Introduction to Computing

Lectured by: Dr. Pham Tran Vu

t.v.pham@cse.hcmut.edu.vn

Programming

- **Programming languages**
- Program design, testing, debugging and documenting
- Data structures

Programming Languages

- Machine language
- Assembly languages
- High-level programming languages
- Language processing

Machine Languages

- Machine languages are the languages that can be understood directly by computer processors
- Code written using machine language (machine code) can be executed directly by a computer's processor
- Also known as native code
- Each CPU model usually has its own machine language or machine code instruction set

Machine Language (2)

- Each machine code instruction performs a very basic operation such as arithmetic calculations or disk read/write operations
- Each machine code instruction commonly has two basic parts: opcode and operand, which are expressed in binary
- It is difficult to remember and use machine language directly to solve real world problems

Machine Language Example

- The following example code is written using Intel 80x86 machine code
- It performs two operations: i=5 and j=i+10

Binary	Hex
10111000 00000101 00000000	b8 05 00
10100011 00000000 00000002	a3 00 02
10100001 00000000 00000002	a1 00 02
00000101 00001010 00000000	05 0a 00
10100011 00000010 00000010	a3 02 02

Assembly Languages

- Assembly languages are low-level programming languages
- They are more readable than machine languages
- An assembly language uses a symbolic representation of numeric machine codes and constants
- Example: add, mov, sub, etc
- Assembly code is translated to machine code by a utility program called assembler

Assembly Language Example

Machine language		Assembly
10111000 00000101 00000000	b8 05 00	mov ax, 5
10100011 00000000 00000002	a3 00 02	mov [200], ax
10100001 00000000 00000002	a1 00 02	mov ax, [200]
00000101 00001010 00000000	05 0a 00	add ax, 10
10100011 00000010 00000010	a3 02 02	mov [202],ax

High-Level Programming Languages

- A high-level language provides a high level abstraction of computer programs
- It is more natural to human languages
- It allows programmers to use many more data types and complex data structures
- High-level languages are independent of computer hardware
- Examples: Pascal, C/C++, Java, etc

High-Level Language Example

- A piece of C code

```
short i, j;        // define two variables i and j
i = 5;             // assign 5 to i
j = i +10;         // calculate i+10 and store the result in j
```

Generations of Programming Languages (1)

- First generation
  - Machine languages
  - Appeared in the 1960s
- Second generation
  - Low-level languages, e.g. assembly languages
- Third generation
  - High-level languages, e.g. C/C++, Pascal, Java

Generations of Programming Languages (2)

- Fourth generation
  - Easier to use than high level languages
  - Quick solutions to data processing task
  - Closer to natural languages
  - Non-procedural
  - E.g. Structured Query Languages
- Fifth generation
  - More declarative
  - E.g. PROLOG, LISP and Smalltalk

Components of Computer Programs

- Keywords
  - Reserved words used by programming languages
- Identifiers
  - Names created by programmers given to variables or constants
- Scope of variables
  - The degree of accessibility (validity) of a variable
  - Global vs local scope

Components of Computer Programs (2)

- Data structures
  - Define the data types in a program
  - E.g.: numeric, character, boolean, pointer, arrays, record, file, etc.
- Operations on data
  - Arithmetic operations: addition, subtraction, etc
  - Logic operations: and, or, xor, nand, etc
- Input and output

Components of Computer Programs (3)

- Control structures
  - Selections: if … then … else
  - Iterations: for, while
- File handling
  - Open files
  - Close files
  - Read, write, delete
- Functions and procedures
  - Subprograms

Components of Computer Programs (4)

- Blocking structures
  - Groups of statements
- Parameters
  - Inputs to a function/procedure
  - Call by value
  - Call by reference

A Sample Program (1)

```
#include<stdio.h>
int cnt = 0;
void printRes(int [], int);
void findPer(int [], int, int);
void reOrder(int [], int , int, int);
void arrayCopy(int [], int [], int);
int main(){
   int ars[] = {1, 2, 3, 4};
   printf("test %d: \n", 4);
   findPer(ars, 0, 4);
}
```

A Sample Program (2)

```
void arrayCopy(int ars1 [], int ars2[],
    int size){
    int i;
    for (i =0; i < size; i++){
        ars2[i] = ars1[i];
    }
}
```

A Sample Program (3)

```
void reOrder(int ars[], int pick, int start, int size){
    int temp;
    int i;
    if (pick == start){
        return;
    }
    if (pick < start || pick >= size || pick < 0){
        printf("Error, pick cannot be smaller than start\n");
        return;
    }
    temp = ars[pick];
    for (i = pick; i > start; i--){
        ars[i] = ars[i-1];
    }
    ars[start] = temp;
}
```

A Sample Program (5)

```
void printRes(int ars[], int size){
   int i;
   printf("Cnt : %d \n", ++cnt);
   for (i =0; i< size; i++){
      printf("%d ", ars[i]);
   }
   printf("\n");
}
```

A Sample Program (4)

```
void findPer(int ars[], int start, int size){
    int i;
    int * temp = (int*)malloc(size*sizeof(int));
    if (start == size - 1){
        printRes(ars, size);
        return;
    }
    arrayCopy(ars, temp, size);
    for (i = start; i < size; i++){
        reOrder(ars, i, start, size);
        findPer(ars, start + 1, size);
        arrayCopy(temp, ars, size);
    }
    free(temp);
}
```

Recursive Programming

- A recursion happens when a function/procedure calls its self
- Example:

```
void findPer(int ars[], int start, int size){
    ...
    for (i = start; i < size; i++){
        reOrder(ars, i, start, size);
        findPer(ars, start + 1, size);
        arrayCopy(temp, ars, size);
    }
    free(temp);
}
```

Language Processing

- Programs written in high-level languages need to be converted to machine code for execution
- A program written in a particular language needs to be processed accordingly
- How do we ensure that a program is written correctly following a programming language?
- How to define a language?

Computer Languages

- Every programming language has a set of rules to govern the syntax of well-formed statements and sentences
- This set of rules is called the grammar of the languages
- Each different language needs a different way to process its programs according to its grammar

Language Syntax

- The syntax of a language describes possible combination of symbols that forms a syntactically correct program
- Syntax is usually defined using a combination of regular expressions and Backus-Naur form
- Example:

```
expression ::= atom | list
atom  ::= number | symbol
number  ::= [+-]?['0'-'9']+
symbol  ::= ['A'-'Z''a'-'z'].*
list  ::= '(' expression* ') '
```

Compilers and Interpreters

- There are two ways to translate a program written in high-level languages into machine code:
  - Using a compiler
  - Using a interpreter

Compilers

- A compiler accept a source program written in a high-level language and translate it into an object program in a low-level language
- The object program can be in assembly code, machine code or byte code (to be executed by virtual machines)
- During compilation, a compiler often needs to access to a run-time library

Steps in a Compilation Process

- Lexical analysis
  - the source code is converted to a form which is more convenient for subsequent processing
- Syntax analysis and semantic analysis
  - Check for grammatical correctness (done by a parser)
- Intermediate code generation
- Code optimisation
- Code generation

Interpreters

- □ Object programs are not generated in this form of translation
- □ Source code statements are translated and executed separately, once after another
- □ Every time a program is run, the interpreter has to read and translate the source code again