

ÍNDICE

Prefacio	25
Organización de la Unidad Didáctica	25
Cómo utilizar el libro	26
Objetivos docentes	27
1 Fundamentos de programación	29
1.1 Introducción	33
1.2 Evolución de los lenguajes de programación	34
1.2.1 La máquina de von Neumann	35
1.2.2 Lenguaje ensamblador	40
1.2.3 FORTRAN	41
1.2.4 ALGOL	47
1.2.5 LISP	49
1.2.6 COBOL	50
1.2.7 Prolog	50
1.2.8 SIMULA 67	51
1.2.9 Pascal	52
1.2.10 C	55
1.2.11 Modula-2	55
1.2.12 Ada	55
1.2.13 Smalltalk	56

1.2.14	C++	56
1.2.15	Java	59
1.3	Paradigmas de programación	61
1.3.1	Programación imperativa	61
1.3.2	Programación funcional	62
1.3.3	Programación lógica	64
1.3.4	Orientación a objetos	64
1.4	Métodos de implementación	66
1.4.1	Interpretación pura	67
1.4.2	Compilación	68
1.4.3	Sistema híbrido	71
1.4.4	Preprocesadores	72
1.5	Lecturas recomendadas	72
1.6	Ejercicios de autocomprobación	73
1.7	Soluciones de los ejercicios	76
2	Comenzando a programar en C++	79
2.1	Introducción	83
2.2	El programa “Hola mundo!”	83
2.3	Literales de tipo string	89
2.4	Salida por consola	90
2.5	Manipuladores del flujo de salida	92
2.5.1	Salida de datos en punto flotante	93
2.5.2	Salida de datos enteros	97
2.5.3	Salida de datos Booleanos	98
2.5.4	Anchura del dato en el flujo de salida	98
2.6	Flujos predefinidos de entrada y salida	100
2.7	Lecturas recomendadas	101

2.8	Ejercicios de autocomprobación	102
2.9	Soluciones de los ejercicios	104
3	Variables y tipos de datos: principios básicos	107
3.1	Introducción	111
3.2	Variables	111
3.2.1	Constantes	112
3.2.2	Ámbito	113
3.2.3	Visibilidad	115
3.3	Tipos de datos	116
3.3.1	Tipos primitivos	117
3.3.2	Tipos definidos por el programador	119
3.4	Arrays	120
3.5	Cadenas de caracteres	123
3.6	Punteros	124
3.7	Variables en memoria dinámica	125
3.8	Lecturas recomendadas	127
3.9	Ejercicios de autocomprobación	128
3.10	Soluciones de los ejercicios	130
4	Variables y tipos de datos: programación en C++	131
4.1	Introducción	135
4.2	Declaración de variables	135
4.3	Tipos de datos básicos	136
4.4	Límites numéricos	138
4.5	Inicialización de variables de tipos básicos	141
4.6	Tipos enumerados	144
4.7	Estructuras	145

4.8	Arrays	146
4.9	Tipos definidos en la librería estándar	149
4.9.1	Strings	149
4.9.2	Contenedores estándar	152
4.9.3	Flujos	154
4.10	Punteros	154
4.11	Variables en memoria dinámica	155
4.12	Ámbito y visibilidad de las variables	157
4.13	Lecturas recomendadas	159
4.14	Ejercicios de autocomprobación	160
4.15	Soluciones de los ejercicios	163
5	Asignaciones y expresiones: principios básicos	165
5.1	Introducción	169
5.2	Sentencia de asignación	169
5.3	Operadores	171
5.3.1	Asignaciones con operadores aritméticos	172
5.3.2	Incremento y decremento	173
5.4	Asociatividad y precedencia	174
5.4.1	Reglas de precedencia	175
5.4.2	Reglas de asociatividad	177
5.5	Sistema de tipos	177
5.5.1	Sobrecarga de los operadores	178
5.5.2	Conversiones de tipo	178
5.5.3	Verificación de tipos	181
5.6	Lecturas recomendadas	182
5.7	Ejercicios de autocomprobación	183
5.8	Soluciones de los ejercicios	185

