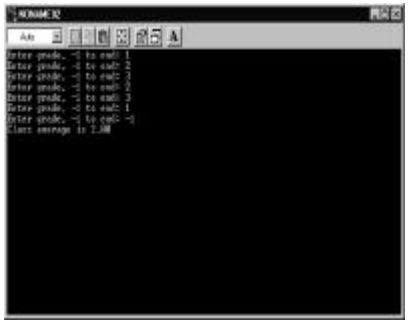
Example

```
/* 2 B: Class average program with  sentinel-controlled repetition */
#include<stdio .h>
#include<conio .h>
int  main()
{
   float average;           /* new data type */
   int counter, grade, total;

   /* initialization phase */
   total = 0;
   counter = 0;

   /* processing phase */
   printf( "Enter grade, -1 to end: " );
   scanf( "%d", &grade );
```


Example (cont')

```
while ( grade != -1 ) {
      total = total + grade;
      counter = counter + 1;
      printf( "Enter grade, -1 to end: " );
      scanf("%d", &grade);
   }

   /* termination phase */
   if ( counter != 0 ) {
     average = ( float ) total / counter;
     printf( "Class average is %.2f", average );
   }
   else
     printf( "No grades were entered\n" );
   getch ();
   return 0;   /* indicate program ended successfully */
}
```

Output Result



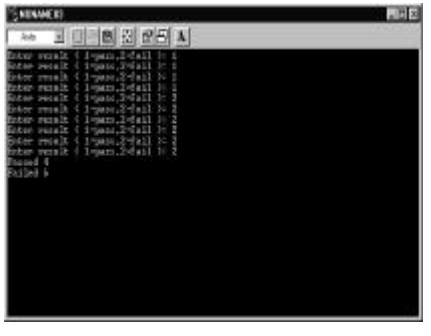
Example

```
/* 2 C: Analysis of examination results */
#include<stdio .h>
#include<conio .h>
int main()
{
  /* initializing variables in declarations */
  int passes = 0, failures = 0, student = 1, result;

  /* process 10 students; counter-controlled loop */
  while ( student <= 10 ) {
    printf( "Enter result ( 1=pass,2=fail ): " );
    scanf( "%d", &result );

    if ( result == 1 )      /* if/else nested in while */
      passes = passes + 1;
    else
      failures = failures + 1;

    student = student + 1;
  }
```


Example (cont’)

```
  printf( "Passed %d\n", passes );
    printf( "Failed %d\n", failures );

    if ( passes > 8 )
      printf( "Raise tuition\n" );
    getch ();
    return 0;   /* successful termination */
  }
```


Output Result



Assignment Operators

- Assignment operators abbreviate assignment expressions
  `c = c + 3;`
  can be abbreviated as `c += 3;` using the addition assignment operator
- Statements of the form
  variable **=** *variable operator expression*;
  can be rewritten as
  variable **`operator=`** *expression*;
- Examples of other assignment operators:

```
d -= 4      (d = d - 4)
e *= 5      (e = e * 5)
f /= 3      (f = f / 3)
g %= 9      (g = g % 9)
```


Increment and Decrement Operators

- Increment operator (**`++`**)
  - Can be used instead of **`c+=1`**
- Decrement operator (**`--`**)
  - Can be used instead of **`c-=1`**
- Preincrement
  - Operator is used before the variable (**`++c`** or **`--c`**)
  - Variable is changed before the expression it is in is evaluated
- Postincrement
  - Operator is used after the variable (**`c++`** or **`c--`**)
  - Expression executes before the variable is changed

Increment and Decrement Operators

- If **`c`** equals **5**, then

```
printf( "%d", ++c );
```

  - Prints **6**

```
printf( "%d", c++ );
```

  - Prints **5**
  - In either case, **`c`** now has the value of **6**

c = c + 1;
printf(%d, c);

printf(%d, c);
c = c + 1;

- When variable not in an expression
  - Preincrementing and postincrementing have the same effect

```
++c;
printf( “%d”, c );
```

  - Has the same effect as

```
c++;
printf( “%d”, c );
```