Sistemas Operativos

Grado en Ingeniería Informática del Software

Seminario 2

Introducción al lenguaje de progamación C

Características principales

- Lenguaje de nivel “mixto” :
  - Características principal de lenguaje de alto nivel:
    - Estructurado.
    - Uso de estructuras de datos complejas.
    - Robusto
    - Amplio conjunto de bibliotecas.
    - Independiente de la arquitectura
  - Algunas características de lenguaje de bajo nivel:
    - Tipado débil.
    - Permite la manipulación de direcciones de memoria.
    - Operaciones a nivel de bit.
  - Muy rápido.
  - Relativamente simple.
- Mejor opción como lenguaje de sistemas.

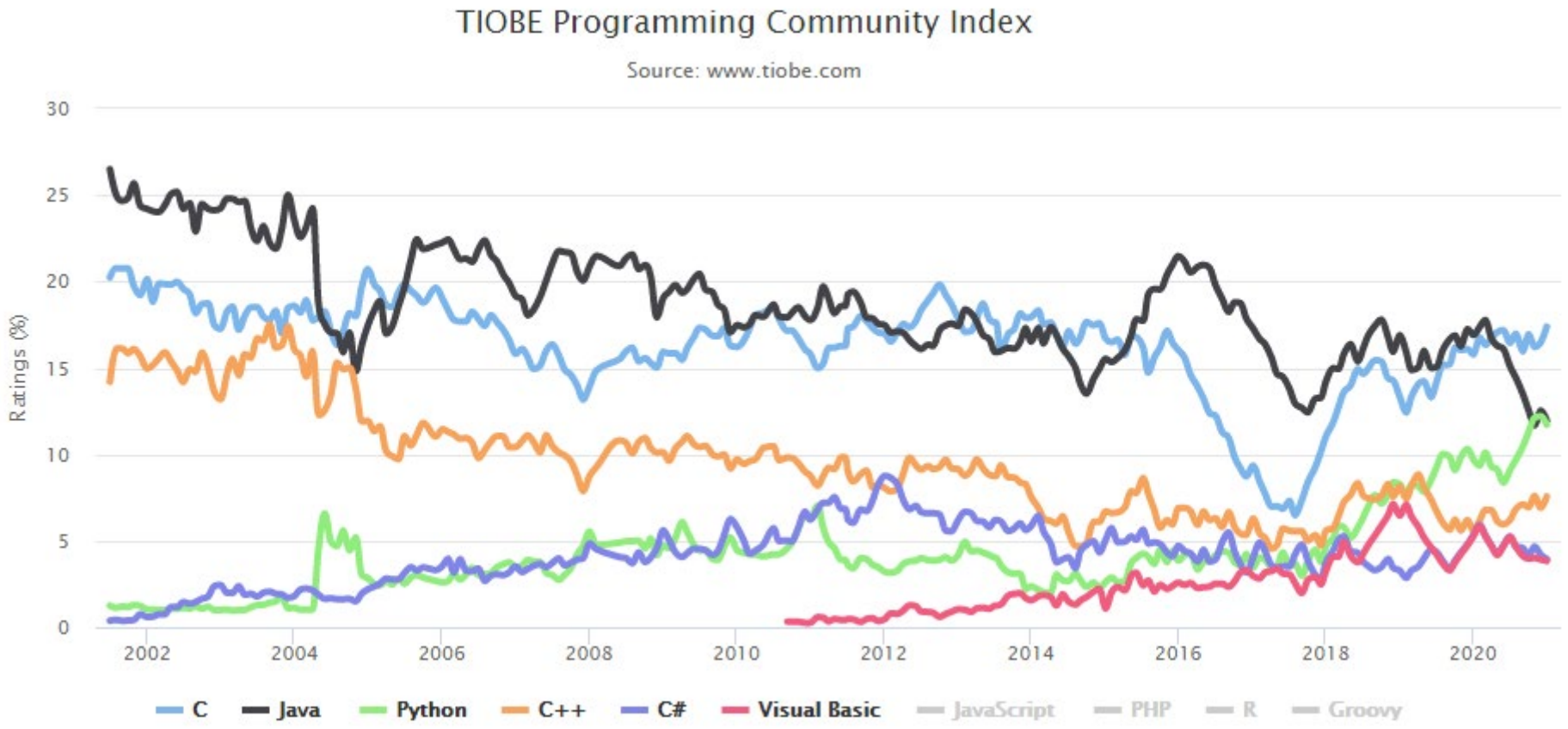
Características principales

- Bastante viejo (creado en 1972-73)
- Todavía se usa mucho ;-)

