Programming Fundamentals

Module C - Logic

Le The Anh
anhlt161@fe.edu.vn

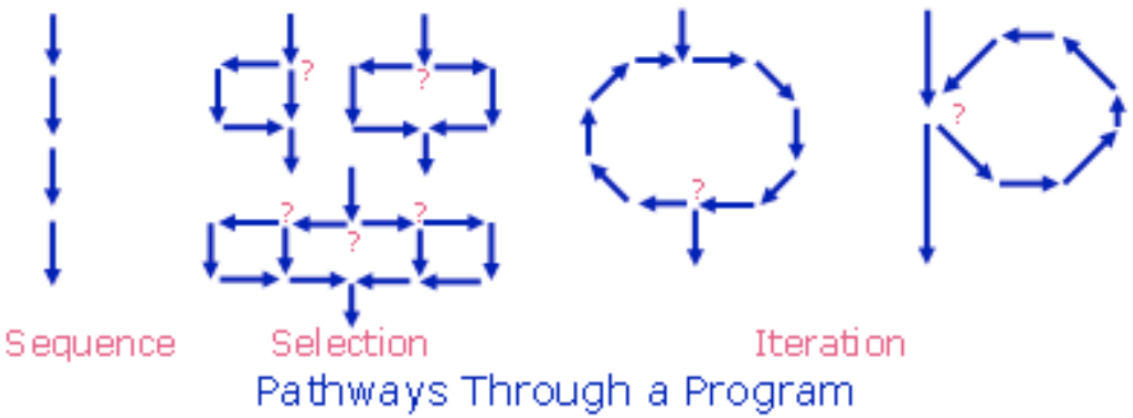
FPT UNIVERSITY

Objectives

1. Logic Constructs
   - Structured Programming
   - Sequence Constructs
   - Selection Constructs
   - In-Class Practice
   - Iteration
   - In-Class Practice
   - In-Class Practice
   - Dangling Else
2. Programming Style
   - Naming
   - Indentation
   - Comments
   - Magic Values
   - General Guidelines
3. Walkthroughs
   - Memory Map
   - Walkthrough Table
   - Example

Logic Constructs

- Expressions enable us to write programs that perform calculations and execute statements in a sequential order.
- To write programs that execute different statements depending upon the satisfaction of certain conditions, we introduce selection constructs.
- To write programs that keep executing sets of statements until certain condition are satisfied, we introduce iteration constructs.



Pathways Through a Program

Structured Programming

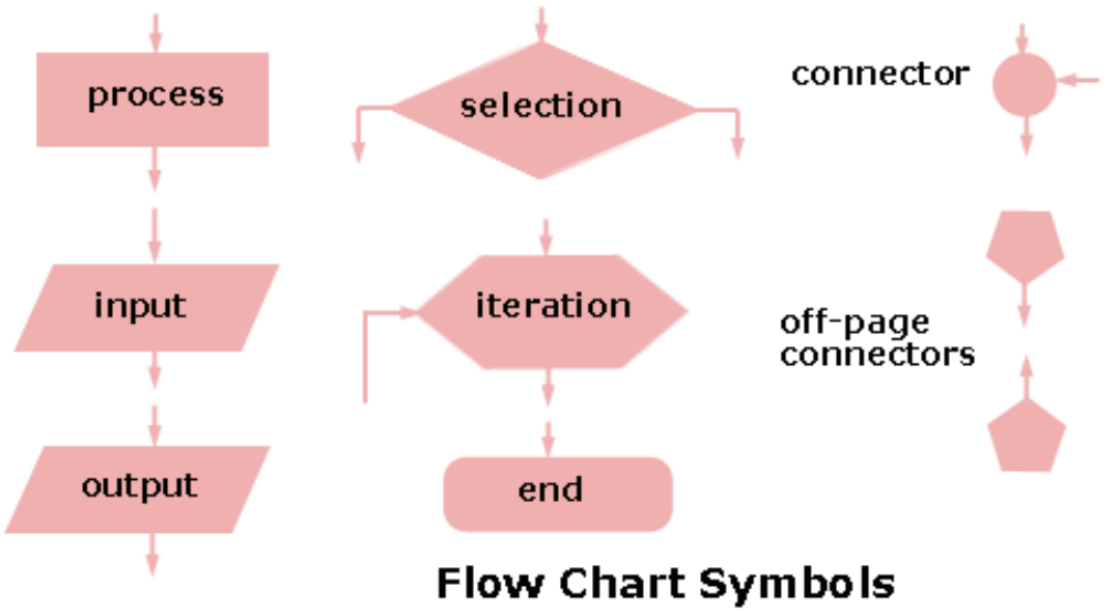
- Are understandable, testable and readily modifiable.
- Consist of simple logical constructs, each of which has one entry point and one exit point.
- The design of a program can be described using:
  - pseudo-coding or
  - flow charting