6	Asignaciones y expresiones: programación en C++	187
6.1	Introducción	191
6.2	El operador asignación	191
6.3	Operadores aritméticos	192
6.4	Operadores relacionales y lógicos	193
6.5	Operadores << y >>	197
6.6	Operando con valores numéricos	199
6.7	Entrada por teclado	202
6.8	Operando con strings	209
6.8.1	Operadores	209
6.8.2	Funciones miembro	213
6.9	Operando con punteros	220
6.10	Relación entre punteros y arrays	225
6.11	Operando con vectores	227
6.11.1	Acceso a los componentes	227
6.11.2	Iteradores para vectores	230
6.11.3	Tamaño del vector	231
6.11.4	Assign e inicialización	232
6.11.5	Inserción y eliminación de componentes	233
6.11.6	Operadores relacionales	238
6.12	Operando con estructuras	240
6.12.1	Acceso a los miembros	240
6.12.2	Asignaciones de estructuras	243
6.12.3	Punteros a estructuras	243
6.12.4	Arrays de tipo estructura	246
6.12.5	Estructuras autorreferenciadas	246
6.13	Lecturas recomendadas	247

6.14	Ejercicios de autocomprobación	248
6.15	Soluciones de los ejercicios	252
7	Control del flujo del programa: principios básicos	257
7.1	Introducción	261
7.2	Sentencias de selección	261
7.2.1	Sentencias de selección con dos alternativas	262
7.2.2	Sentencias de selección múltiple	265
7.3	Sentencias iterativas	271
7.3.1	Control mediante contador	271
7.3.2	Control mediante expresión Booleana	276
7.3.3	Sentencias break y continue	278
7.4	Excepciones	279
7.5	Lecturas recomendadas	281
7.6	Ejercicios de autocomprobación	282
7.7	Soluciones de los ejercicios	285
8	Control del flujo del programa: programación en C++	289
8.1	Introducción	293
8.2	Sentencias de selección	293
8.2.1	Sentencia if	293
8.2.2	Sentencia switch	295
8.3	Sentencias iterativas	296
8.3.1	Bucle for	296
8.3.2	Bucles while y do-while	297
8.3.3	Sentencias break y continue	298
8.4	Excepciones	299
8.4.1	Captura y tratamiento de las excepciones	300

8.4.2	Tipos estándar de excepciones	304
8.4.3	Excepción <i>std::bad_alloc</i>	305
8.5	Entrada por teclado	306
8.6	Entrada y salida por fichero	315
8.7	Lecturas recomendadas	322
8.8	Ejercicios de autocomprobación	323
8.9	Soluciones de los ejercicios	329
9	Subprogramas: principios básicos	341
9.1	Introducción	345
9.2	Funciones	346
9.2.1	Definición	346
9.2.2	Invocación	347
9.2.3	Evaluación	348
9.2.4	Variables locales	350
9.3	Funciones recursivas	352
9.3.1	Recursividad lineal	353
9.3.2	Recursividad de cola	355
9.4	Procedimientos	356
9.4.1	Definición	357
9.4.2	Invocación	357
9.5	Lecturas recomendadas	358
9.6	Ejercicios de autocomprobación	359
9.7	Soluciones de los ejercicios	361
10	Subprogramas: programación en C++	365
10.1	Introducción	369
10.2	Definición de las funciones	369

10.3	Llamada a las funciones	372
10.4	Paso de parámetros a las funciones	372
10.5	Ámbito y visibilidad	378
10.6	Sentencia return	379
10.7	Punteros a funciones	381
10.8	Excepciones	385
10.9	Prototipos	391
10.10	Organización del programa en varios ficheros	393
10.11	Espacios de nombres	395
10.12	Lecturas recomendadas	396
10.13	Ejercicios de autocomprobación	398
10.14	Soluciones de los ejercicios	409
11	Estructuras de datos: principios básicos	421
11.1	Introducción	425
11.2	Listas	425
11.2.1	Operaciones sobre listas	426
11.2.2	Implementación	427
11.2.3	Pilas	431
11.2.4	Colas	432
11.3	Mapas	435
11.3.1	Operaciones sobre mapas	435
11.3.2	Implementación	435
11.4	Árboles	436
11.4.1	Operaciones sobre árboles	439
11.4.2	Implementación	440
11.5	Lecturas recomendadas	441
11.6	Ejercicios de autocomprobación	442

