1806ICT
Programming Fundamentals

Data Types, Operators, Expressions

1

1

Topics

- Structure of a C program
- Data types and variables
- Operators
- Expressions
- Comments

2

2

From Algorithms to Programs

- Both are sets of instructions on how to do a task
- Algorithm:
  - talking to humans, easy to understand
  - in plain (English) language
- Program:
  - talking to computer (compiler)
  - can be regarded as a "formal expression" of an algorithm

3

3

High-Level Language

```
#include <stdio.h>

int main()
{
 printf("Hello World");

 return 0;
}
```

Source code

```
10100110 01110110
00100110 00000000
11111010 11111010
01001110 10100110
11100110 10010110
11001110 00101110
10100110 01001110
11111010 01100110
01001110 10000110
etc...
```

Executable code

- Compilers and linkers translate a high level program into executable machine code.

4

4

Basic Structure of a C Program

Example: Hello World

C Program:

```
#include <stdio.h>

int main()
{
   printf("Hello World!");

   return 0;
}
```

Includes standard input/output library of procedures.

Read: "Hash-include"

5

Basic Structure of a C Program

Example: Hello World

C Program:

```
#include <stdio.h>

int main()
{
   printf("Hello World");

   return 0;
}
```

Curly braces mark the **beginning** and **end** of a block of instructions.

6

Basic Structure of a C Program

Example: Hello World

Instruction (**function call**) to output "Hello World"

C Program:

```
#include <stdio.h>

int main()
{
   printf("Hello World");

   return 0;
}
```

7

Basic Structure of a C Program

Example: Hello World

"Statements" (lines of instructions) always end with a **semi-colon** (;)

C Pr

```
#inc

int main()
{
   printf("Hello World");

   return 0;
}
```

8

Topics

- ✓Structure of a C program
- Data types and variables
- Operators
- Expressions
- Comments

9

9

Variables

Are <u>containers</u> for values – places to store values

Example:

<u>Variable</u>

This jar

can contain

<u>Values</u>

- 10 cookies
- 50 grams of sugar
- 3 slices of cake
- *etc.*

10

10

Variable

- Is a logical name for a container
  - (an actual piece of computer memory for values)
- The data stored by a variable is called its *value*
  - The value is stored in the memory location associated with the variable
- All variables have a *value*, *data type* and a *name* (or identifier)
- A variable has a type associated with it
  - tells the computer how to interpret the bits in the stored value

11

11

Variable

- Syntax

  data_type variable_name

- Must be declared before use

- Examples of variable declarations:

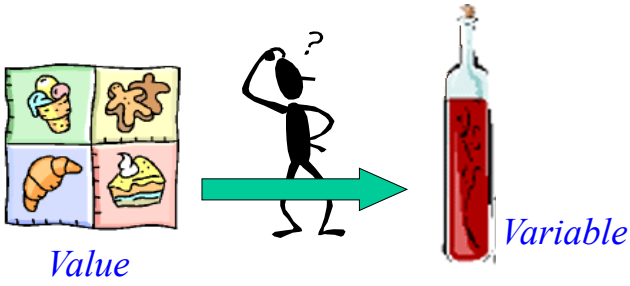
```
int myHeight;
char initial;
float chanceA;
double chanceB;
```

12

12

Data Type

- The kind of a value is its "type"
- Not all values are of the same kind
  - For example: `7 + "cat"` makes no sense



13

13

Data Type

- Built-in types: **`char`**, **`int`**, **`float`**, **`double`**
- Type modifiers: **`long`**, **`short`**, **`const`**
- User-defined types (arrays and structs)
- What about "strings"?
  - Strings are arrays of **`char`**

14

14

Data Type: `int` and `float`

- Integers (`int`)
  `0  1  1000  -1  -10  666`
  Normally stored in 4 bytes of storage
- Floating point numbers – values containing decimal places (`float`)
  `1.0  .1  1.0e-1  1e1`
  Normally stored in 4 bytes of storage

Another type of floating point numbers is `double`, similar to `float`, but with roughly twice the precision. Normally stored in 8 bytes of storage.

