

Tabla 2.1. Tipos de datos fundamentales (notación completa)

| | | | |
|----------------------|--------------------|--------------|-------------------|
| <i>Datos enteros</i> | char | signed char | unsigned char |
| | signed short int | signed int | signed long int |
| | unsigned short int | unsigned int | unsigned long int |
| <i>Datos reales</i> | float | double | long double |

La palabra **char** hace referencia a que se trata de un carácter (una letra mayúscula o minúscula, un dígito, un carácter especial, ...). La palabra **int** indica que se trata de un número entero, mientras que **float** se refiere a un número real (también llamado de punto o coma flotante). Los números enteros pueden ser positivos o negativos (**signed**), o bien esencialmente no negativos (**unsigned**); los caracteres tienen un tratamiento muy similar a los enteros y admiten estos mismos cualificadores. En los datos enteros, las palabras **short** y **long** hacen referencia al número de cifras o rango de dichos números. En los datos reales las palabras **double** y **long** apuntan en esta misma dirección, aunque con un significado ligeramente diferente, como más adelante se verá.

Esta nomenclatura puede simplificarse: las palabras **signed** e **int** son las opciones por defecto para los números enteros y pueden omitirse, resultando la Tabla 2.2, que indica la nomenclatura más habitual para los tipos fundamentales del C++.

Explicación: Se evalúa expresión y se considera el resultado de dicha evaluación. Si dicho resultado coincide con el valor constante expresión_cte_1, se ejecuta sentencia_1 seguida de sentencia_2, sentencia_3, ..., sentencia. Si el resultado coincide con el valor constante expresión_cte_2, se ejecuta sentencia_2 seguida de sentencia_3, ..., sentencia. En general, se ejecutan todas aquellas sentencias que están a continuación de la expresión_cte cuyo valor coincide con el resultado calculado al principio. Si ninguna expresión_cte coincide se ejecuta la sentencia que está a continuación de default. Si se desea ejecutar únicamente una sentencia_i (y no todo un conjunto de ellas), basta poner una sentencia break a continuación (en algunos casos puede utilizarse la sentencia return o la función exit()). El efecto de la sentencia break es dar por terminada la ejecución de la sentencia switch. Existe también la posibilidad de ejecutar la misma sentencia_i para varios valores del resultado de expresión, poniendo varios case expresión_cte seguidos.

EJEMPLO:

```
#include <stdio.h>

void main(void) {
    int opcion;
    printf("1. ALTAS\n");
    printf("2. BAJAS\n");
    printf("3. MODIFICA\n");
```

```
printrf("4.SALIR\n");
printrf("Elegir opción: ");
scanf("%d",&opcion);
switch(opcion)
{
    case 1:
        printf("Opción Uno");
        break;
    case 2:
        printf("Opción Dos");
        break;
    case 3:
        printf("Opción Tres");
        break;
    case 4:
        printf("Opción Salir");
        break;
    default: printf("Opción incorrecta");
}
getch();
}
```

1.) Para librerias:

las librerias las declaramos porque vamos a usar recursos que contienen ellas.[más detalles]

#include <iostream.h> —>, printf, scanf..

#include <conio.h> —> funcion getch()..

#include <string.h> —> para manipular cadenas

#include <math.h> —> para operaciones numericas

#include <time.h> —> para usar la hora

#include <stdio.h> —> para usar alternativas de entrda – salida como printf , scanf

2.) Para variables:

las variables las declaramos con el fin de tener un espacio para almacenar algun dato que va a cambiar con el tiempo. [más detalles]

char nombre; —> Declarando variable tipo caracter

int a,b,c; —> Declarando variable tipo entero

double sueldo —> Declarando variable tipo decimal

short contador —> Declarando variable tipo entero-corto

float —>

Sugerencia: leer cuanto espacio ocupa usar cada tipo de variable [Aqui]

Consejo: A las unicas variables que se deben dar valores iniciales son a:

– los contadores

– los acumuladores

Pero, ¿Cuál es la difencia entre ambos ?

Acumuladores: se incrementa o decrementa en un valor variable.

Ejemplo: sacar el promedio de un alumno, se suman las notas (que varian) y se divide para el numero de notas.

Contadores: se incrementa o decrementa en una forma constante.

Ejemplo: va contando de “1 en 1” ó de “-3 en -3” , etc...

3.) Para constantes:

las constantes las declaramos con el fin de tener un espacio para almacenar algun dato que no va a cambiar. [más detalles]

Se las puede declarar de dos formas:

Tomemos como ejemplo la formula para hallar el area de un triangulo:

¿ qué es lo que nunca cambia ?

La base puede variar, la altura puede variar. Pero como vemos el "2" es constante, sea cual sea la base o la altura el 2 se queda ahí. Entonces si queremos

declarar al "2" como una constante, podemos hacerlo de dos maneras:

1) anteponiéndole " #define " al nombre de la constante y luego el valor que le corresponde, así:

```
#define nomb 2
```

Nota: al usar éste metodo no se le pone el delimitador " ; " al final de la linea.

A continuación coloco un ejemplo en código, para que tengan una idea de como seria:

```
#include <iostream.h>
```

```
main()
```

```
{
```

```
#define nomb 2 ——> Declarada la constante de la forma 1.
```

```
int base, altura, resultado, exponente;
```

```
printf("Ingrese base: \n");
```

```
scanf ("%i",& base);
```

```
printf ("Ingrese altura: \n");
```

```
scanf "%i",& altura);
```

```
printf ("Ingrese exponente: \n");
```

```
scanf "%i",& exponente);
```

```
resultado= (base*altura)/exponente;
```

```
printf("El area del triangulo es: resultado);
```

```
}
```

2) anteponiéndole " const " seguido del tipo de variable que és, despues el nombre de la constante y luego el valor que va a contener, así:

```
const int nomb = 2;
```

Nota: a diferencia del metodo anterior, éste si lleva el delimitador " ; " al final de la linea.

A continuación coloco un ejemplo en código, para que tengan una idea de como seria: