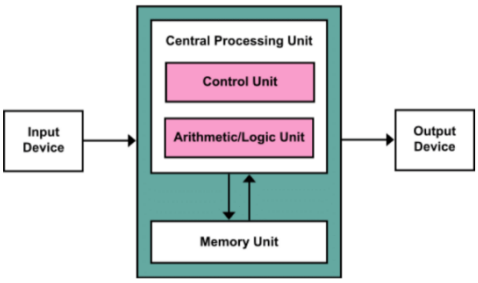
GNG 1106 | 2019/2020 | Yongi Mao

Engineering Computation Study Sheet

Computer: A device that carries out *programs* using arithmetic or logical expressions. This requires *hardware* and *software.*

Figure 1: CPU



- Input and output devices
- Memory
- CPU
- Secondary storage (long term)

8 bits = 1 byte

Machine language: Only the computer can understand, and all programs and data must be translated into *machine language* before the computer can understand it.

Assembler: Translates program into machine language

Compiler: Translates *high level program* into machine language

Build = preprocess + compile + link

- **Preprocess:** Executes preprocessor directives
- **Compile:** Checks *syntax* and translates code into *object code*
- **Link:** Links code with necessary library (e.g. <stdio.h>, <math.h>, etc.

Run = load + execute

- **Load:** Loads executable code into the memory
- **Execute:** Computer performs operation based on the code

IDE (integrated development environment): A software has all tools previously mentioned (e.g. CodeBlocks)

Pseudo code: English statements that *describe* the desired operation of the computer.

Variables:

- A variable stores a value
- A variable is of a unique type (e.g. *int, double, float, char,* etc.)
    - Type implies:
        - How much storage will be allocated
        - How it is encoded into bytes
- A variable will have an *address*

Byte = house && *memory* = street

Functions:

- Functions are used to *modularize* a program
- A function is a sub-program
- A function has 3 purposes:
  - Take input
  - Operate
  - Output
- A function with a non-void return type must include a *return statement* (e.g. return _____;)
- Calling a function examples:
  - hello();
  - prtmax(7, 9-6);
  - x=max(5, 9);

Figure 2: Function Declaration

```
aType functionName(parameterList)
{
        declarations;
        commands;
}
```

- `parameterList` contains the declaration of one or a list of input variables that will receive input data
  - Individual declarations are separated by "," (commas).
  - the list can be empty (or made as `void`) if the function takes no input
- `functionName` is the name of the function
- `aType` is the type of the value returned by the function.
- If `aType` is `void`, it means that the function returns nothing
- Variable declaration and commands are inside "{}"
- There is no ";" after }

Implicit rule of promotion:

- Float type values can be represented as a double type value without loosing information
- E.g:
  - float → double (works)
  - double → float (doesn't)
- Low byte x High byte = High byte

Figure 3: Conversion Specifiers

Type	printf	scanf
short	%d, %i, %hd, %hi	%hd, %hi
int	%d, %i	%d, %i
long	%ld, %li	%ld, %li
unsigned short	%hu	%hu
unsigned int	%u	%u
unsigned long	%lu	%lu
float	%f, %e, %E, %g, %G	%f, %e, %E, %g, %G
double	%f, %e, %E, %g, %G	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG	%Lf, %Le, %LE, %Lg, %LG

Logical operators: "AND," "OR," and "NOT" are logical operators.

Figure 4: Logical Operators

expression	C Operator	C expression
$x > y$	`>`	`x > y`
$x < y$	`<`	`x < y`
$x \geq y$	`>=`	`x >= y`
$x \leq y$	`<=`	`x <= y`
$x = y$	`==`	`x == y`
$x \neq y$	`!=`	`x!= y`
AND	`&&`	`x && y`
OR	`||`	`x || y`
NOT	`!`	`!x`

The operator ! is a unary operator.

Function prototype:

- type functionName (type1, type2, …);
- type functionName (type1 var1, type2 var2, …);

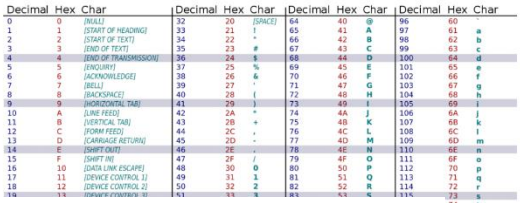
Cast operator: can be used to apply a type to a variable (e.g. (aType) aValue). Cast *low-precision to high-precision* variables.

Char type:

- Used to store characters
- The most common way to code characters into integers is through *ASCII code*

```
myChar = 'a';
myChar = 97;
```

Figure 5: ASCII Chart



Decision Structures:

- *If* statement
- *If else* statement
- *Switch* statement

Figure 7: If Statement with {}

```
if (S)
{
    Statement1;
    Statement2;
    ...
}
```

Figure 6: Nested else if

```
if (grade >= 90)
  printf("You have an A+\n");
else if (grade >= 85)
  printf("You have an A\n");
else if (grade >= 80)
  printf("You have an A-\n");
else if (grade >= 70)
  printf("You have a B\n");
...
else
  printf("You failed\n");
```

Figure 8: Switch Statement

```
switch (I)
{
   case CONST1:
      DoA;
      break;
   case CONST2:
      DoB;
      break;
   ...
   default:
      DoX;
}
```

- `switch`, `case`, `default`, and `break` are all keywords.
- `I` is an expression that evaluates to an integer.
- `CONST1`, `CONST2` ... are integer literals.
- `DoA`, `DoB`, ... `DoX` are each a line of code or a block of code.
  - If any of them is a block of code, the block needs not to be enclosed by curly brackets `{ }`.
- Arbitrarily many `case`'s are allowed.

Loops:

- *For loop*
- *While loop*
  - *Sentinel* is used to control the loop if it was to operate for an undefined # of loops
- *Do while loop*

Figure 9: For Loop

```
for (initializeCounter; S; modifyCounter)
  DoX;
```

Figure 10: While loop

```
while (S)
    DoX;
```

Figure 11: Do While Loop

```
do
    DoX;
while(S);
```

Array:

- An *array* contains same type variables
- `Type arrayName[arrayLength];`
- `float x[5]={0.0, 0.5, 1.0, 1.5, 2.0};`
- Arrays can also be in *matrix* form e.g. `type arrayName[numOfRows][numOfColumns];`

Structures:

- Store many *different type* variables

Figure 12: Furthest Simplification of Structure Declaration

- Serves as new type: “struct structureName”
- To access a structures member:
    - Use: *structureName.memberName*
- typedef can rename a type and makes structures simpler

Figure 13: Unary Operators '&' and ''*

Highlight (“&” and “*” as a Reciprocal Pair of Operators)

- `&X` gives the address for variable `X`.
- `*X` gives the “variable” having address `X`.

Highlight (A New View of Variable)

A variable is nothing more than a block of bytes in the memory, which has been associated with a prescribed method specifying how it should be interpreted as a number or as numbers.

using *

Figure 14: Function Using Array Pointers to Receive Array Input

```
returnType myFunc(someType* ptrArray, int lengthArray)
{
    ...
}
```

```
typedef struct
{
 int ID;
 char name[200];
 float grade;
} REC;
```

Pointers:

- Pointers are used as variables that store addresses
- Declared using:
    - varType * pointerName = &var
- *NULL* pointers point nowhere and *NULL* == 0 by default
- A pointer can be dereferenced