15

15

Data Type: `char`

- Characters (`char`)
  `'a' 'z' 'A' 'Z' '?' '@' '0' '9'`
  - Special Characters: preceded by `\`
    `'\n' '\t' '\0' '\'' '\\'` *etc*
- Enclosed within a single pair of quotation marks `''`
- Stored in 1 byte of storage

16

16

Data Type: character string

- Character Strings (a string of **`char`**-s)
- Examples:
  - **`"Hi there!"`**
  - **`"Line 1\nLine 2\nLine 3"`**
  - **`""`**
  - **`"\"\""`**
- Enclosed in double quotes `""`

17

17

Type Modifiers

- long, short – Prefixed on data type to modify (either increase or decrease) the amount of memory storage space allocated to a variable
  - long int mySalary;
  - short int myHeight;
  - long double chanceOfADate;
- unsigned – used to make a variable store only positive values
  - unsigned int myAge;
- signed – used to make a variable store both positive and negative values
  - signed int myBankAccountBalance;

18

18

Variable Declaration: Examples

```
float commission = 0.05;

short int myHeight = 183;

long int mySalary = 100000000000000000000;

long double chanceOfADate = 3e-500;
```

19

Variable Declaration: Examples

```
float commission = 0.05;

short int myHeight = 183;

long int mySalary = 100000000000000000000;

long double chanceOfADate = 3e-500;
```

"Keywords"

20

Keyword

- ...has a special meaning in C
- ...is "case-sensitive"
- ...cannot be used as variable names
- Examples:

```
int, char, long, main, float,
double, const, while, for, if,
else, return, break, case,
switch, default, typedef,
struct, etc.
```

21

21

Variable Declaration: Examples

```
float commission = 0.05;
short int myHeight = 183;
long int mySalary = 10000000000000000000;
long double chanceOfADate = 3e-500;
```

"Identifiers"

22

22

Identifier

- ...is a series of characters consisting of letters, digits and underscores (_)
- ...cannot begin with a digit
- ...must not be a keyword
- ...is "case-sensitive"
- Examples:

  `sUmoFA, x1, y2, _my_ID_, Main` *(careful!)*

23

23

Naming Conventions

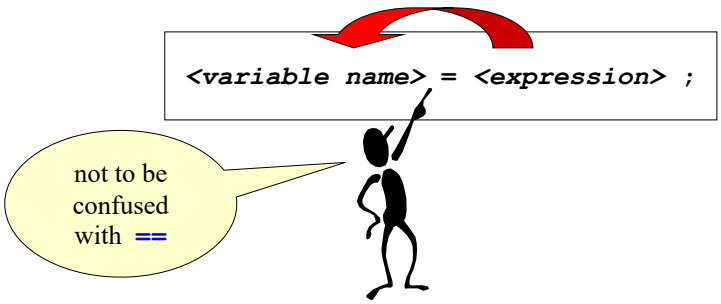
- Variables begin with a lowercase letters (e.g. `answer`, `count`)
- Multiword names are "punctuated" using uppercase letters. (e.g. `myName`, `numberOfCards`)

24

24

Assignment

- Puts a specified value into a specified variable
- Assignment operator: **=**



25

25

Assignment: Examples

```
float x = 2.5;

char  ch;
int   num;
int   val = 3;

ch = '\n';
num = 4 + 5;   /* current value of num is 9 */
num = num * 2; /* current value of num is now 18 */
num = num + val; /* current value of num is now 21 */
```

26

26

Constant Variables

- ...are variables that don't vary
- ...must be initialized when first declared
- ...may not be assigned to

```
const float Pi = 3.14159;
const int classSize = 100;
```

27

27

Constant Variables: Examples

```
const int myID = 192;

myID = 666;   /* Error! */
```

```
const int passMark = 80;

const float pi = 3.1415926;

const double golden_ratio = 1.61803398874989;
```

28

28

Topics

- ✓Structure of a C program
- ✓Data types and variables
- Operators
- Expressions
- Comments

29

Arithmetic Expressions

- Take arithmetic (numerical) values *and* return an arithmetic (numerical) value
- Are composed using the following operators:

```
+ (addition)
- (subtraction)
* (multiplication)
/ (division or quotient)
% (modulus or remainder)
```

Binary operators

```
+ (unary plus)
- (unary minus)
```

Unary operators

30

Arithmetic Expressions

- Arithmetic expressions
  - combination of variables or numbers (known as *operands*) and the arithmetic operators

```
int numStudents = 50;
int numClasses = 3;
int totalNumStudents = numStudents * numClasses
```

operand operand

Arithmetic expression

31

31

Arithmetic Expressions

- When both operands are of the same type, the result is of that type.

```
int numStudents = 50;
numStudents = numStudents + 1;
```

int int int

```
double payRate = 20.5;
double hoursWorked = 2.00;
double salaryPaid = payRate * hoursWorked;
```

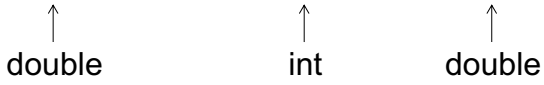
double double double

32

32

Arithmetic Expressions

- When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.
  - `int numApples = 10;`
    `double costApple = 0.50;`
    `double totalAppleCost = numApples * costApple;`