11.7	Soluciones de los ejercicios	445
12	Estructuras de datos: programación en C++	449
12.1	Introducción	453
12.2	Standard Template Library	453
12.2.1	Contenedores	453
12.2.2	Algoritmos	454
12.2.3	Iteradores	454
12.3	Listas	455
12.3.1	Declaración	457
12.3.2	Operaciones sobre el comienzo y el final de la lista	457
12.3.3	Iteradores	458
12.3.4	Inicialización	460
12.3.5	Inserción de elementos	463
12.3.6	Eliminar elementos	467
12.3.7	Movimiento de elementos entre listas	469
12.3.8	Ordenar los elementos	473
12.4	Pilas	475
12.5	Colas	476
12.6	Mapas	477
12.6.1	Declaración	479
12.6.2	Iteradores	479
12.6.3	Inserción de elementos	480
12.6.4	Búsqueda de elementos	486
12.6.5	Eliminar elementos	486
12.7	Lecturas recomendadas	488
12.8	Ejercicios de autocomprobación	489
12.9	Soluciones de los ejercicios	495

13 Algoritmos: principios básicos	501
13.1 Introducción	505
13.2 Paradigmas de diseño	505
13.3 Descripción del algoritmo	507
13.4 Complejidad	508
13.5 Algoritmos de ordenación	509
13.5.1 Método de la burbuja	510
13.5.2 Ordenación por inserción	511
13.5.3 Ordenación por mezcla	512
13.6 Lecturas recomendadas	514
13.7 Ejercicios de autocomprobación	515
13.8 Soluciones de los ejercicios	517
14 Algoritmos: programación en C++	523
14.1 Introducción	527
14.2 Contar elementos	527
14.3 Eliminar y reemplazar elementos	530
14.4 Invertir el orden de los elementos	531
14.5 Transformar los elementos	534
14.6 Lecturas recomendadas	537
14.7 Ejercicios de autocomprobación	538
14.8 Soluciones de los ejercicios	545
Índice alfabético	551
Bibliografía	559

CÓDIGO

1.1	Programa en Fortran 90 almacenado en un único fichero.	46
1.2	Función en Fortran 90.	46
1.3	Programa principal en Fortran 90.	46
1.4	Programa en Pascal almacenado en un único fichero.	54
1.5	Función en Pascal.	54
1.6	Fichero externs.h.	54
1.7	Programa principal en Pascal.	54
1.8	Programa en C++ almacenado en un único fichero.	58
1.9	Fichero externsC.h.	60
1.10	Ejemplo de una función en C++.	60
1.11	Ejemplo de un programa en C++.	60
2.1	Programa “ <i>Hola mundo!</i> ”.	83
2.2	Otra versión del programa “ <i>Hola mundo!</i> ”.	89
2.3	Ejemplo de uso de caracteres especiales en un literal de tipo string . .	90
2.4	Escritura en el flujo estándar de salida.	91
2.5	Escrituras sucesivas en el flujo estándar de salida.	91
2.6	Manipuladores de formato y precisión para números en punto flotante.	95
2.7	Manipuladores de la base para números enteros.	97
2.8	Manipuladores para datos Booleanos.	98
2.9	Ejercicio sobre el uso de los manipuladores de salida.	103
2.10	Modificación del programa “ <i>Hola, mundo!</i> ”.	104

2.11	Programa tras corregir los errores de compilación.	105
2.12	Programa que escribe la tabla en la consola.	105
4.1	Declaración de variables de algunos tipos básicos.	138
4.2	Límites de los tipos de datos básicos.	140
4.3	Inicialización por defecto de variables y arrays de tipos básicos. . . .	143
4.4	Declaración y uso de tipos enumerados.	145
4.5	Programa “ <i>Hola mundo!</i> ”, usando una variable.	149
4.6	Programa “ <i>Hola mundo!</i> ”, inicializando la variable.	151
4.7	Programa “ <i>Hola mundo!</i> ”, usando una constante.	151
4.8	Programa “ <i>Hola mundo!</i> ”, enmarcando la frase.	152
4.9	Declaración de variable en memoria dinámica y acceso a la misma. . .	156
4.10	Aplicación de las reglas de visibilidad en C++.	159
4.11	Programa correspondiente al Ejercicio 4.2.	160
4.12	Programa correspondiente al Ejercicio 4.3.	161
4.13	Programa correspondiente al Ejercicio 4.4.	161
4.14	Programa correspondiente al Ejercicio 4.5.	162
4.15	Programa que escribe la tabla en la consola usando variables.	163
6.1	Tabla de la verdad de NOT, AND y OR, usando <i>true</i> y <i>false</i>	195
6.2	Tabla de la verdad de NOT, AND y OR, usando 1 y 0.	195
6.3	Ejemplo de uso de variables complejas.	201
6.4	Entrada de nombre por teclado y saludo.	202
6.5	Entrada de nombre por teclado y saludo.	205
6.6	Entrada de nombres y edades por teclado.	206
6.7	Arrays unidimensionales: declaración, inicialización y entrada de valores por teclado.	207
6.8	Arrays bidimensionales: declaración, inicialización y entrada de valores por teclado.	208
6.9	Programa “ <i>Hola mundo!</i> ”, con concatenación de <i>strings</i>	210

