C LANGUAGE

Reading Code

Reading Code

- Before you write you should learn how to read the code.
- Reading code means executing it by hand.
- There are two important parts in executing code by hand:
  1. Understanding what it does.
  2. Keeping track of the state of the program in a correct fashion.

Reading Code

Variables

- The program keeps track of its state by means of variables.
- A variable is a box that stores the values.
- To use a variable, the programmer must declare it, specify its type and name.

Reading Code

Declaration

int	myVariable	;
variable type	variable name	ends with a semicolon

Variable name (identifier)

- Can be of any length.
- Contains any combination of letters, numbers and underscores.
- Can start only with a letter or underscore (not a number).
- Case sensitive.

Reading Code

Declaration

- When declaring a variable, a box-like space in memory is reserved and labeled with the variable name.
- Its initial value is not known.
- *int x;* 

- *int y;* 

Reading Code

Assignment

myVariable	=	5	;
lvalue	assignment operator	rvalue/expression	end

Assignment

The rvalue expression (a combination of values and operations) is evaluated to a value. The value is assigned to the variable on the left (lvalue).

Reading Code

Initialization

- The declaration and initialization (first assignment) of a variable may be combined into a single statement
- *int x = 10;*. This is equivalent to: *int x; x = 10;*

Reading Code

Expressions

- An expression is a combination of values and operations that evaluates to a value.
- The simplest expression is a numerical constant that evaluates to itself.
- Expression can include mathematical operators (+, -, *, \)
- Mathematical operators have the standard rule of precedence. Division and multiplication occur before subtraction and addition.

Reading Code

Expressions

- Another commonly used operator is % (modulus, or mod). *a % b* gives the remainder of dividing a by b. % has the same precedence of division and multiplication.
- Variables may appear in the expression.
- Some assignment statements are not valid algebraic equation, however it is totally valid in programming languages. $x = x + 1$ is an example of this.

Exercise

Evaluate the following expression:
- 6 + 4 * 5 / 2 - (10 % 3 + 2)

Reading Code

Functions

- *Function* is a self-contained block of code that performs a specific task.
- Functions are a fundamental concept in most programming languages and serve several important purposes such as: Modularity, Reuse, and Abstraction.
- To use a function in your code:
  - *Declaring* a function: how a function behaves.
  - *Calling* a function : executes the function for specific values of the parameters.

Reading Code

Functions

- A function declaration:

return type: $\overbrace{int}$

function's name: $\underbrace{myFunction}$

parameters list: $\overbrace{(int\ x,\ int\ y)}$

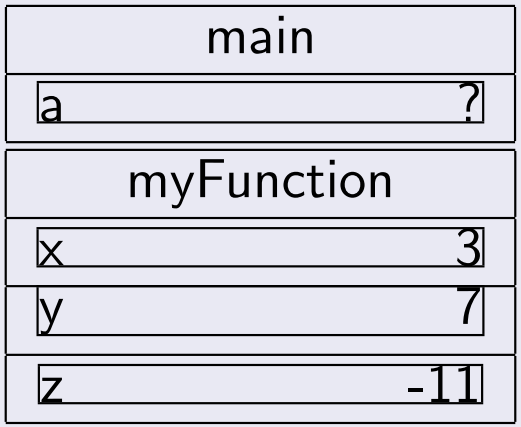
```
{
    int z = x - 2 * y;
    return z * x;
}
```

} function's body

Reading Code

Example

```
1 int myFunction(int x, int y){
2     int z = x - 2*y;
3     return z*x;
4 }
5 int main(void){
6     int a;
7     a = myFunction(3, 7);
8     return 0;
9 }
```