33

33

The Division (/) Operator

- If one of the operands is a floating-point type
  - 9.0 / 2.0 → `4.5`
  - 9 / 2.0 → `4.5`
  - 9.0 / 2 → `4.5`
- When both operands are integer types, the result is truncated, not rounded.
  - 9/2 → `4`
- (10/3 + 1.5)/2 → `2.25`

34

34

The `modulus` (%) Operator

- The `modulus` (%) operator is used with operators of integer type to obtain the remainder after integer division.
- 14 divided by 4 is 3 *with a remainder of 2.*
  - 14 / 4 = 3 with remainder 2
  - Hence, `14 % 4` is equal to `2`
  - `14 = (4 x 3) + 2`
- The modulus operator has many uses, for e.g.
  - determining if an integer is even or odd (if x is an int variable, x is even if x % 2 equals 0, and x is odd if x % 2 equals 1)

35

35

Unary Operators

- Called *unary* because they only require one operand
- Example

```
i = +1;    /* + used as a unary operator */
j = -i;    /* - used as a unary operator */
```

- The unary + operator does nothing – just emphasize that a numeric operand is positive.
- The unary – operator produces the negative of its operand.

36

36

Increment and Decrement Operators

- ++ is the *increment* operator

```
i++;
```

is equivalent to

```
i = i + 1;
```

- −− is the *decrement* operator

```
j--;
```

is equivalent to

```
j = j - 1;
```

37

37

Increment and Decrement Operators in Expressions

- `++count` means increment the value before using it.
- `count++` means increment the value after using it. (use the value, then increment it)

-
```
int m = 4;
int result = 3 * (++m);
```
`result` has a value of `15` and `m` has a value of `5`

-
```
int m = 4;
int result = 3 * (m++);
```
`result` has a value of `12` and `m` has a value of `5`

38

38

Assignment Operators

- Assignment operators can be combined with arithmetic operators (`+`, `-`, `*`, `/`, and `%`).

```
amount = amount + 5;
age = age / 2;
```

can be written as

```
amount += 5;
age /= 2;
```

yielding the same results.

39

39

Precedence in Expressions

- Defines the order in which an expression is evaluated

```
int result = 3 * 7 + 15 % 2 + 6;
```

Highest Precedence

First: the unary operators +, -, !, ++, and --

Second: the binary arithmetic operators *, /, and %

Third: the binary arithmetic operators + and –

Lowest Precedence

40

40

Precedence in Expressions -- Example

```
1 + 2 * 3 - 4 / 5 =
1 + (2 * 3) - (4 / 5)
```

B.O.D.M.A.S.

B stands for brackets,
O for Order (exponents),
D for division,
M for multiplication,
A for addition, and
S for subtraction.

41

41

More on Precedence

- `*`, `/`, `%` are at the same level of precedence
- `+`, `-` are at the same level of precedence
- For operators at the same "level", left-to-right ordering is applied.

```
2 + 3 - 1 = (2 + 3) - 1 = 4
2 - 3 + 1 = (2 - 3) + 1 = 0

6 * 3 / 4 = (6 * 3) / 4 = 18 / 4
6 / 3 * 4 = (6 / 3) * 4 = 2 * 4
```

42

42

Precedence in Expressions – Examples

```
1 + 2 * 3 - 4 / 5
= 1 + (2 * 3) - (4 / 5)
= 1 + 6 – 0 = 7
```

```
1 + 2 * 3 - 4.0 / 5
= 1 + (2 * 3) - (4.0 / 5)
= 1 + 6 - 0.8
= 6.2
```

```
int a = 2;
int b = 1 + ++a * 5;     → 16
```

43

43

Topics

- ✓Structure of a C program
- ✓Data types and variables
- ✓Operators
- ✓Expressions
- Comments

44

44

Comments

- Essential for documenting programs
- Run from a **/*** to the next ***/**
- Examples:

```
/* THIS IS A COMMENT */

/* So is
      this        */

/*
** ...and this.
**
*/
```

45

Comments (cont)

- Comments do not "nest"

```
/*    Comments start with a "/*"
      and end with a "*/"
      but they don't nest!  */
```

46

Common Mistakes

- Forgetting the semi-colon (;)
  - `int x;`
    `int y`
    `int z;`
  - compilation error
- Using undeclared variables, or changing variable names later in the program
  - `int itemCost = 100;`
    `int amountToPay = ItemCost - 10;`
  - compilation error

47

47

Common Mistakes

- Using uninitialized variables
  - `int itemCost;`
    `int amountToPay = itemCost - 10;`
  - obtain result that is different from what is expected
- Using integer division instead of floating point division
  - `float x = 1.0 + 5/2;` ✗
  - `float x = 1.0 + 5.0/2;` ✓
  - obtain result that is different from what is expected

48

48