6.10	Concatenación usando un flujo <i>stringstream</i>	211
6.11	Uso de algunos operadores de <i>strings</i>	212
6.12	Uso de algunas funciones miembro de las variables <i>string</i>	215
6.13	Uso de funciones miembro para búsqueda en <i>string</i>	217
6.14	Uso de funciones miembro para comparar <i>string</i>	219
6.15	Ejemplo de uso de los operadores <i>dirección-de</i> e <i>indirección</i>	220
6.16	Escritura en la variable apuntada.	221
6.17	Asignación de los punteros y de las variables apuntadas.	223
6.18	Ejemplo de acceso a variables en memoria dinámica: programa. . . .	224
6.19	Declaración de un array en memoria dinámica de componentes double y acceso a los componentes.	226
6.20	Acceso a componentes de vector mediante operador [].	227
6.21	Acceso a componentes de vector mediante función <code>at()</code>	229
6.22	Acceso a componentes de vector mediante iterador.	231
6.23	Inicialización y asignación de valores a un vector.	234
6.24	Añadir y eliminar componentes al final de un vector.	235
6.25	Inserción y eliminación de componentes en un vector.	236
6.26	Reserva de memoria para un vector.	238
6.27	Operaciones relacionales entre vectores.	239
6.28	Declaración, inicialización y acceso a estructuras.	242
6.29	Acceso a los miembros de una variable estructura mediante puntero, usando el operador punto.	245
6.30	Acceso a los miembros de una variable estructura mediante puntero, usando el operador <code>-></code>	245
6.31	Programa correspondiente al Ejercicio 6.3.	248
6.32	Programa correspondiente al Ejercicio 6.4.	249
6.33	Programa correspondiente al Ejercicio 6.7.	251
6.34	Programa solución del Ejercicio 6.1.	252

6.35	Programa solución del Ejercicio 6.2.	252
6.36	Programa solución al Ejercicio 6.5.	254
8.1	División de dos números.	294
8.2	Ejemplo de uso de break y continue	299
8.3	Excepciones de tipos básicos y enumerados.	303
8.4	Código para capturar la excepción <i>std::bad_alloc</i>	305
8.5	Entrada de datos por teclado usando una sentencia while	306
8.6	Entrada de datos por teclado, usando las funciones <i>clear</i> e <i>ignore</i> . . .	314
8.7	Uso de <i>getline</i> con el flujo <i>std::cin</i>	315
8.8	Lectura de fichero a variable del tipo char	319
8.9	Lectura de fichero a variable <i>string</i> y escritura a fichero.	320
8.10	Lectura de fichero a un vector.	321
8.11	Programa solución del Ejercicio 8.1.	329
8.12	Programa solución al Ejercicio 8.2.	330
8.13	Solución al Ejercicio 8.3 empleando una sentencia for	330
8.14	Solución al Ejercicio 8.3 empleando una sentencia while	330
8.15	Programa solución al Ejercicio 8.4, que realiza la suma de n términos de una serie geométrica de razón $1/2$, armónica o armónica alterna. .	331
8.16	Programa solución al Ejercicio 8.5: cálculo de los primeros N números primos.	332
8.17	Programa solución al Ejercicio 8.6: conversor de binario a decimal. . .	333
8.18	Programa solución al Ejercicio 8.7: producto de dos matrices (comienzo). .	334
8.19	Programa solución al Ejercicio 8.7: producto de dos matrices (continuación).	335
8.20	Programa solución al Ejercicio 8.8: media, varianza y desviación estándar.	336
8.21	Programa solución al Ejercicio 8.9: estimación del número pi.	337
8.22	Programa solución al Ejercicio 8.10: filtrado de una señal.	338

