Illustrations List ***(Main Page)***

Identifier	Description
va_list	A type suitable for holding information needed by macros **va_start**, **va_arg**, and **va_end**. To access the arguments in a variable-length argument list, an object of type **va_list** must be declared.
va_start	A macro that is invoked before the arguments of a variable-length argument list can be accessed. The macro initializes the object declared with **va_list** for use by the **va_arg** and **va_end** macros.
va_arg	A macro that expands to an expression of the value and type of the next argument in the variable-length argument list. Each invocation of **va_arg** modifies the object declared with **va_list** so that the object points to the next argument in the list.
va_end	A macro that facilitates a normal return from a function whose variable-length argument list was referred to by the **va_start** macro.

Fig. 18.1 The type and the macros defined in header **stdarg.h**.

```
1  // Fig. 18.2: fig18_02.cpp
2  // Using variable-length argument lists
3  #include <iostream.h>
4  #include <iomanip.h>
5  #include <stdarg.h>
6
7  double average( int, ... );
8
9  int main()
10 {
11    double w = 37.5, x = 22.5, y = 1.7, z = 10.2;
12
13    cout << setiosflags( ios::fixed | ios::showpoint )
14         << setprecision( 1 ) << "w = " << w << "\nx = " << x
15         << "\ny = " << y << "\nz = " << z << endl;
16    cout << setprecision( 3 ) << "\nThe average of w and x is "
17         << average( 2, w, x )
18         << "\nThe average of w, x, and y is "
19         << average( 3, w, x, y )
20         << "\nThe average of w, x, y, and z is "
21         << average( 4, w, x, y, z ) << endl;
22    return 0;
23 }
24
25 double average( int i, ... )
26 {
27    double total = 0;
```

Fig. 18.2 Using variable-length argument lists (part 1 of 2).

```
28    va_list ap;
29
30    va_start( ap, i );
31
```

```
32     for ( int j = 1; j <= i; j++ )
33        total += va_arg( ap, double );
34
35     va_end( ap );
36
37     return total / i;
38  }
```

```
w = 37.5
x = 22.5
y = 1.7
z = 10.2

The average of w and x is 30.000
The average of w, x, and y is 20.567
The average of w, x, y, and z is 17.975
```

Fig. 18.2 Using variable-length argument lists (part 2 of 2).

```
1   // Fig. 18.3: fig18_03.cpp
2   // Using command-line arguments
3   #include <iostream.h>
4   #include <fstream.h>
5
6   int main( int argc, char *argv[] )
7   {
8      if ( argc != 3 )
9         cout << "Usage: copy infile outfile" << endl;
10     else {
11        ifstream inFile( argv[ 1 ], ios::in );
12        if ( !inFile )
13           cout << argv[ 1 ] << " could not be opened" << endl;
14
15        ofstream outFile( argv[ 2 ], ios::out );
16        if ( !outFile )
17           cout << argv[ 2 ] << " could not be opened" << endl;
18
19        while ( !inFile.eof() )
20           outFile.put( static_cast< char >( inFile.get() ) );
21     }
22
23     return 0;
24  }
```

Fig. 18.3 Using command-line arguments.

```
1   // Fig. 18.4: fig18_04.cpp
2   // Using the exit and atexit functions
3   #include <iostream.h>
4   #include <stdlib.h>
5
6   void print( void );
7
8   int main()
9   {
10     atexit( print );       // register function print
11     cout << "Enter 1 to terminate program with function exit"
12          << "\nEnter 2 to terminate program normally\n";
```

```
13
14     int answer;
15     cin >> answer;
16
17     if ( answer == 1 ) {
18        cout << "\nTerminating program with function exit\n";
19        exit( EXIT_SUCCESS );
20     }
21
22     cout << "\nTerminating program by reaching the of main"
23          << endl;
24
25     return 0;
26  }
27
28  void print( void )
29  {
30     cout << "Executing function print at program termination\n"
31          << "Program terminated" << endl;
32  }
```

Fig. 18.4 Using functions **exit** and **atexit** (part 1 of 2).

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
: 1

Terminating program with function exit
Executing function print at program termination
Program terminated
```

```
Enter 1 to terminate program with function exit
Enter 2 to terminate program normally
: 2

Terminating program by reaching end of main
Executing function print at program termination
Program terminated
```

Fig. 18.4 Using functions **exit** and **atexit** (part 2 of 2).

Signal	Explanation
SIGABRT	Abnormal termination of the program (such as a call to **abort**).
SIGFPE	An erroneous arithmetic operation, such as a divide-by-zero or an operation resulting in overflow.
SIGILL	Detection of an illegal instruction.
SIGINT	Receipt of an interactive attention signal.
SIGSEGV	An invalid access to storage.
SIGTERM	A termination request sent to the program.

Fig. 18.5 The signals defined in header **signal.h**.