Pseudo-coding

- Is a shorthand that itemizes key steps in the flow of a program
- For example: Pseudo-code of the absolute calculator
  1. prompt the user for an integer value
  2. accept an integer value from the user
  3. if the value is positive, store the value in x
  4. if the value is negative, store the negative of the value in x
  5. display the value of x

Flow Charts

- Describe the flow of a program unit symbolically



Three Kinds of Constructs

- Sequence constructs
- Selection constructs
- Iteration constructs

Sequence Constructs

- A sequence is either a simple statement or a code block.
- Simple Statements

```
expression;
change = 28;  /* assigns 28 to the variable change */
```

- Code Blocks: is a set of statements enclosed in curly braces.

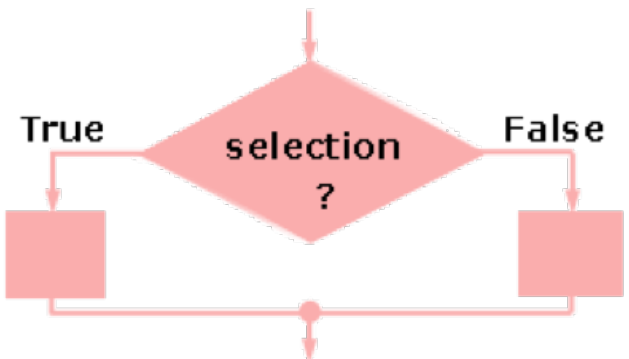
```
{
    statement
    ...
    statement
}

{
    temp = a;
    a = b;
    b = temp;
}
```

Selection Constructs

The selection constructs are:

- if
- if else
- if else if else
- switch
- ?:



Selection Construct - if

- Syntax:

```
if (condition)
    statement(s)
```

- Example:

```
if(a%2 == 0)
    printf("%d is an even number.", a);

if(a > b)
{
    temp = a;
    a = b;
    b = temp;
}
```

Selection Construct - if else

- Syntax:

```
if (condition)
    statement(s) 1
else
    statement(s) 2
```

- Example:

```
if(a%2 == 0)
    printf("%d is an even number.", a);
else
    printf("%d is an odd number.", a);

if(a > b)
{
    temp = a;
    a = b;
    b = temp;
}
else
    printf("%d is not greater than %d", a, b);
```

Selection Construct - if else-if else

- Syntax:

```
if (condition 1)
    statement(s) 1
else if (condition 2)
    statement(s) 2
...

else
    statement(s) n
```

- Example:

```
if(mark > 79)
    printf("Great! Congrat!");
else if (mark > 59)
    printf("Good job! You passed the exam");
else
    printf("Ups .. See you in the summer semester!");
```

Selection Construct - Switch

- Syntax:

```
switch (variable or expression)
{
    case constant 1:
        statement(s) 1
        break;
    case constant 2:
        statement(s) 2
        break;
    ...

    default:
        statement(s) n
}
```

- Notes:
  - The break statement shifts control to the end of the switch construct.
  - No braces are needed around the statements between the case labels.
  - The default case is optional.

Switch Example

```
char myChoice;
double cost;

printf("Choose a candy (a, b or c) ? ");
scanf("%c", &myChoice);

switch ( myChoice ) {
case 'A':
case 'a':
    printf ("You selected candy A\n");
    cost = 1.50;
    break;
case 'B':
case 'b':
    printf ("You selected candy B\n");
    cost = 0.75;
    break;
case 'C':
case 'c':
    printf ("You selected candy C\n");
    cost = 1.10;
    break;
default:
    printf("Sorry, we don't have your selection. Try again.\n");
    cost = 0.0;
}
```

Conditional Expression

- Syntax:

  ```
  condition ? expression if true : expression if false
  ```

- Example:

  ```
  c = a > b ? a : b;
  ```

In-Class Practice

1. Write a program to find the largest number among three numbers.
2. Write a program to find the smallest odd number among four numbers.
3. Complete the program for the selection problem described in the Handout on Logic Constructs (on the given offline website).

Iteration

- while
- do while
- for

Iteration - while

- Syntax:

```
while (condition)
    statement(s)
```

- Example:

```
int i = 1, s = 0;

while(i <= 10)
{
    if(i%2 == 0)
        s += i;
    i++;
}
printf("Sum of even numbers in range [1, 10] is %d", s);
```

Iteration - do while

- Syntax:

```
do
    statement(s)
while (condition);
```

- Example:

```
int pin_code = 1234;
int entered_pin_code;

do
{
    printf("Enter four-number pin code: ");
    scanf("%d", &entered_pin_code);

    if(entered_pin_code != pin_code)
        printf("Wrong pin code. Please try again!\n\n");
}
while (entered_pin_code != pin_code);
```

Iteration - for

- Syntax:

```
for(initialization; condition; change)
    statement(s)
```

- Example:

```
int i, s = 0;

for(i=0; i < 10; i++)
    printf("%d ", i);

for(i=9; i >= 0; i -= 2)
    printf("%d ", i);

for(i=1; i < 10; s += i, i += 2);
    printf("%d", s);
```

In-Class Practice

1. Write a program that takes as input three integers a, b, c and find the largest number in range [a, b] that is divisible by c.
2. An integer is a palindrome when it reads the same backward as forward. For example, 121 is palindrome while 123 is not. Write a program to check whether a number is palindrome.

Flags

- The one entry, one exit principle is fundamental to structured programming.
- C includes three keywords that allow jumps across statements: goto, continue, and break.
- Using any of these keywords, except for break in a switch construct, violates the one entry, one exit principle of structured programming.

Examples

```
int i = 0;

A:
i += 1;
if(i<10)
    goto A;
printf("%d", i);
```

```
for(i = 0; i < 10; i++)
{
    printf("%d ", i);
    if(i == 5)
        break;
}
```

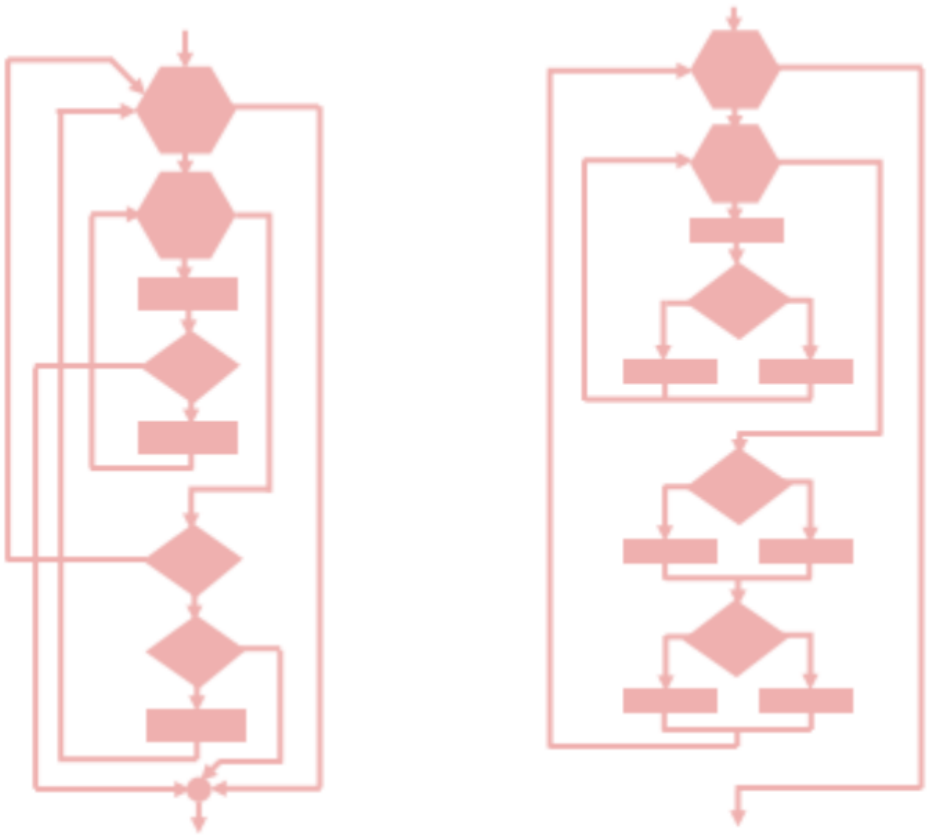
```
for(i = 0; i < 10; i++)
{
    printf("%d ", i);
    if(i == 5)
        continue;
}
```

Flags

- Designing a program with jumps makes the code more difficult to read.
- To improve readability, programmers advocated:
  - the use of whitespace to identify the logical structure of the code
  - the abolition of all goto statements
  - the abolition of all continue statements
  - the abolition of all break statements, except with switch

Flags

- Spaghetti code:



Left: Spaghetti code (various paths cross one another). Right: Clear code.