9.1	Función de Ackermann.	359
9.2	Función recursiva para el cálculo de la potencia de un número y programa principal desde el cual se invoca la función.	363
10.1	Función para el cálculo del factorial de un entero entre 1 y 10, usando códigos de error.	371
10.2	Función con parámetros y tipo de retorno estructura.	373
10.3	Paso de parámetros por referencia: punteros como parámetros formales y direcciones de memoria como parámetros actuales.	375
10.4	Paso de parámetros por referencia: referencias como parámetros formales y variables como parámetros actuales.	375
10.5	Función para el cálculo del valor mínimo y máximo de los componentes de un vector.	377
10.6	Función con variable local static	379
10.7	Error en la programación de una función.	380
10.8	Invocación a una función mediante un puntero.	382
10.9	Paso de un puntero a función como parámetro.	384
10.10	Función para división de dos enteros, que lanza excepción <i>invalid_argument</i>	386
10.11	Función que calcula el valor mínimo y máximo de los componentes de un vector, lanzando una excepción si el vector no contiene componentes.	388
10.12	Excepciones y encadenamiento de llamadas a funciones.	389
10.13	Captura y relanzamiento de las excepciones.	390
10.14	Prototipo y definición de la función para la división de dos enteros.	392
10.15	Prototipo de la función <i>division</i>	394
10.16	Definición de la función <i>division</i>	394
10.17	Definición de la función main	394
10.18	Prototipo de la función en un espacio de nombres.	397
10.19	Definición de la función en un espacio de nombres.	397
10.20	Invocación de la función definida en un espacio de nombres.	397

10.21	Programa solución al Ejercicio 10.1.	409
10.22	Programa solución al Ejercicio 10.2.	410
10.23	Programa solución al Ejercicio 10.3.	411
10.24	Programa solución al Ejercicio 10.4.	412
10.25	Programa solución al Ejercicio 10.5.	413
10.26	Programa solución al Ejercicio 10.6 (principio).	414
10.27	Programa solución al Ejercicio 10.6 (continuación).	415
10.28	Programa solución al Ejercicio 10.7.	416
10.29	Programa solución al Ejercicio 10.8 (comienzo).	417
10.30	Programa solución al Ejercicio 10.8 (parte final).	418
10.31	Programa solución al Ejercicio 10.9: declaración de las entidades del espacio de nombres.	418
10.32	Programa solución al Ejercicio 10.9: definición del espacio de nombres.	419
10.33	Programa solución al Ejercicio 10.9: definición de la ecuación diferencial y programa principal.	420
11.1	Programa solución al Ejercicio 11.1: creación de una lista enlazada simple.	445
11.2	Programa solución al Ejercicio 11.2: longitud de una lista enlazada.	446
11.3	Programa solución al Ejercicio 11.3: función PUSH de una pila.	446
12.1	Operaciones sobre el comienzo y final de una lista.	459
12.2	Inicialización (num,valor) de una lista y acceso mediante iterador.	461
12.3	Inicialización de una lista copiando parte de otra lista.	462
12.4	Inserción de elementos en una lista especificando el valor de los nuevos elementos.	465
12.5	Inserción en una lista de la copia de una sublista.	466
12.6	Borrado de elementos no terminales de una lista.	468
12.7	Manipulaciones mostradas en la Figura 12.2. Primera parte del programa: definición de las funciones.	471

