The C language

Introductory course #1

History of C

- Born at AT&T Bell Laboratory of USA in 1972. Written by Dennis Ritchie
- C language was created for designing the UNIX operating system
- Quickly adopted as a mainstream language because of its powerful features

Main Features of C

- (Reasonably) High level as well as (very) low level programming support
    - User defined data structure, modular programming
    - bit manipulation, efficient pointer manipulation, close to assembly language but much more comfortable
- Program portability

Applications of C

- Computer application implementation
- Embedded software
- Drivers and firmware programming
- Complex tools implementation such as verification and validation tools, interpreters, compilers…
- Operating systems and OS extensions

Outline

- History, main features and applications of C
- “Hello” example coding, compiling and running
- Literals, constant, variables and types
- Basic input/output
- Expressions and statements

Let’s go

```
#include <stdlib.h>
#include <stdio.h>

int main(){
   printf("Hello"); // comment
   return EXIT_SUCCESS;
}
```

Compilation:
from the code to the executable

```
# edition of the code
> vi hello.c

# compilation with the GNU compiler
> gcc hello.c

# running of the executable
> ./a.out

# as a result, « Hello » output
Hello>
```

Compilation:
from the code to the executable

```
# specifying the output name
> gcc hello.c -o hello

# running of the executable
> ./hello
```

Literals

- Integers: 123
- Characters: 'a', 'B'
  - Special characters: '\n', '\t'
  - Strings: "ab \t123", "Hello\n"
- Floats: 1.23, 0.123, 1e+2
- Arrays: { 1, 2, 3 }

Constants

- Two ways: with the **const** keyword or with a **#define** directive

```
#define AGE
int main {
  const int age = 45;
  printf("Hello, I’m %d years old", AGE);
  printf("Hello, I’m %d years old", age);
  return EXIT_SUCCESS;
}
```

Variables

- A variable is a container that stores a data
- Must be declared before being used
  - Declaration: a type followed by the variable name
    - the type specifies the kind of data that can be stored

```
int main {
  int age; // declaration
  age = 24; // storage of the integer 24
  printf("Hello, I'm %d years old", age);
  return EXIT_SUCCESS;
}
```

Types

- A type is an information used by the compiler to know the size of the container and to check that data are valid (type compatibility checked at compile time)
- Type informations are absent at run time

```
int main () {
        int age;
        age = "24";
        printf("Hello, I’m %d years old", age);
        return EXIT_SUCCESS;
}
```

```
> gcc assign_error.c
assign_error.c:5:6: warning: incompatible pointer to integer
conversion
      assigning to 'int' from 'char [3]' [-Wint-conversion]
        age = "24";
            ^ ~~~~
1 warning generated

> ./a.out
Hello, I’m 166592382 years old
```

Main builtin types

- **int, long int** : for integers
- **char** : for characters
- **float, double, long double** : for real numbers

Remarks:

✓ **No String type** : a string is an array of chars

✓ **No boolean** : 0 for false, whatever value except 0 for true

✓ A char is handled as an integer

✓ More builtin types exist

```c
int main () {
  int age;
  age = 24;

  char gender = ‘M’;
  float rate = 4.56754;

  int succes = EXIT_SUCCESS;

  char * name = "Paulette";

  printf("Name: %s\n", name);
  printf("Age: %d\n", age);
  printf("Gender: %c\n", gender);
  printf("Rate: %f\n", rate);

  return success;
}
```

```
> gcc name_age_gender.c
> ./a.out
Name: Paulette
Age: 24
Gender: M
Rate: 4.567540
>
```

Input/Output
very basic approach

- value printing and input through 2 functions:
  - printf and scanf:
  - at least one string argument which is a template for the output or the input

printf

- One string argument, eventually followed by any number of arguments

```
int main () {
  char * name = "Paulette";
  int age = 67;
  printf("Hello\n");
  printf("\tI'm %s and I'm %d years old\n",
    name, age);
…
```

- %d for int, %c for char, %s for char *, %f for float, %d for double…
- char * is for a pointer on character, we will see that later

scanf

- use a & followed by a variable name (need to pass variable address, we will explain that point later in deeper detail)

```
int main () {
  char gender;
  int age;
  scanf("%d", &age);
  scanf("%c", &gender);
  printf("I'm %d years old [%c]\n",
    age,
    gender);
…
```

scanf

- It is also possible to read a string
- Useful for simple applications

```
#include <stdio.h>
int main( )
{
  char str[100];
  printf( "Enter a string followed by a return:");
  scanf("%s", str);
  printf( "You entered: [%s]\n", str);
  return 0;
}
```

Operators

- Expressions are specified by using builtin operators
- 6 kind of operators:
    - Assignment Operators
    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
    - Bitwise Operators
    - Misc Operators

