CS 101: Computer Programming and Utilization

Jul-Nov 2017

Umesh Bellur
(cs101@cse.iitb.ac.in)

Lecture 4: Variables, Data Types, and Expressions
Recall

• In the previous slide set, we learnt that computers essentially do arithmetic operations on numbers stored in the memory

• Now we will learn details of how different types of numbers are represented and stored, and referred to in a program
Outline

• How to store numbers in the memory of a computer
• How to perform arithmetic
• How to read numbers into the memory from the keyboard
• How to print numbers on the screen
• Many programs based on all this
Reserving Memory For Storing Numbers

Before you store numbers in the computer's memory, you must explicitly reserve space for storing them in the memory. This is done by a variable declaration statement.

variable: name given to the space you reserved.
You must also state what kind of values will be stored in the variable: data type of the variable.

Byte#5 reserved for some variable named, "c", say.
Variable Declaration

A general statement of the form:

\[ data\_type\_name \ variable\_name; \]

Creates and declares variables

Earlier example

\[ \text{int sides;} \]

\text{int} : name of the data type. Short form for integer. Says

reserve space for storing integer values, positive or negative, of
a standard size

Standard size = 32 bits on most computers

sides : name given to the reserved space, or the variable created
int sides;

Results in a memory location of size 32 bits being reserved for this variable. The program will refer to it by the name sides.
Variable Names: **Identifiers**

Sequence of **one** or more letters, digits and the underscore “_” character

- Should not begin with a digit
- Some words such as `int` cannot be used as variable names. **Reserved** by C++ for its own use
- **Case matters.** ABC and abc are distinct identifiers

Examples:

- **Valid identifiers:** sides, telephone_number, x, x123, third_cousin
- **Invalid identifiers:** #sides, 3rd_cousin, third cousin

Recommendation: use meaningful names, describing the purpose for which the variable will be used
Some Other Data Types Of C++

- **unsigned int**: Used for storing integers which will always be positive
  - 1 word (32 bits) will be allocated
  - Ordinary binary representation will be used
- **char**: Used for storing characters or small integers
  - 1 byte will be allocated
  - ASCII code of characters is stored
- **float**: Used for storing real numbers
  - 1 word will be allocated
  - IEEE FP representation, 8 bits exponent, 24 bits significand
- **double**: Used for storing real numbers
  - 2 words will be allocated
  - IEEE FP representation, 11 bits exponent, 53 bits significand
Variable Declarations

- Okay to define several variables in same statement
- The keyword `long` says, I need to store bigger or more precise numbers, so give me more than usual space.
- `long unsigned int`: Likely 64 bits will be allocated
- `long double`: Likely 96 bits will be allocated

```c
unsigned int telephone_number;
float velocity;
float mass, acceleration;
long unsigned int crypto_password;
long double more_precise_value;
```
Variable Initialization

- **Initialization** – an INITIAL value is assigned to the variable

  the value stored in the variable at the time of its creation

- Variables $i$, $vx$, $vy$ are declared and are initialized

- $2.0e5$ is how we write $2.0 \times 10^5$

- `'f'` is a **character constant** representing the ASCII value of the quoted character

- $result$ and $weight$ are declared but not initialized

```c
int i=0, result;
float vx=1.0, vy=2.0e5, weight;
char value = ‘f’;
```
Const Keyword

const double pi = 3.14;

The keyword const means: value assigned once cannot be changed

Useful in readability of a program

    area = pi * radius * radius;

reads better than

    area = 3.14 * radius * radius;
Reading Values Into Variables (1)

- Can read into several variables one after another

- If you read into a char type variable, the ASCII code of the typed character gets stored

- If you type the character ‘f’, the ASCII value of ‘f’ will get stored

```cpp
    cin >> noofsides;
    cin >> vx >> vy;
    char command;
    cin >> command;
```
Some rules:
- User expected to type in values consistent with the type of
  the variable into which it is to be read
- *Whitespaces* (i.e. space characters, tabs, newlines)
  typed by the user are ignored.
- newline/enter key must be pressed after values are typed
Printing Variables On The Screen

- General form: `cout << variable;`
- Many values can be printed one after another
- To print newline, use `endl`
- Additional text can be printed by enclosing it in quotes
- This one prints the text `Position:`, then `x` and `y` with a comma between them and a newline after them
- If you print a `char` variable, then the content is interpreted as an ASCII code, and the corresponding character is printed.
  - `G` will be printed.

```cpp
cout << x;

cout << x << y;

cout << "Position:" << x << "", " << y << endl;

char var = 'G';

cout << var;
```
An Assignment Statement