Jan 2021	Jan 2020	Change	Programming Language	Ratings	Change
1	2	↑	C	17.38%	+1.61%
2	1	↓	Java	11.96%	-4.93%
3	3		Python	11.72%	+2.01%
4	4		C++	7.56%	+1.99%
5	5		C#	3.95%	-1.40%
6	6		Visual Basic	3.84%	-1.44%
7	7		JavaScript	2.20%	-0.25%
8	8		PHP	1.99%	-0.41%
9	18	⇈	R	1.90%	+1.10%
10	23	⇈	Groovy	1.84%	+1.23%
11	15	⇈	Assembly language	1.64%	+0.76%

TIOBE Index for January 2021

Características principales




```
Highest Position (since 2001): #1 in Jan 2021
Lowest Position (since 2001): #2 in Apr 2020
Language of the Year: 2008, 2017, 2019
```

Estructura de un programa en C

- C es un lenguaje imperativo y procedimental.
- Un programa en C es un conjunto de funciones, donde debe haber una función *main*.

```
ÓRDENES DEL PREPROCESADOR

DECLARACIONES GLOBALES

DECLARACIÓN DE FUNCIONES
```

Funciones

- Proporcionan modularización: código más limpio; reutilización de código..
- Declaración de funciones:

```
tipoRetorno nombre(tipo par1, tipo par2, ...) {
 declaración variables locales;
 instrucciones;
}
```

- Llamada a una función

```
nombre(par1, par2, ...)
```

Estructura de un programa en C

- Nuestro primer programa:

```
/* The most popular program */

#include <stdio.h>

void main(){
   printf("Hello World.\n");
}
```

Orden para el preprocesador

Declaración de función

- Compilación y ejecución:

```
$ cc hello.c –o hello
$ ./hello
Hello World.
$
```

Función printf

- Incluida en <stdio.h>

```
printf(formating_string, param1, ...)
```

- Formating string: texto a mostrar, conteniendo marcas especiales a sustituir por los valores de los parámetros:
  - %d para enteros en decimal
  - %c para char
  - %f para float
  - %lf para double
  - %s para string
- Ejemplo:

```
printf(“El caracter %d en %s es %c\n",3,s1,s1[3]);
```

Estructura de un programa en C

- Nuestro Segundo programa:

```
#include <stdio.h>

int factorial (int n) {
 int f,c;

 f=1;
 for (c=1;c<=n;c++)
  f=f*c;
 return f;
}

main() {
 int n;
 printf("Give me one number: ");
 scanf("%d",&n);
 printf("Factorial(%d)=%d\n",n,factorial(n));
}
```

Orden para el preprocesador

Declaración de función

Variables locales

Declaración de función

Estructura de un programa en C

- Las variables globales son accesibles desde cualquier función.
- Las variables locales son accesibles sólo desde dentro de la función donde están declaradas.

Funciones
Parámetros

- Todos los parámetros se pasan por valor. Para usar un parámetro y modificar su valor hay que usar punteros:

```
#include <stdio.h>
void swap (int a, int b) {
 int tmp;

 tmp=a; a=b; b=tmp;
}

main() {
 int p1=1, p2=2;

 swap (p1, p2);
 printf("p1=%d; p2=%d\n",p1,p2);
}
```

```
p1=1; p2=2
```

```
#include <stdio.h>
void swap (int *a, int *b) {
 int tmp;

 tmp=*a; *a=*b; *b=tmp;
}

main() {
 int p1=1, p2=2;

 swap (&p1, &p2);
 printf("p1=%d; p2=%d\n",p1,p2);
}
```

```
p1=2; p2=1
```

Funciones

Uso de punteros como parámetros

- & operador de dirección, * operador de indirección.

```
#include <stdio.h>
main() {
 int a=666; // a es en realidad la dirección de memoria donde se guarda el valor
 int *b;    // b contiene la dirección de una variable entera (puntero)

 b=&a;         // b contiene ahora la dirección de a.

 printf("Before: %d\n",a);
 *b=10;     // La variable apuntada por b contiene ahora el valor 10
 printf("After: %d\n",a);
}
```

- Output:

```
Before: 666
After: 10
```

- No se inicializan punteros…

```
int* p = NULL; // Conveniente inicializar, y comprobar antes de acceder
…
if (p == NULL){
   printf("Cannot dereference pointer p.\n"); exit(1);
}
```

Tipos de datos
Tipos primitivos

- Tipos enteros (pueden ser con signo o sin signo):
  - char : 1 byte.
  - int: 4 bytes en Ritchie
  - short: 2 bytes en Ritchie
  - long: 8 bytes en Ritchie
- Tipos numéricos con decimales:
  - float: 4 bytes en Ritchie
  - double: 8 bytes en Ritchie
- No hay tipo Boolean. En su lugar se utiliza tipo entero:
  - 0-> false
  - != 0 -> true

- Ejemplos:

```
char c=´A´;
char c=100;
unsigned char byte=255;

int i=´a´;
int i=-2343234;
unsigned int ui=100000000;

float pi=3.14;

double long_pi=0.31415e+1;
```

