Introducción a C++

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

2023-01-28

C++

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

C++ was developed by Bjarne Stroustrup at Bell Labs since 1979, as an extension of the C language as he wanted an efficient and flexible language similar to C, which also provided high-level features for program organization.

C++ is standardized by the International Organization for Standardization (ISO).

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg.

— Bjarne Stroustrup —

AZ QUOTES

Hello World

```cpp
#include <iostream>



int main()
{
  std::cout << "Hello, World !!" << std::endl;

  return 0;
}
```

```
bash:~> g++ -std=c++20 -o hello hello.cc ⏎
bash:~> hello ⏎

bash:~> clang++ -std=c++20 -o hello hello.cc ⏎
bash:~> hello ⏎
```

Hello World

```cpp
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello, World !!" << endl;

  return 0;
}
```

```
bash:~> g++ -o hello hello.cc ⏎
bash:~> hello ⏎

bash:~> clang++ -o hello hello.cc ⏎
bash:~> hello ⏎
```

Tipos de Datos

Fundamentales

- bool
- enum
- char
- int
- float, double

Compuestos

- array
- struct
- union
- bitfield
- string

Derivados

- punteros
- referencias

Especiales

- void

Tipos de Datos

Modificadores

- `signed`, `unsigned`
- `short`, `long`

Problema

- `char` puede ser `signed`/`unsigned`
- `int` puede ser `short`/`long`

dependiendo de la arquitectura (LP32, LP64).

Solución

Tipos estandarizados con tamaño:

```
#include <cstdint> // [u]int[8,16,32,64]_t
int32_t   var;
uint64_t  bigvar;
```

Tipos de Datos

Tipos enumerados

```
enum State { ON, OFF };
```

Se pueden especificar los valores numéricos para los símbolos:

```
enum State { ON = 0xFF, OFF = 0x00 };
enum Month { Jan = 1, Feb, Mar, Apr ... };
```

Son definiciones de TIPOS.
Al contrario que en C, aquí se han definido dos tipos llamados

State
Month

Tipos de Datos

Strings

```
#include <string>

string he = "Hello";
string wo;
string greet;

wo = "World";
greet = he+" "+wo;
```

Operaciones:

```
cout << greet << endl;
cout << greet.length() << endl;

string cmd="ls";
FILE* f = popen(cmd.c_str(),"r");
```

Tipos de Datos

Registros

```
1  struct Point
2  {
3    float x,y;
4  }
```

En C:

```
1  struct Point origin;
```

o bien

```
1  typedef struct Point Point;
2  ...
3  Point origin;
```

En C++, es una definición de TIPO:

```
1  Point origin;
```

Tipos de Datos

Uniones

```
union ByteSex
{
  uint16_t i;
  uint8_t  b[2];
  struct {
    uint8_t big;
    uint8_t little;
  };
}

ByteSex bx;
bx.i = 256;
cout << sizeof(bx) << endl;
cout << int(bx.b[0]) << " " << int(bx.b[1])   << endl;
cout << int(bx.big)  << " " << int(bx.little) << endl;
```

Tipos de Datos

Punteros y Memoria Dinámica

```
1   struct Node { ... }
2
3   Node* n = nullptr;
4   n = new Node;
5   ...
6   delete n;
7
8   Node* vn = new Node[16];
9   ...
10  delete[] vn;
```

Siempre con new/delete, nunca malloc/free !!!

(excepto para tratar con el SO...)

Tipos de Datos

Referencias

Una referencia define un alias a otra variable del mismo tipo:

```
1    int  a = 0;
2    int& ra = a;
3    ra++;
```

¡¡ **NO** es un puntero !!

Una referencia no se puede definir sin inicializar:
no existe la referencia "nula".
Se pueden utilizar independientemente:

```
1  int& data = that->info->deeply->buried.in.a->structure;
```

Pero sobre todo son útiles como parámetros y en otras estructuras de datos.