12.8	Manipulaciones mostradas en la Figura 12.2. Parte final del programa: programa principal.	472
12.9	Ordenación de los elementos de una lista y fusión de dos listas.	474
12.10	Inserción, lectura y extracción de elementos de una pila.	475
12.11	Inserción, lectura y extracción de elementos de una cola.	477
12.12	Inserción y acceso de elementos en un mapa usando el operador [].	482
12.13	Inserción de elementos en el mapa usando la función miembro <i>insert</i>	485
12.14	Búsqueda de elementos en un mapa empleando la función <i>find</i>	487
12.15	Programa correspondiente al Ejercicio 12.3.	490
12.16	Programa solución al Ejercicio 12.2: muestra en orden inverso una secuencia de cadenas de caracteres previamente introducidas por teclado.	495
12.17	Programa solución al Ejercicio 12.4: empareja cadenas de caracteres que empiezan con mayúscula, con cadenas que comienzan con minúscula.	496
12.18	Programa solución al Problema 12.5: frecuencia de cada cadena de caracteres insertada por teclado.	497
12.19	Programa solución al Ejercicio 12.6: ordena una secuencia de números enteros introducidos por teclado empleando dos pilas.	498
12.20	Programa solución al Problema 12.7: búsqueda de un patrón en un fichero.	499
12.21	Programa solución al Ejercicio 12.8: estadístico del test K-S.	500
13.1	Programa solución al Ejercicio 13.3: ordenación por el método de la burbuja.	518
13.2	Programa solución al Ejercicio 13.4: algoritmo de ordenación por in- serción.	519
13.3	Programa solución al Ejercicio 13.5 (parte inicial): algoritmo de orde- nación por mezcla.	520
13.4	Programa solución al Ejercicio 13.5 (parte final): algoritmo de ordena- ción por mezcla.	521
14.1	Algoritmo que cuenta el número de elementos con cierto valor en un vector.	529

14.2	Algoritmo que cuenta el número de elementos de una secuencia que satisfacen cierto predicado.	530
14.3	Copia del contenido de un vector a otro, eliminando y reemplazando componentes.	532
14.4	Inversión del orden de los elementos.	533
14.5	Algoritmo <i>transform</i> aplicado a una lista.	535
14.6	Algoritmo <i>transform</i> aplicado a dos listas.	537
14.7	Programa del Ejercicio 14.4, que emplea algoritmos para eliminar, copiar y reemplazar componentes de un vector.	541
14.8	Programa solución al Ejercicio 14.1: estimación de π mediante un método de Montecarlo.	545
14.9	Programa solución al Ejercicio 14.2: cuenta el número de componentes de un vector iguales a 4, pares y mayores que 4.	546
14.10	Programa solución al Ejercicio 14.3: traduce notas numéricas a calificaciones (MH, SB, NB, A, SS) y muestra estadística.	547
14.11	Simulación de la difusión de partículas (parte inicial).	548
14.12	Simulación de la difusión de partículas (parte final).	549

3.3. TIPOS DE DATOS

El tipo de una variable indica qué *valores* puede tener y qué *operaciones* pueden realizarse sobre ella. La motivación para la definición de tipos en los lenguajes de programación surge a diferentes niveles:

- **Nivel de la máquina.** Los valores soportados directamente por la máquina son clasificados en *tipos primitivos*, tales como entero, carácter, real y Booleano. Por ejemplo, dado que la instrucción máquina para realizar una operación entre enteros es diferente a la instrucción para realizar esa misma operación entre reales, los compiladores necesitan tener información acerca del tipo de los datos, a fin de poder generar el código máquina correspondiente a las operaciones.
- **Nivel del lenguaje.** Además de los tipos primitivos, los lenguajes de programación soportan *tipos estructurados*, tales como arrays, estructuras, listas, etc. que son contruidos a partir de los tipos primitivos de datos. La construcción del lenguaje que permite definir tipos estructurados se denomina *constructor de tipo*.
- **Nivel del programador.** Los tipos definidos por el programador son agrupaciones de datos y funciones. Un ejemplo son los *tipos enumerados*. Otro las *clases*.

El tipo de dato de la variable condiciona el número de posiciones de memoria que son necesarias para almacenarla. Por ejemplo, el espacio de almacenamiento necesario para guardar una variable de tipo Booleano, cuyo valor puede ser *true* o *false*, en general no será el mismo que el necesario para guardar una variable de tipo real. Asimismo, no se necesita el mismo espacio para almacenar una variable simple, que para almacenar una variable vectorial.

El valor de la variable se almacena en memoria codificado como una o varias palabras binarias, las cuales normalmente se almacenan en un conjunto de posiciones de memoria consecutivas.

En función del tipo de dato de la variable, la palabra o palabras binarias que representan su valor se interpreta de una forma u otra. Por ejemplo, una misma secuencia de bits puede ser interpretada como un entero o como un carácter. Dependiendo del tipo de dato de la variable, la interpretación será una u otra.