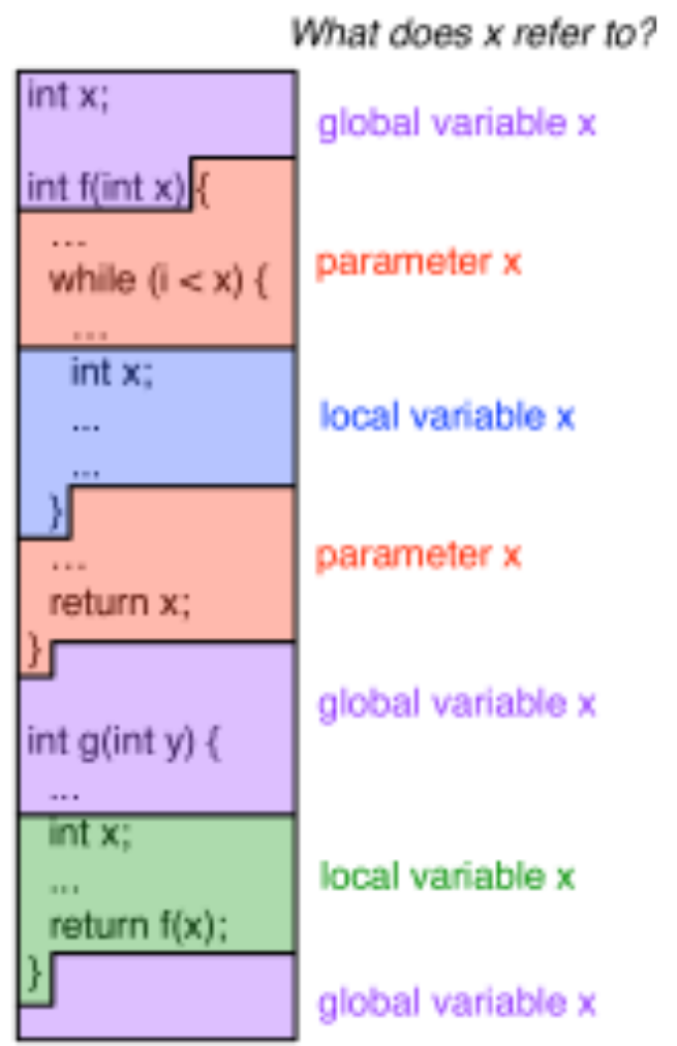


Reading Code

Scope

- The scope of a variable is the region of code in which it is visible.
- Most of the variables are *local variables* (inside a function) and *function parameters*.
- Another scope is the *global variable* (outside any function).

Scope - example



Reading Code

Printing

- To provide output to the terminal we use *printf*.
- *printf* function takes string as parameter.
- To print anything other than strings we use format *specifier*.
- Example:

```
int x = 3;
int y = 4;
printf("x + y = %d",x+y);
```

Reading Code

Printing

- Another addition is the *escape sequences*.
- *Escape sequence* start by a \.
- Example:

```
printf("This is a line.\n This is another line");
```

Reading Code

Printing

```
1  int main(void){
2      int a = 42;
3      int b = 7;
4      printf("Hello world\n");
5      printf("a is %d\n", a);
6      printf("b is %d, a + b is %d\n", b, a+b);
7      return 0;
8  }
```

Reading Code

Conditional statements

- Conditional expressions are used in *if/else* statements.
- When an expression is evaluated to *true* an if branch is executed otherwise else branch is executed.
- *false* in c is 0. *true* is a number other than 0.

Reading Code

Logical operators

1	*expr*1 == *expr*2	tests if *expr*1 is equal to *expr*2
2	*expr*1 != *expr*2	tests if *expr*1 is not equal to *expr*2
3	*expr*1 < *expr*2	tests if *expr*1 is less than *expr*2
4	*expr*1 <= *expr*2	tests if *expr*1 is less than or equal to *expr*2
5	*expr*1 > *expr*2	tests if *expr*1 is greater than *expr*2
6	*expr*1 >= *expr*2	tests if *expr*1 is greater than or equal to *expr*2
7	!*expr*	computes the logical NOT of *expr*
8	*expr*1 && *expr*2	computes the logical AND of *expr*1 and *expr*2
9	*expr*1 || *expr*2	computes the logical OR of *expr*1 and *expr*2

1 - 6 are relational operators and 7 - 8 are boolean operators.

Reading Code

Short circuit

- Short circuit is available for && and || operators.
- The first operand is always evaluated.
- If the value of the operand determines the result the rest of the expression is not evaluated at all.

Reading Code

if / else

```
1 if (x == 3){
2     y = x + 1;
3 }
4 else{
5     z = x - 2;
6     x = x + 1;
7 }
```

Reading Code

if / else

```
1  int f(int x, int y){
2      if (x < y){
3          printf("x < y\n");
4          return x + y;
5      }
6      else{
7          printf("x >= y\n");
8          if(x > 8){
9              return y + 7;
10         }
11     }
12     return x - 2;
13 }
14
15 int main(void){
16     int a = f(3, 4);
17     int b = f(a, 5);
18     return 0;
19 }
```

Reading Code

Conditional / TernaryOperator

Syntax:

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Examples

```
int time = 20;
(time < 18) ? printf("Good day.") : printf("Good evening.");

int grade = 60;
char result;
result= (grade>=50) ? 'P':'F';
printf("%c",result);
```