Assignment

= Simple assignment operator
+= Add AND assignment operator
-= Subtract AND assignment operator
*= Multiply AND assignment operator
/= Divide AND assignment operator
%= Modulus AND assignment operator
…

```c
#include <stdio.h>
main() {

   int a = 21;
   int c ;
   c = a;
   printf("Line 1 - = Operator Example, Value of c = %d\n", c );

   c += a;
   printf("Line 2 - += Operator Example, Value of c = %d\n", c );

   c -= a;
   printf("Line 3 - -= Operator Example, Value of c = %d\n", c );

   c *= a;
   printf("Line 4 - *= Operator Example, Value of c = %d\n", c );

   c /= a;
   printf("Line 5 - /= Operator Example, Value of c = %d\n", c );

   c = 200;
   c %= a;
   printf("Line 6 - %= Operator Example, Value of c = %d\n", c );

}
```

Arithmetic, relational and logical operators

- Arithmetic operators

  +, -, *, /, %

  ++, —

- Relational operators

  ==, !=, <, <=, >, >=

- Logical operators

  &&, ||, !

- Misc operators

  sizeof(), &, *, ?:

Associativity and precedence

- Associativity :
  left to right except for + - ! ~ ++ - - (type)* & sizeof

- Precedence:
  From the highest to the lowest:
  - () [] -> .
  - ++ -- - ! & * (unary operators)
  - * / %
  - + - (binary operators)
  - < > <= >=
  - == !=
  - &&
  - ||
  - = += -+ *= /= %=

```
main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;          // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n", e);

   e = ((a + b) * c) / d; e );   // (30) * (15/5)
   printf("Value of ((a + b) * c) / d is  : %d\n", e);

   e = (a + b) * (c / d);           //  20 + (150/5)
   printf("Value of (a + b) * (c / d) is  : %d\n", e);

   e = a + (b * c) / d;
   printf("Value of a + (b * c) / d is  : %d\n", e);

   return 0;
}
```

Decision making

- if-else
- if-else if-else … if-else
- switch
- ?: operator

```c
#include <stdio.h>
int main () {
   int a = 100;
   if( a == 10 ) {
      printf("Value of a is 10\n" );
   } else if( a == 20 ) {
      printf("Value of a is 20\n" );
   } else if( a == 30 ) {
      printf("Value of a is 30\n" );
   } else {
      printf("None of the values is matching\n" );
   }
   printf("Exact value of a is: %d\n", a );
   return 0;
}
```

```
switch(ch1) {
   case 'A':
      printf("This A is part of outer switch" );
      switch(ch2) {
         case 'A':
            printf("This A is part of inner switch" );
            break;
         case 'B': /* case code */
      }
      break;
   case 'B': /* case code */
}
```

```c
int main()
{
   int num;

   printf("Enter the Number : ");
   scanf("%d",&num);

   flag = ((num%2==0)?1:0);

   if(flag==0)
    printf("\nEven");
   else
      printf("\nOdd");
}
```

Can be coded as follow:

```c
int main()
{
   int num;
   printf("Enter the Number : ");
   scanf("%d",&num);

   (num%2 ==0)? printf("Even") : printf("Odd");
}
```

Loops

Three loop statements

- while
- do-while
- for

Loop related statements

- break
- continue

The while loop statement

while (<condition>) <statement>

```
while ( <condition>){

    <statement_1> … <statement_n>

}
```

A while loop example

```
int sum = 0;

int pos = 0;

int val;

while ( (val = array[pos] ) > -1 ) {

    sum += val;

    pos ++;

}
```

A second while loop example

```
int done = 0;

int sum = 0;

while (! done) {

        printf(“enter 0 to exit, else enter any number\n”);

         scanf(“%d”, &done);

        sum += done;

}

printf (“the sum of entered integers is %d\”, sum);
```

Remark: the loop is run at least one time.

The do-while loop statement

do <statement> while (<condition>) ;

```
do {
    <statement_1> … <statement_n>
} while ( <condition>) ;
```

A do-while loop example

For the previous example, the do-while is more appropriate

```
int done;

int sum = 0;

do {

        printf(“enter 0 to exit, else enter any number\n”);

         scanf(“%d”, &done);

        sum += done;

} while (! done);

printf (“the sum of entered integers is %d\”, sum);
```

The for loop statement

for (<initialization> ; <stop_condition> ; <increment>)
    <statement>

for (<initialization> ; <stop_condition> ; <increment>){

    <statement_1> … <statement_n>

}

A for loop example

```
int i;

int tab[100];

for ( i = 0; i < 100; i++) tab[i] = 0;
```

We have learned

- How to code, compile and run a simple example
- How to write simple algorithms with variables, expressions and statements