CeNAT
Centro Nacional de Alta Tecnología

CNCA

KABRÉ
SUPERCOMPUTER

HPC SCHOOL
COSTA RICA

LENGUAJE DE PROGRAMACIÓN C

Josef Ruzicka

BREVE HISTORIA

Invención

Creado por Dennis Ritchie de Bell Laboratories (Hoy, Nokia Bell Labs) en 1972

Bases

Lenguaje sucesor a B (Creado por Ken Thompson)

Uso

Inicialmente sólo era utilizado en UNIX

USO DEL LENGUAJE C

¿Por qué usar C?

- Permite controlar el Hardware.
- Compilación y ejecución rápida.

Estructura de programas

- Archivos .c contienen la implementación del código.
- Archivos .h (header files) contienen interfaces con las definiciones de funciones y variables globales

ARCHIVO .h

```h
#ifndef MODULARIZED_SUM_H
#define MODULARIZED_SUM_H

#define A 5
#define B 2

/*
function that returns the sum of two integer parameters.
Example: modularized_sum(3,2) returns 5.
*/
int modularized_sum(int num_a, int num_b);
int modularized_sum_simple_code(int num_a, int num_b);

#endif //MODULARIZED_SUM_H
```

ARCHIVO .C

```c
#include "modularized_sum.h"

int modularized_sum(int num_a, int num_b)
{
    // local variable declaration
    int result;

    // local variable initialization
    result = 0;

    result = num_a + num_b;

    return result;
}

int modularized_sum_simple_code(int num_a, int num_b)
{
    // result = num_a + num_b;
    // return result;

    return num_a + num_b;
}
```

main (OTRO ARCHIVO.C)

```c
#include <stdio.h>
#include "modularized_sum.h"

void main(){
    int my_sum = modularized_sum(A, B);
    int my_simple_code_sum = modularized_sum_simple_code(5, 2);
    printf("my_sum: %d\n", my_sum);
    printf("my_simple_code_sum: %d\n", my_sum);
}
```

```c
#include <stdio.h>

int sum(int num_a, int num_b);
int sum_simple_code(int num_a, int num_b);

void main(){
    int my_sum = sum(5, 2);
    int my_simple_code_sum = sum_simple_code(5, 2);
    printf("my_sum: %d\n", my_sum);
    printf("my_simple_code_sum: %d\n", my_sum);
}

int sum(int num_a, int num_b)
{
    // local variable declaration
    int result;

    // local variable initialization
    result = 0;

    result = num_a + num_b;


    return result;
}

int sum_simple_code(int num_a, int num_b)
{
    // result = num_a + num_b;
    // return result;

    return num_a + num_b;
}
```

Código sin modularizar

"DIVIDE Y VENCERÁS"

COMPILACIÓN Y EJECUCIÓN

```
[jruzicka@login-2 hello_world]$ ls
hello_world.c  Makefile
[jruzicka@login-2 hello_world]$ gcc -o hello hello_world.c
[jruzicka@login-2 hello_world]$ ls
hello  hello_world.c  Makefile
[jruzicka@login-2 hello_world]$ ./hello
Hello World from CRHPCS!
[jruzicka@login-2 hello_world]$
```

COMPILACIÓN Y EJECUCIÓN CON Makefile

```
1  CC=gcc
2  DIRS:=./
3
4  HEADERS:=$(wildcard $(DIRS:%=%/*.h))
5  SOURCES:=$(wildcard $(DIRS:%=%/*.c))
6
7  make:
8          $(CC) -o exe $(SOURCES)
9
10 clean:
11         rm exe
```

```
[jruzicka@login-2 hello_world]$ ls
hello_world.c  Makefile
[jruzicka@login-2 hello_world]$ make
gcc -o exe .//hello_world.c
[jruzicka@login-2 hello_world]$ ls
exe  hello_world.c  Makefile
[jruzicka@login-2 hello_world]$ ./exe
Hello World from CRHPCS!
[jruzicka@login-2 hello_world]$ make clean
rm exe
[jruzicka@login-2 hello_world]$ ls
hello_world.c  Makefile
[jruzicka@login-2 hello_world]$
```

```
1 #!/bin/sh
2 #SBATCH --nodes=1                    #Nodes to use
3 #SBATCH --ntasks-per-node=1          #Tasks per node
4 #SBATCH --ntasks=1                   #MPI Tasks per node
5 #SBATCH --cpus-per-task=8            #Threads per node
6 #SBATCH -p nu                        #Kabre Partition
7 #SBATCH --time=10:00                 #Time
8 echo "Compiling"
9 make -j
10 echo "Running on $SLURM_JOB_NODELIST"
11 time  ./exe
12 make clean
13 echo "done"
```