```
1  // Fig. 18.6: fig18_06.cpp
2  // Using signal handling
3  #include <iostream.h>
4  #include <iomanip.h>
5  #include <signal.h>
6  #include <stdlib.h>
7  #include <time.h>
8
9  void signal_handler( int );
10
11 int main()
12 {
13    signal( SIGINT, signal_handler );
14    srand( time( 0 ) );
15
16    for ( int i = 1; i < 101; i++ ) {
17       int x = 1 + rand() % 50;
18
19       if ( x == 25 )
20          raise( SIGINT );
21
22       cout << setw( 4 ) << i;
23
24       if ( i % 10 == 0 )
25          cout << endl;
26    }
27
28    return 0;
29 }
30
31 void signal_handler( int signalValue )
32 {
33    cout << "\nInterrupt signal (" << signalValue
34         << ") received.\n"
35         << "Do you wish to continue (1 = yes or 2 = no)? ";
36
37    int response;
38    cin >> response;
39
40    while ( response != 1 && response != 2 ) {
41       cout << "(1 = yes or 2 = no)? ";
42       cin >> response;
43    }
44
45    if ( response == 1 )
46       signal( SIGINT, signal_handler );
47    else
48       exit( EXIT_SUCCESS );
49 }
```

Fig. 18.6 Using signal handling (part 1 of 2).

```
   1   2   3   4   5   6   7   8   9  10
  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30
  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50
  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70
  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88
Interrupt signal (4) received.
Do you wish to continue (1 = yes or 2 = no)? 1
  89  90
  91  92  93  94  95  96  97  98  99 100
```

Fig. 18.6 Using signal handling (part 2 of 2).

```
1  // Fig. 18.7: fig18_07.cpp
2  // Using goto
3  #include <iostream.h>
4
5  int main()
6  {
7     int count = 1;
8
9     start:                    // label
10       if ( count > 10 )
11          goto end;
12
13       cout << count << "  ";
14       ++count;
15       goto start;
16
17    end:                      // label
18       cout << endl;
19
20    return 0;
21 }
```

```
 1  2  3  4  5  6  7  8  9  10
```

Fig. 18.7 Using `goto`.

```
1  // Fig. 18.8: fig18_08.cpp
2  // An example of a union
3  #include <iostream.h>
4
5  union Number {
6     int x;
7     float y;
8  };
```

Fig. 18.8 Printing the value of a `union` in both member data types (part 1 of 2).

```
 9
10  int main()
11  {
12     Number value;
13
14     value.x = 100;
15     cout << "Put a value in the integer member\n"
16          << "and print both members.\nint:   "
17          << value.x << "\nfloat: " << value.y << "\n\n";
18
19     value.y = 100.0;
20     cout << "Put a value in the floating member\n"
21          << "and print both members.\nint:   "
22          << value.x << "\nfloat: " << value.y << endl;
23     return 0;
24  }
```

```
Put a value in the integer member
and print both members.
int:   100
float: 3.504168e-16

Put a value in the floating member
and print both members.
int:   0
float: 100
```

Fig. 18.8 Printing the value of a **union** in both member data types (part 2 of 2).

```
 1  // Fig. 18.9: fig18_09.cpp
 2  // Using an anonymous union
 3  #include <iostream.h>
 4
 5  int main()
 6  {
 7     // Declare an anonymous union.
```

Fig. 18.9 Using an anonymous **union** (part 1 of 2).

```
 8     // Note that members b, d, and f share the same space.
 9     union {
10        int b;
11        double d;
12        char *f;
13     };
14
15     // Declare conventional local variables
16     int a = 1;
17     double c = 3.3;
18     char *e = "Anonymous";
19
20     // Assign a value to each union member
21     // successively and print each.
22     cout << a << ' ';
23     b = 2;
24     cout << b << endl;
25
26     cout << c << ' ';
27     d = 4.4;
28     cout << d << endl;
```

```
29
30     cout << e << ' ';
31     f = "union";
32     cout << f << endl;
33
34     return 0;
35  }
```

```
1 2
3.3 4.4
Anonymous union
```

Fig. 18.9 Using an anonymous **union** (part 2 of 2).