Namespaces

Los *espacios de nombres* o *namespaces* permiten al programador aislar definciones en su propio ámbito para que no colisionen con otras con el mismo identificador:

```
1  queue.h:
2
3  namespace Queue {
4    const int MAX_SIZE = 256;
5    ...
6  }
```

```
1  stach.h:
2
3  namespace Stack {
4    const int MAX_SIZE = 128;
5    ...
6  }
```

```
1  int qsz = Queue::MAX_SIZE;
2  int ssz = Stack::MAX_SIZE;
```

Los *namespaces* son abiertos y varios ficheros de cabecera pueden añadir definiciones al ámbito.

Comandos

Asignación (múltiple)

```
a = b;
a = b = c = d;
```

Condicional

```
1  if (a>0)
2  {
3    cout << "positive" << endl;
4  }
5  else
6  {
7    cout << "negative" << endl;
8  }
```

Comandos

Selección múltiple

```
1   switch(c)
2   {
3       case ' ':
4           cout << "spaze" << endl;
5           break;
6       case 'a':
7       case 'e':
8       case 'i':
9       case 'o':
10      case 'u':
11          cout << "vowel" << endl;
12          break;
13      default:
14          cout << "consonant" << endl;
15  }
```

Iteración

```
<init>
while (<cond>)
    <sentencia>

<init>
do
    <sentencia>
while (<cond>)

for (<init>; <cond>; <step>)
    <sentencia>
```

Secuenciadores

```
break
continue
```

Comandos

Iteración

Las instrucciones de iteración soportan la declaración de variables en la inicialización. Sólo son visibles en el interior de la sentencia.

Por ejemplo:

```
1    int i;
2    for (i=0; i<10; i++)
3        cout << "*";
```

O bien:

```
1    for (int i=0; i<10; i++)
2        cout << "*";
```

Incluso:

```
1    for (Node* n=list->head; n!=nullptr; n=n->next)
2        process_node(n);
```

Funciones

C++ soporta la definición de funciones..

```
1 int abs(int x)
2 {
3     return (a > 0 ? a : -a);
4 }
```

...recursivas:

```
1 int fact(int n)
2 {
3     return (n<=1 ? 1 : n*fact(n-1));
4 }
```

Funciones

C++ soporta la definición de valores por defecto para los parámetros de las funciones:

```
process.h: (especificacion)
  void process(Person p,bool doExtraStuffToo = false);

process.cc: (implementacion)
  void process(Person p,bool doExtraStuffToo)
  {
    doUsualProcess(p);
    if (doExtraStuffToo) doExtraProcess(p);
  }

main.cc: (utilizacion)
  process(person1);
  process(person2,true);
```

Ojo, tiene truco. ¿ Cuál ?

Funciones

El paso de parámetros se realiza por valor o por referencia.

	Valor	Referencia
Entrada	`int n`	`const int& n`
Salida	-	`int& n`

Qué tipo de argumentos pueden enviarse a cada uno de esos tipos de parámetros ?
Piensa como funcionaria el paso de parámetros con argumentos que fueran:

- variables
- constantes
- literales

Sobrecarga

C++ soporta la definición de varias funciones con el mismo nombre, mientras se diferencien en los argumentos que reciben:

sobrecarga independiente del contexto

```
1  datastructs.h:
2    void add(Tree& t,int x);
3    void add(List& l,int x);
```

```
1  datastructs.cc:
2    void add(Tree& t,int x)
3    {
4      ...
5    }
6
7    void add(List& l,int x)
8    {
9      ...
10   }
```

Acceso a bibliotecas C

Se pueden incluir bibliotecas de C, por ejemplo para el acceso a llamadas al sistema, pero para la mayoria existen versiones especiales para C++.

En C:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdint.h>
```

En C++:

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include <cstdint>
```

Acceso a bibliotecas C

Se pueden incluir bibliotecas de C, por ejemplo para el acceso a llamadas al sistema, pero para la mayoria existen versiones especiales para C++.

point.h:

```
struct Point
{
  float x,y;
};

#ifndef __cplusplus
typedef struct Point Point;
#endif
```

Preguntas ?

Introducción a C++

Tecnología de Programación

Adolfo Muñoz – Juan Magallón

Grado en Ingeniería Informática

Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas Universidad Zaragoza

2023-01-28