Flags

A technique for avoiding jumps is called flagging. A flag is a variable that keeps track of a true or false state.

```
total = 0;
for(i = 0; i < 10; i++){
    printf("Enter an integer (0 to stop): ");
    scanf("%d", &value);
    if (value == 0)
        break;  /* Poor programming */
    else
        total += value;
}
printf("The total entered is %d\n", total);
```

```
total = 0;
keepreading = 1;
for(i = 0; i < 10 && keepreading == 1; i++){
    printf("Enter an integer (0 to stop): ");
    scanf("%d", &value);
    if (value == 0)
        keepreading = 0;
    else
        total += value;
}
printf("The total entered is %d\n", total);
```

Dangling Else

- We can embed one logic construct within one another. This is called nesting.
- An ambiguity arises in the case of nested if else constructs.
- Consider the following code:

```
if(windows == 3)
    if(floor == 1)
        printf("Found the room\n");
else
    printf("Try again\n);
```

- In C is that an else always belongs to the innermost if available.
- To associate an else with the next innermost selection, we must wrap the innermost selection in braces.

```
if (windows == 3){
    if(floor ==1)
        printf("Found the room\n");
} else
    printf("Try again\n);
```

Summary

- Logic Constructs
  - Structured Programming
  - Sequence Constructs
  - Selection Constructs
  - In-Class Practice
  - Iteration
  - In-Class Practice
  - Flags
  - Dangling Else

Q&A

Programming Style

- A well-written program is a pleasure to read. Other programmers can understand it without significant effort. The coding style is consistent and clear throughout.
- Develop your own style guide or adopt the style outlined here or elsewhere, but adopt some style.

Naming

- Adopt names that are self-descriptive so that comments clarifying their meaning are unnecessary
- Use names that describe identifiers completely, avoiding cryptic names
- Prefer nouns for variable names
- Keep variable names short - studentName rather than theNameOfAStudent
- Keep the names of indices very short - treat them as mathematical notation

Naming

- Indent the body of any construct that is embedded within another construct.
- Use the same number of spaces for each indentation.

```
for(i = 0; i < n; i++){
      for(j = 0; j < n; j++ ){
          for(k = 0; k < n; k++){
              if (i * j * k != 0)
                  printf(" %4d", i*j*k);
              else
                  printf("    ");
          }
          printf("\n");
      }
      printf("\n");
 }
```

Comments

- Use comments to declare what is done, rather than describe how it is done.
- Comments introduce what follows.
- Keep them brief and avoid decoration.
- Begin every file with a header comment that includes:
  - the title of the program
  - the source file name
  - the name of the author
  - the date last modified
  - other information that you consider important

Comments

Align comments with the code that they describe. Indent both identically.

```
/* display the number 1,...,5  <== GOOD STYLE - ALIGNED */
for(j = 0; j < 5; j++)
    printf(" %4d", j);

    /* display the number 1,...,5  <== POOR STYLE - NOT ALIGNED */
for(j = 0; j < 5; j++)
    printf(" %4d", j);

/* display the number 1,...,5  <== POOR STYLE - NOT ALIGNED */
    for(j = 0; j < 5; j++)
        printf(" %4d", j);
```

Comments

Avoid inline comments, except for name clarifications in variable declarations.

```
int item /* item identifier  <== GOOD STYLE */

/* display the number 1,...,5  <== GOOD STYLE - NOT INLINE */
for(j = 0; j < 5; j++)
    printf(" %4d", j);

for(j = 0; j < 5; j++) /* display the number 1,...,5  <== POOR STYLE - INLINE */
    printf(" %4d", j);
```

Magic Values

- These may be mathematical constants, tax rates, default values or names.
- To improve readability, assign symbolic names to these magic values and refer to the symbolic names throughout the code.
- Use the directive

  #define SYMBOLIC_NAME value
- Example:

  #define PI 3.14159

```
main (){
    double radius:
    printf("Enter radius : ");
    scanf("%lf", &radius);
    printf("The area of your circle is : %lf\n",
           PI * radius * radius);
 }
```

Magic Values

- Compiler Idiosyncracies
  - Use #define to manage idiosyncracies across platforms.
  - For example, the Borland 5.5 compiler does not recognize the long long data type. Instead, it recognizes an _int64 data type. To improve portability, #define the data type using a symbolic name such as LONG_LONG and embed that name throughout our code.

```
#define LONG_LONG long long
#define LL_FORMAT "ll"
...
    LONG_LONG barcode;
    ...
    scanf("%"LL_FORMAT"d", &barcode);
    ...
    printf("%13"LL_FORMAT"d\n", barcode);
```

- In switching to Borland, change the #define values to

```
#define LONG_LONG _int64
#define LL_FORMAT "I64"
```

General Guidelines

- Limit line length to 80 characters - both comments and code
- Avoid global variables
- Select data types for variables wisely and carefully
- Initialize a variable when declaring it only if the initial value is part of the semantic of the variable. If the initial value is part of an algorithm, use a separate assignment statement. For example, instead of

```
int price = units * UNIT_PRICE;
int gst = price * GST;
```

  write

```
int price, gst;
price = units * UNIT_PRICE;
gst = price * GST;
```

General Guidelines

- avoid goto, continue, break except in switch.
- avoid using the character encodings for a particular machine: instead of

```
char newline = 10;
```

  write

```
char newline = '\n';
```

- use a single space either side of an operator

```
i = 0;
```

  or no spaces either side of the operator (but don't mix the two styles)

```
i=0;
```

- use in-line opening braces

```
for (i = 0; i < FINGERS; i++){
    printf("Finger %d\n", i+1 );
}
```

  or start opening braces on a newline (but don't mix the two styles)

```
for (i = 0; i < FINGERS; i++)
{
    printf("Finger %d\n", i+1 );
}
```

General Guidelines

- initialize iteration variables in the context of the iteration. Instead of

```
i = 0;
for( ; i < FINGERS; i++)
{
    printf("Finger %d\n", i+1 );
}
```

  write

```
for(i = 0; i < FINGERS; i++)
{
    printf("Finger %d\n", i+1 );
}
```

General Guidelines

- avoid assignments nested inside logical expressions
- avoid iterations with empty bodies - reserve the body for the algorithm
- limit the initialization and iteration clauses of a for statement to the iteration variables
- distribute and nest complexity
- avoid fancy algorithms that may be efficient but are difficult to read
- add additional comments where code has been fine tuned for efficient execution
- add an extra pair of parentheses where an assignment is also used as a condition
- remove unreferenced variables
- remove all commented code and debugging statements from release and production code
- Refer to some links to published guidelines on the given offline website for more details!

Summary

- Programming Style
  - Naming
  - Indentation
  - Comments
  - Magic Values
  - General Guidelines

Q&A

Walkthroughs

- "Walkthrough" = walk through the program, line by line, to determine exactly what the program does.
- A walkthrough is
  - a record of the changes that occur in the values of program variables as a program executes and
  - a listing of the output, if any, produced by the program.
- Read the Appendix A of Foundations of programming using C - Evan Weaver - page 101 for more details.

Memory Map

- The local variables of a program allocates on the stack segment.

```c
#define ADULT_FARE 2.25

main() {
    int passengers;
    double total;

    printf("Enter the number of passengers : ");
    scanf("%d", &passengers);

    total = passengers * ADULT_FARE;
    printf("passengers is at address %p\n", &passengers);
    printf("total      is at address %p\n", &total);
    printf("total fare is %.2lf\n", total);
}
```

Memory Map

- The specifier %p converts the value of an address to an implementation dependent format, typically, hexadecimal. Each compiler allocates memory differently, for example
  - phobos compiler

Code	Data	Stack	
		passengers 2ff22b60	total 2ff22b68

  - .net compiler

Stack		Code	Data
total 0012FED4	passengers 0012FEE0		

Walkthrough Table

- To prepare a walkthrough of a program, construct an abstract memory map with sufficient room to tabulate the value of each program variable:

-- write program name here --			
data type	data type	...	data type
variable x	variable y	...	variable a
initial value	initial value	...	initial value
next value	next value	...	next value
next value	next value	...	next value
...	...	...	...

Output: write output here (line by line)

Walkthrough Example

- Consider the following program:

```
main()
 {
      int a;
      double b, c;

      a = 6;
      b = 0.7;
      while (a < 10 && b < 3.0) {
          if (a < 8) {
              a = a + 1;
              b = b * 2;
              c = a - b;
          }
          else {
              a = a - 2;
              b = b + 0.8;
          }
          c = a - b;
          printf("%.2lf-%d-%.2lf\n", c, a, b);
      }
 }
```

Walkthrough Example

- Walkthrough table: name of the program, data type, variable name, variable address (optionally)

main()		
int	double	double
a	b	c
6	0.7	
7	1.4	5.6
8	2.8	5.2
6	3.6	2.4

- Output
  5.60-7-1.40
  5.20-8-2.80
  2.40-6-3.60
- Do walkthrough exercises on pages 102-103 of Evan Weaver's notes.

Summary

- Walkthroughs
  - Memory Map
  - Walkthrough Tables
  - Example

Q&A