Used to store results of computation into a variable. Form: \texttt{variable\_name = expression;}

Example:

\texttt{s = u*t + 0.5 * a * t * t;}

Expression: can specify a formula involving constants or variables, almost as in mathematics

- If variables are specified, their values are used.
- Operators must be written explicitly
- Multiplication, division have higher precedence than addition, subtraction
- Multiplication, division have same precedence
- Addition, subtraction have same precedence
- Operators of same precedence will be evaluated left to right.
- Parentheses can be used with usual meaning
Examples

int x=2, y=3, p=4, q=5, r, s, t;
x = r*s;                       // disaster.  r, s undefined
r = x*y + p*q;                // r becomes 2*3 + 4*5 = 26
s = x*(y+p)*q;               // s becomes 2*(3+4)*5 = 70
t = x − y + p − q;            // equal precedence,
                             // so evaluated left to right,
                             // t becomes ((((2−3)+4)-5 = -2
Arithmetic Between Different Types Allowed

int x=2, y=3, z, w;
float q=3.1, r, s;
r = x; // representation changed
    // 2 stored as a float in r "2.0"
z = q; // store with truncation
    // z takes integer value 3
s = x*q; // convert to same type,
    // then multiply
    // Which type?
Evaluating $\text{varA op varB}$
e.g. $x*q$

- if $\text{varA, varB}$ have the same data type: the result will have same data type
- if $\text{varA, varB}$ have different data types: the result will have more expressive data type
- int/short/unsigned int are less expressive than float/double
- shorter types are less expressive than longer types
Rules for storing numbers of one type into variable of another type

- C++ does the “best possible”.

```c
int x; float y;
x = 2.5;
y = 123456789;
```

- `x` will become 2, since it can hold only integers. Fractional part is dropped.
- `123456789` cannot be precisely represented in 24 bits, so something like \(1.234567 \times 10^8\) will get stored.
Integer Division

```cpp
int x=2, y=3, p=4, q=5, u;
u = x/y + p/q;
cout << p/y;
```

- **x/y**: both are `int`. So truncation. Hence 0
- **p/q**: similarly 0
- **p/y**: 4/3 after truncation will be 1
- So the output is 1
More Examples of Division

```c
int noosides=100, i_angle1, i_angle2;
i_angle1 = 360/noosides + 0.45;        // 3
i_angle2 = 360.0/noosides + 0.45;      // 4

float f_angle1, f_angle2;
f_angle1 = 360/noosides + 0.1;        // 3.1
f_angle2 = 360.0/noosides + 0.1       // 3.7
```
An Example Limited Precision

float w, y=1.5, avogadro=6.022e23;
w = y + avogadro;

• Actual sum : 602200000000000000000000001.5
• y + avogadro will have type float, i.e. about 7 digits of precision.
• With 7 digits of precision ($2^{23}$), all digits after the 7\textsuperscript{th} will get truncated and the value of avogadro will be the same as the value of y + avogadro
• w will be equal to avogadro
• No effect of addition!
Program Example

```c
main_program{
    double centigrade, fahrenheit;
    cout << "Give temperature in Centigrade: ";
    cin >> centigrade;
    fahrenheit = centigrade * 9 / 5 + 32;
    cout << "In Fahrenheit: " << fahrenheit
       << endl;  // newline
}
```

Prompting for input is meaningless in Prutor because it is non-interactive
Re-Assignment

- Same variable can be assigned a value again
- When a variable appears in a statement, its value at the time of the execution of the statement gets used

```cpp
int p=3, q=4, r;
r = p + q; // 7 stored into r
cout << r << endl; // 7 printed as the value of r
r = p * q; // 12 stored into r (could be its
            // temporary location)
cout << r << endl; // 12 printed as the value of r
```
In C++ "=" is assignment not "equal"

```c++
int p=12;
p = p+1;
```

See it as:
```c++
p ← p+1;  // Let p become p+1
```

Rule for evaluation:

- FIRST evaluate the RHS and THEN store the result into the LHS variable
- So 1 is added to 12, the value of p
- The result, 13, is then stored in p
- Thus p finally becomes 13

```
p = p + 1 is nonsensical in mathematics
"=" in C++ is different from "=" in mathematics
```
Repeat And Reassignment

main_program{
    int i=1;
    repeat(10){
        cout << i << endl;
        i = i + 1;
    }
}

This program will print 1, 2,…, 10 on separate lines
Another Idiom: Accumulation

```cpp
main_program{
    int term, s = 0;
    repeat(10){
        cin >> term;
        s = s + term;
    }
    cout << s << endl;
}
```