- Declaración de un array:

```
int a[10]; // Se reserve memoria para el array
int c[]={1,2,3,7,8,9}; // Se reserve memoria y se da
                       //  valores iniciales
b=c;
for (i=0; i<10; i++)
  a[i]=i;
```

- No puedes usar funciones “size” como en Java:

```
 printf("Java stile size of array a: %d\n", sizeof(a));
 printf("Real Size of array a: %d\n",
sizeof(a)/sizeof(a[0]));
```

mostrará:

```
Java stile size of array a: 40
Real Size of array a: 10
```

Tipos de datos
Strings

- No existe un tipo “string” como tal en C.
- Se utilizan arrays de caracteres, terminados en \0.
- Se pueden manipular Strings así definidos usando las funciones de la biblioteca “string” :
  - `strcpy, strcmp, strcat, strstr, strchr`
- El espacio para cada string debe crearse antes de ser usado.

```
char s1[]="Hello "; char s2[]="world" ; char s3[1]; char s4[]=". Fine";
printf("Valores iniciales:           s1=%s s2=%s s3=%s s4=%s\n", s1, s2, s3, s4);
strcpy(s3, s1);
printf("Despues de \"strcpy(s3,s1)\": s1=%s s2=%s s3=%s s4=%s\n", s1, s2, s3, s4);
strcat(s3, s2);
printf("Despues de \"strcat(s3,s2)\": s1=%s s2=%s s3=%s s4=%s\n", s1, s2, s3, s4);
```

podría mostrar algo como:

```
Valores iniciales:          s1=Hello  s2=world s3= s4=. Fine
Despues de "strcpy(s3,s1)": s1=Hello  s2=ello  s3=Hello  s4=. Fine
Despues de "strcat(s3,s2)": s1=Hello  s2=ello ello  s3=Hello ello  s4=. Fine
```

Tipos de datos
Tipos enumerados

- Si una variable puede tener sólo algunos valores específicos, podemos usar un tipo *enum*:

```
enum ProcessStates {NEW, READY, EXECUTING, BLOCKED,
EXIT};
```

- Internamente, se usa un valor entero para representar cada valor (0, 1, 2, ...). Pero puedes usar tus propios valores:

```
enum INT_BITS {SYSCALL_BIT=2, CLOCKINT_BIT=9,
EXCEPTION_BIT=6};
```

Tipos de datos
struct

- Una *struct* es un conjunto de varias variables agrupadas bajo el mismo nombre, mostrando que son parte del mismo “concepto”:

```
struct date {
   int day;
   int month;
   int year;
   char *monthName;
};
struct date d1 = {1,1,2019, “January”};
struct date d2;
d2=d1; d2.day=5;
```

Tipos de datos
Nombres alternativos

- Se pueden crear un nombre alternativo para un tipo de datos::

```
typedef unsigned char boolean;
 boolean myBool;
typedef struct date date;
 date d1, d2;
typedef long myInt;
 myInt i;
```

- Ventajas:
  - Evitar problemas de portabilidad.
  - Código más claro.

Operadores

- Aritméticos:
  - `* +,-,*,/,% *`
  - `++,--,*=,...`
- Relacionales:
  - `<, >, <=, >=, ==, !=`
- Logicos:
  - `&&, ||, !, ? : (condition ? result_if_true : result_if_false)`
- Bits:
  - `&, |, ^, !, <<, >>`

Estructuras de control
Condicional

```
if (condición)
  instrucción;
else
 instrucción;
```

```
switch (var) {
 case value1: instrucciones;
              break;
 case value2: instrucciones;
              break;
      ...
 default: ...
}
```

Estructuras de control
Bucles

```
while (condición )
 instrucción;
```

```
for (init; condición ;incr)
 instrucción;
```

```
do
 instrucción;
while (condición );
```

`break:` sale del bucle

`continue:` salta a la siguiente iteración

Preprocesador

- Orden include
  - Para incluir bibliotecas estándard.

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

  - Para incluir bibliotecas/ficheros del usuario.

```
#include “processor.h”
```

- Definición de constantes:

```
#define INTERRUPT_TYPES 10
```

- Compilación condicional

```
#ifndef PROCESSOR_H
#define
  ...
#endif
```

Todas las órdenes del preprocesador comienzan con #

Programas con múltiples ficheros

- Para organizar el código, los programas se pueden dividir en módulos. Cada modulo se suele almacenar en dos ficheros:
- **Fichero de cabecera (*header*)**(*.h): contiene los prototipos de funciones y la definición de variables/tipos globales (no suele incluir código).
- **Fichero en sí**(*.c): contiene la implementación de las funciones, variables del modulo, ... Comienza incluyendo el fichero de cabecera.
- Para compilar, se deben compilar todos los módulos y enlazarlos juntos. Hay utilidades como make para automatizar el proceso.