EN KABRÉ, ARCHIVOS

batch y slurm

```
[jruzicka@login-2 hello_world]$ ls
hello.batch  hello_world.c  Makefile
[jruzicka@login-2 hello_world]$ sbatch hello.batch
Submitted batch job 75195
[jruzicka@login-2 hello_world]$ ls
hello.batch  hello_world.c  Makefile  slurm-75195.out
[jruzicka@login-2 hello_world]$ cat slurm-75195.out
Compiling
gcc -o exe .//hello_world.c
Running on nu-2a.cnca
Hello World from CRHPCS!

real    0m0.062s
user    0m0.001s
sys     0m0.006s
rm exe
done
```

TIPOS DE DATOS BÁSICOS

Char

caracteres, tiene un byte de capacidad. Ejm: ‘a’, ‘A’, ‘1’, ‘/’, ‘#’

Int

números enteros, con 2 o 4 bytes de capacidad. Ejm: 1, -25, 999

Float

números flotantes, con 4 bytes de capacidad. Ejm: 3.1416, 107.33, -0.5

Double

números reales, con 8 bytes de capacidad. Ejm: 1.7, 1.79769e+308

Otros

También existen otros como short, long, unsigned int, y algunos tipos más, que puede estudiar en la siguiente referencia si lo considera adecuado.

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

```c
#include <stdio.h>

/*
Program to test printing variables of different data types.
*/
void main()
{
    int my_int = 3;
    float my_float = 3.141592f;
    double my_double = 3.141592653589793;
    char my_char = 'p';

    printf("Value of pi is:\nint: %d\nfloat: %f\ndouble: %0.15lf\nchar:  %c\n", my_int, my_float, my_double, my_char);
}
```

```
[jruzicka@login-1 data_types]$ gcc -o data_types data_types.c
[jruzicka@login-1 data_types]$ ./data_types
Value of pi is:
int: 3
float: 3.141592
double: 3.141592653589793
char:  p
```

REFERENCIA PARA ESTUDIAR FORMAT SPECIFIERS:
HTTPS://WWW.TUTORIALSPOINT.COM/FORMAT-SPECIFIERS-IN-C

Estructuras de control: if.

USO DE BOOLEANOS

```c
#include <stdio.h>
#include <stdbool.h>
void main()
{
    int first_value;
    int second_value;

    bool first_value_is_even = false;
    bool second_value_is_even = false;

    printf("Insert an integer value\n");
    scanf("%d", &first_value);

    printf("Insert a second integer value\n");
    scanf("%d", &second_value);

    // if statements to check if input numbers are even or odd
    // making use of the modulo operator
    if (first_value % 2 == 0) {
        printf("%d is an even number\n", first_value);
        first_value_is_even = true;
    } else {
        printf("%d is an odd number\n", first_value);
    }

    if (second_value % 2 == 0) {
        printf("%d is an even number\n", second_value);
        second_value_is_even = true;
    }
    else if (second_value % 2 != 0)
    {
        printf("%d is an odd number\n", second_value);
    }

    // ternary conditional operator
    (first_value_is_even && second_value_is_even) ? printf("both numbers are even\n") : printf("at least one number is odd\n");
}
```

Estructuras de control: switch. OPERADORES LÓGICOS

```c
#include <stdio.h>
#include <stdbool.h>
void main() {
    bool a = false;
    bool b = false;
    bool c = false;
    int temp_value = 0;

    printf("variable a: insert 0 for false or 1 for true\n");
    scanf("%d", &temp_value);
    (temp_value == 1) ? (a = true) : (a = false);

    printf("variable b: insert 0 for false or 1 for true\n");
    scanf("%d", &temp_value);
    (temp_value == 1) ? (b = true) : (b = false);

    printf("variable c: insert 0 for false or 1 for true\n");
    scanf("%d", &temp_value);
    (temp_value == 1) ? (c = true) : (c = false);
```

Estructuras de control: switch. OPERADORES LÓGICOS

```
21      while (temp_value != 5){
22          // menu.
23          printf("\nLogical operators main menu\n1. check AND\n2. check OR\n3. check XOR\n4. check NOT\n5. exit program\n");
24
25          scanf("%d", &temp_value);
26
27          switch (temp_value) {
28              case 1:
29                  (a && b && c) ? printf("TRUE\n------\n") : printf("FALSE\n------\n");
30                  break;
31              case 2:
32                  (a || b || c) ? printf("TRUE\n------\n") : printf("FALSE\n------\n");
33                  break;
34              case 3:
35                  (a && (!b && !c) || b && (!a && !c) || c && (!a && !b)) ? printf("TRUE\n------\n") : printf("FALSE\n------\n");
36                  break;
37              case 4:
38                  (!a && !b && !c) ? printf("TRUE\n------\n") : printf("FALSE\n------\n");
39                  break;
40              default:
41                  printf("terminating program\n");
42                  break;
43          }
44      }
45  }
```

Ciclos (for, while y do while) y Estructuras de datos: Arreglos

OPERADORES ARITMÉTICOS

++, +=, *=, /=, - -

```c
#include <stdio.h>

void main()
{
    // array initialization
    int array[5] = {1,2,3,4,4};

    // pseudo code: array_length = (memory used by array) / (memory size of int data type)
    int array_length = sizeof(array)/sizeof(array[0]);

    array[4] = 5; // array = {1,2,3,4,5}

    int array_sum = 0;

    // print array, and calculate the sum of its values
    for (int i = 0; i < array_length; i++) {
        printf("%d ", array[i]);
        array_sum += array[i];
    }
    printf("\nSum of array values: %d\n", array_sum);

    /* subtract 1 from array_sum in each iteration while
     * array_sum is greater or equal than 10 */
    while (array_sum >= 10) {
        array_sum--;
        printf("array_sum: %d\n", array_sum);
    }
    printf("end of while loop\n");

    /* subtract 1 from array_sum in each iteration while
     * array_sum is less than 5.
     * note that the condition is not met */
    do {
        array_sum--;
        print("array_sum: %d\n", array_sum);
    } while (array_sum < 5);
}
```

```
[jruzicka@login-2 data_structures]$ ./array
1 2 3 4 5
Sum of array values: 15
array_sum: 14
array_sum: 13
array_sum: 12
array_sum: 11
array_sum: 10
array_sum: 9
end of while loop
array_sum: 8
```

```c
#include <stdio.h>

void main()
{
    // matrix initialization
    int matrix[3][3] = {{1,2,3},
                        {4,5,6},
                        {7,8,9}};

    int matrix_transposed[3][3];

    int matrix_row_length = sizeof(matrix)/sizeof(matrix[0]);
    int matrix_column_length = sizeof(matrix[0])/sizeof(matrix[0][0]);

    // transpose matrix
    for (int row = 0; row < matrix_row_length; row++) {
        for (int col = 0; col < matrix_column_length; col++) {
            matrix_transposed[col][row] = matrix[row][col];
        }
    }

    // print matrix
    for (int row = 0; row < matrix_row_length; row++) {
        for (int col = 0; col < matrix_column_length; col++) {
            printf("%d ",matrix_transposed[row][col]);
        }
        printf("\n");
    }
}
```

```
[jruzicka@login-2 data_structures]$ ./matrix
1 4 7
2 5 8
3 6 9
```

CICLOS Y ESTRUCTURAS DE DATOS:

Matrices

PUNTEROS

```c
#include <stdio.h>

void value_add(int);
void value_add_ptr(int*);

void main() {
    int value = 0;

    // pointer to value's address
    int* value_pointer = &value;

    value_add(value);
    printf("value add: %d\n", value);

    value_add_ptr(value_pointer);
    printf("value add ptr: %d\n", value);
}

void value_add (int value) {
    value++; // note that this function's value is a copy of main's value
}

void value_add_ptr (int *value) {
   (*value)++;
   //*value++; this increments the address pointed at by the pointer by 1
}
```

STRUCTS.

```c
#include <stdio.h>

typedef struct Vehicule {
    int wheel_count;
    char *model_name;
} vehicule;

void get_largest_vehicule(vehicule, vehicule);

void main() {
    vehicule bike;
    vehicule car;

    bike.wheel_count = 2;
    bike.model_name  = "ultra bike 100";

    car.wheel_count = 4;
    car.model_name  = "fast 2007";

    get_largest_vehicule(bike, car);
}

void get_largest_vehicule(vehicule a, vehicule b) {
    if (a.wheel_count > b.wheel_count) {
        printf("%s is the largest vehicule\n", a.model_name);
    }
    else if (a.wheel_count < b.wheel_count) {
        printf("%s is the largest vehicule\n", b.model_name);
    }
    else {
        printf("both vehicules are the same size\n");
    }
}
```

ASIGNACIÓN DE MEMORIA DINÁMICA

```c
#include <stdio.h>
#include <stdlib.h>
void main() {
    int value_count = 10;

    // malloc allocates uninitialized memory
    // calloc allocates memory initialized with 0
    int* ptr = (int*)malloc(value_count * sizeof(int));
    //int* ptr = (int*)calloc(value_count, sizeof(int));

    if (ptr != NULL) {
        printf("Memory allocated successfully\n");
    } else {
        printf("Memory allocation failed\n");
    }

    // realloc re-allocates memory, useful when previously allocated memory is insufficient
    ptr = realloc(ptr, (value_count+5) * sizeof(int));

    printf("Memory re-allocated successfully\n");
    // memory de-allocation
    free(ptr);

    printf("Memory de-allocated successfully\n");
}
```