Reading Code

switch/case

```
1 switch (y - x){
2      case 0:
3           y = 7;
4           break;
5      case 1:
6           y = 9;
12          break;
8      case 2:
9           z = 42;
10          break;
11     default:
12          n = 3;
13}
```

Reading Code

switch/case

```
1  int g(int n, int x){
2      switch (n + x){
3          case 8:
4              x = x + 1;
5          case 0:
6              n = n - 1;
7              break;
8          case 14:
9              return n - x;
10         default:
11             x = n;
12             break;
13     }
14     return x * n;
15 }
16
17 int main(void){
18     int a = g(10, 4);
19     int b = g(a, 2);
20     int c = g(9, b);
21     return 0;
22 }
```

Shorthand

$x \mathrel{+}= y;$	$x = x + y;$
$x \mathrel{-}= y;$	$x = x - y;$
$x \mathrel{*}= y;$	$x = x * y;$
$x \mathrel{/}= y;$	$x = x / y;$
$x{+}{+};$	$x = x + 1;$
$+{+}x;$	$x = x + 1;$
$x{-}{-};$	$x = x - 1;$
$-{-}x;$	$x = x - 1;$

Reading Code

Loops

- Repetition is often needed in a program.
- Finding the repetition is a crucial step in devising an algorithm as a solution for a problem.
- Loops and recursions are methods of repeating a block of code more than one time.

Reading Code

while loop

```
1 while (x < n){
2     y = y * x;
3     x++;
4 }
```

Reading Code

while loop

```
1  int g(int a, int b){
2      int total = 0;
3      while(a < b){
4          total +=a;
5          printf("a = %d, b = %d\n", a, b);
6          a++;
7          b--;
8      }
9      return total;
10 }
11
12 int main(void){
13     int x = g(3,9);
14     printf("x = %d\n", x);
15     return 0;
16 }
```

Reading Code

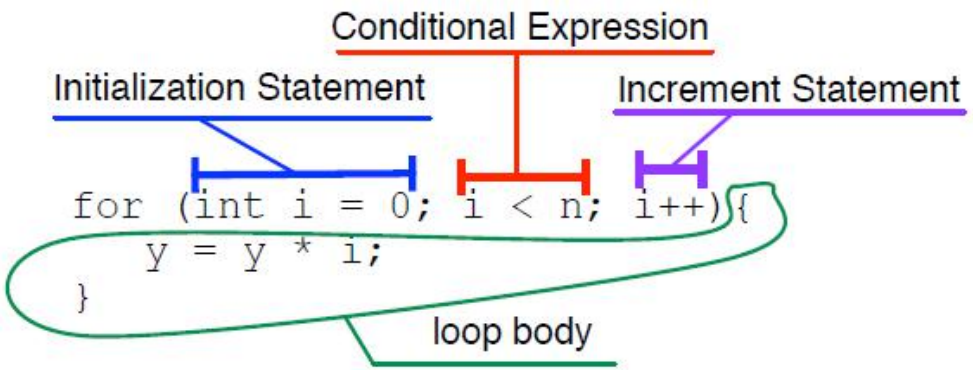
do - while loop

```
1 do {
2      y = y * x;
3      x++;
4 } while(x<n);
```

Reading Code

For loop

Syntax of a for loop



Equivalent while loop

```
{
  int i = 0;
  while (i < n) {
    y = y * i;
    i++;
  }
}
```

Reading Code

Nesting

```
1  void f(int a, int b){
2      while(a<b){
3          printf("a = %d, b = %d\n", a, b);
4          if(a % 2 == 0){
5              for(int i = a; i<b; i++){
6                  printf("i = %d", i);
7              }
8          }
9          b++;
10         a--;
11     }
12 }
13
14 int main(void){
15     f(3, 8);
16     return 0;
17 }
```

Reading Code

continue and break

- *break*: exit the loop completely.
- *continue*: jump back to the top of the loop.

```
1  void printRemainders( int lo, int hi, int n){
2      for(int i = lo; i<hi; i++){
3          if(i == 0){
4              printf("Cannot divide by zero\n");
5              continue;
6          }
7          printf("%d mod %d = %d\n", n, i, n%i);
8      }
9  }
10
11 int main(void){
12     printRemainders(-2, 4, 5);
13     return 0;
14 }
```

Reading Code

Higher-level meaning

- The skill of reading and understanding code is very important.
- You need to understand a code written by others and use it and possibly modifying it.
- Reading the code is a reverse process of writing algorithms. Sometimes you need to workout some examples to understand what it does.
- Sometimes the original programmer was helpful and wrote comments explaining the algorithm.