3.3.1. Tipos primitivos

Los tipos de datos cuya definición no se basa en otros tipos de datos se denominan *tipos primitivos*. Los tipos primitivos soportados por los primeros lenguajes de programación eran únicamente numéricos. Correspondían con los tipos *número entero* y *número real*. Los lenguajes de programación actuales ofrecen una amplia gama de tipos primitivos de datos, entre los cuales típicamente están los descritos a continuación.

- **Entero.** Los números enteros son soportados directamente por el hardware del ordenador, ya que pueden ser codificados en base binaria como una cadena de bits, en la cual el bit más significativo representa el signo. Cuanto mayor sea la longitud (el número de bits) del tamaño de palabra que se usa para almacenar los números enteros, mayor será el rango de valores que puedan tomar los enteros.

Algunos lenguajes soportan diferentes tipos de números enteros. La diferencia entre ellos es el número de bits de la palabra empleada para almacenar los datos. Por ejemplo, Java soporta los cuatro tipos de enteros siguientes:

Tipo	Número de bits	Rango
<code>long</code>	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
<code>int</code>	32	-2.147.483.648 a 2.147.483.647
<code>short</code>	16	-32.768 a 32.767
<code>byte</code>	8	-128 a 127

- **Real en coma flotante.** Los números reales se representan en formato de *coma flotante*. Es decir, como un coeficiente multiplicado por 10 elevado a un exponente entero. El coeficiente es un número real compuesto por un único dígito entero, seguido de una coma y un número fijo de dígitos decimales. Por ejemplo, en el número en coma flotante 2.3462e-3 el coeficiente es 2.3462 y el exponente es -3.

Como en el caso de los tipos enteros, los lenguajes suelen soportar varios tipos real, de modo que el programador pueda seleccionar aquel cuyo rango y precisión en el coeficiente (esto es, número de decimales del coeficiente) se ajusta más a las necesidades de la aplicación. Por ejemplo, Java soporta dos tipos reales: simple precisión (**float**) y doble precisión (**double**).

Tipo	Número de bits	Rango
<code>double</code>	>64	1.7e-308 a 1.7e+308
<code>float</code>	>32	3.4e-38 a 3.4e+38

En aplicaciones de cálculo numérico es recomendable trabajar con reales en doble precisión, ya que permiten representar los números de manera más precisa. Esto es debido a que los números en doble precisión tienen al menos el doble de dígitos fraccionales en el coeficiente que los números en simple precisión. Por otra parte, en algunos casos trabajar en doble precisión es más rápido que en simple precisión, ya que los procesadores modernos están optimizados para trabajar en doble precisión. Normalmente las funciones matemáticas transcendentales (trigonométricas, raíz cuadrada, etc.) devuelven valores del tipo real con doble precisión.

- **Booleano.** Los datos de este tipo pueden tomar dos posibles valores: *true* y *false*. ALGOL 60 fue el primer lenguaje que introdujo este tipo de dato, el cual es soportado por casi todos los lenguajes que se desarrollaron posteriormente. Una excepción es el lenguaje C, en el cual las condiciones lógicas se representan mediante expresiones numéricas: cuando una expresión numérica se interpreta como una condición lógica, vale *true* si la expresión es distinta de cero y *false* si vale cero.

Aunque el ordenador internamente podría representar un valor Booleano mediante un único bit (por ejemplo, si el bit vale 0 representa *false*, y si vale 1 representa *true*), normalmente el dato Booleano se almacenan ocupando una posición de memoria, ya que es la unidad de memoria más pequeña que puede ser direccionada eficientemente.

- **Caracter.** Este tipo de datos se usa para almacenar caracteres. El conjunto de caracteres estándar conocido como ASCII (*American Standard Code for Information Interchange*) tiene 128 caracteres, y el conjunto extendido de 8 bits, ISO Latin-1, tiene 256 caracteres (dado que $2^8 = 256$). Este último es el soportado por C y C++. Así pues, los datos de tipo carácter son almacenados en palabras de 8 bits en C y C++.

Otro conjunto estándar de caracteres es *Unicode*, el cual define un conjunto completo e internacional de caracteres que incluye todos los caracteres que pueden encontrarse en todas las lenguas de la humanidad, incluyendo los caracteres de los alfabetos latín, griego, árabe, cirílico, hebreo y muchos más. Java fue el primer lenguaje en soportar este código, para lo cual los datos de tipo carácter en Java son almacenados en palabras de 16 bits.