- Values read are accumulated into `s`
- Accumulation happens here using `+`
- We could use other operators too
Fundamental idiom

Sequence generation

• Can you make $i$ take values 1, 3, 5, 7, …?
• Can you make $i$ take values 1, 2, 4, 8, 16, …?
• Both can be done by making slight modifications to previous program.
Composing The Two Idioms

Write a program to calculate n! given n.

```c++
main_program{
    int n, nfac=1, i=1;
    cin >> n;
    repeat(n){
        nfac = nfac * i;
        i = i + 1;
    }
    cout << nfac << endl;
}
```

Accumulation idiom
Sequence idiom
Finding Remainder

- $x \% y$ computes the remainder of dividing $x$ by $y$
- Both $x$ and $y$ must be integer expressions
- Example

```c
int n=12345678, d0, d1;
d0 = n % 10; // 8
d1 = (n / 10) % 10; // 7
```

d0 will equal 8 (the least significant digit of $n$)
d1 will equal 7 (the second least significant digit of $n$)
Some Additional Operators

• The fragment \( i = i + 1 \) is required very frequently, and so can be abbreviated as \( i++ \)
  
  \[ \text{++ : increment operator. Unary} \]

• Similarly we may write \( j-- \) which means \( j = j - 1 \)
  
  \[ \text{-- : decrement operator. Unary} \]
Intricacies Of ++ and --

++ and -- can be written after or before the variable. Both cause the variable to increment or decrement but with subtle differences

```cpp
int i=5, j=5, r, s;
r = ++i;
s = j++;
cout << "r= " << r << " s= " << s;
```

i, j both become 6 but r is 6 and s is 5.

++ and -- can be put inside expressions but not recommended in good programming
Compound Assignment

The fragments of the form `sum = sum + expression` occur frequently, and hence they can be shortened to `sum += expression`.

Likewise you may have `*=` , `-=`, `...`

Example

```c
int x=5, y=6, z=7, w=8;
x += z; // x becomes x+z = 12
y *= z+w; // y becomes y*(z+w) = 90
```
Blocks and Scope

• Code inside {} is called a block.
• Blocks are associated with repeats, but you may create them otherwise too.
• You may declare variables inside any block.
New summing program:
• The variable term is defined close to where it is used, rather than at the beginning. This makes the program more readable.
• But the execution of this code is a bit involved.

```
// The summing program
// written differently.

main_program{
    int s = 0;
    repeat(10){
        int term;
        cin >> term;
        s = s + term;
    }
    cout << s << term << endl;
}
```
How definitions in a block execute

Basic rules
• A variable is defined/created every time control reaches the definition.
• All variables defined in a block are destroyed every time control reaches the end of the block.
• “Creating” a variable is only notional; the compiler simply starts using that region of memory from then on.
• Likewise “destroying” a variable is notional.
• New summing program executes exactly like the old, it just reads different (better!).
Shadowing and scope

• Variables defined outside a block can be used inside the block, if no variable of the same name is defined inside the block.
• If a variable of the same name is defined, then from the point of definition to the end of the block, the newly defined variable gets used.
• The new variable is said to “shadow” the old variable.
• The region of the program where a variable defined in a particular definition can be used is said to be the scope of the definition.
Example

main_program{
  int x=5;
  cout << x << endl; // prints 5
  repeat (3) {
    cout << x << endl; // prints 5
    int x = 10;
    x *= 2;
    cout << x << endl; // prints 20
  }
  cout << x << endl; // prints 5
}
Concluding Remarks

Variables are regions of memory which can store values.
Variables have a type, as decided at the time of creation.
Choose variable names to fit the purpose for which the variable is defined.
The name of the variable may refer to the region of memory (if the name appears on the left hand side of an assignment), or its value (if the name appears on the right hand side of an assignment).
Expressions in C++ are similar to those in mathematics, except that values may get converted from integer to real or vice versa and truncation might happen.

Truncation may also happen when values get stored into a variable.

Sequence generation and accumulation are very common idioms.

Increment/decrement operators and compound assignment operators also are commonly used (they are not found in mathematics).
More Remarks

Variables can be defined inside any block.

Variables defined outside a block may get shadowed by variables defined inside.
SAFE quiz

• What is the result of evaluating the expression \((3+2)/4\)?

• What is printed by this code snippet: "float f=6.022E23; float r=f+2-f; cout<<r;"? 

• What is printed by this code snippet: "int t=10; repeat(2){t=t-1.2;} cout<<t;"? 

• What is printed by this code: "int i=2, j=3, k=4; i=j; j=k; k=i; cout << (i*j*k)"?