三、填空题

1. C++语言是在\_\_\_\_\_\_\_\_\_语言的基础上发展起来的。

2. C++语言的编译单位是扩展名为\_\_\_\_\_\_\_\_\_\_的\_\_\_\_\_\_\_\_\_\_文件。

3. 行尾使用注释的开始标记符为\_\_\_\_\_\_\_\_\_\_。

4. 多行注释的开始标记符和结束标记符分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

5. 用于输出表达式值的标准输出流对象是\_\_\_\_\_\_\_\_\_。

6. 用于从键盘上为变量输入值的标准输入流对象是\_\_\_\_\_\_\_\_。

7. 一个完整程序中必须有一个名为\_\_\_\_\_\_\_\_的函数。

8. 一个函数的函数体就是一条\_\_\_\_\_\_\_\_\_语句。

9. 当执行cin语句时，从键盘上输入每个数据后必须接着输入一个\_\_\_\_\_\_\_\_符，然后才能继续输入下一个数据。

10. 在C++程序中包含一个头文件或程序文件的预编译命令为\_\_\_\_\_\_\_\_\_\_。

11. 程序中的预处理命令是指以\_\_\_\_\_\_字符开头的命令。

12. 一条表达式语句必须以\_\_\_\_\_\_作为结束符。

13. 在#include命令中所包含的头文件，可以是系统定义的头文件，也可以是\_\_\_\_\_\_\_\_定义的头文件。

14. 使用#include命令可以包含一个头文件，也可以包含一个\_\_\_\_\_\_文件。

15．一个函数定义由\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_两部分组成。

16．若一个函数的定义处于调用它的函数之前，则在程序开始可以省去该函数的\_\_\_\_\_\_语句。

17．C++头文件和源程序文件的扩展名分别为\_\_\_\_\_\_和\_\_\_\_\_\_。

18．程序文件的编译错误分为\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_两类。

19．当使用\_\_\_\_\_\_\_保留字作为函数类型时，该函数不返回任何值。

20．当函数参数表用\_\_\_\_\_\_保留字表示时，则表示该参数表为空。

21．从一条函数原型语句“int fun1(void);”可知，该函数的返回类型为\_\_\_\_\_\_，该函数带有\_\_\_\_\_\_个参数。

22. 当执行cout语句输出endl数据项时，将使C++显示输出屏幕上的光标从当前位置移动到\_\_\_\_\_\_\_\_的开始位置。

23. 假定x=5，y=6，则表达式x++\*++y的值为\_\_\_\_\_\_\_\_\_\_。

24. 假定x=5，y=6，则表达式x--\*--y的值为\_\_\_\_\_\_\_\_\_\_。

25. 假定x=5，y=6，则执行表达式y\*=x++计算后，x和y的值分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

26. 假定x=5，y=6，则执行表达式y+=x--计算后，x和y的值分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

27. C++常数0x145对应的十进制值为\_\_\_\_\_\_\_\_\_\_。

28. C++常数0345对应的十进制值为\_\_\_\_\_\_\_\_\_\_。

29. 十进制常数245对应的十六进制的C++表示为\_\_\_\_\_\_\_\_\_\_。

30. 十进制常数245对应的八进制的C++表示为\_\_\_\_\_\_\_\_\_\_。

31. signed char类型的值域范围是\_\_\_\_\_\_\_\_至\_\_\_\_\_\_\_\_\_\_之间的整数。

32. int和float类型的数据分别占用\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_个字节。

33. float和double类型的数据分别占用\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_个字节。

34. bool和char类型的数据分别占用\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_个字节。

35. unsigned short int和int类型的长度分别为\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

36. 字符串”This\’s a book.\n”的长度为\_\_\_\_\_\_\_\_\_\_。

37. 字符串”\nThis\’s a pen\n\n”的长度为\_\_\_\_\_\_\_\_\_\_。

38. 在C++中存储字符串”abcdef”至少需要\_\_\_\_\_\_\_\_个字节。

39. 在C++中存储字符串”a+b=c”至少需要\_\_\_\_\_\_\_\_个字节。

40. 假定x和y为整型，其值分别为16和5，则xy和x/y的值分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

41. 假定x和y为整型，其值分别为16和5，则x/y和double(x)/y的值分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

42. 假定x是一个逻辑量，则x  true的值为\_\_\_\_\_\_\_\_\_\_。

43. 假定x是一个逻辑量，则x || true的值为\_\_\_\_\_\_\_\_\_\_。

44. 假定x是一个逻辑量，则x  false的值为\_\_\_\_\_\_\_\_\_\_。

45. 假定x是一个逻辑量，则x || false的值为\_\_\_\_\_\_\_\_\_\_。

46. 假定x是一个逻辑量，则!x || false的值为\_\_\_\_\_\_\_\_\_\_。

47. 假定x是一个逻辑量，则x  !x的值为\_\_\_\_\_\_\_\_\_\_。

48. 假定x是一个逻辑量，则x || !x的值为\_\_\_\_\_\_\_\_\_\_。

49. 数学算式 的C++表达式为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

50. 数学算式 的C++表达式为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

51. 数学算式 的C++表达式为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

52. 设enum Printstatus{ready,busy,error}; 则 cout<<busy的输出结果是\_\_\_\_\_\_\_\_。

53. 设enum Printstatus{ready=2,busy,error}; 则cout<<busy的输出结果是\_\_\_\_\_\_\_\_。

54. 常数-4.205和6.7E-9分别具有\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_位有效数字。

55. 枚举类型中的每个枚举值都是一个\_\_\_\_\_\_\_\_\_\_，它的值为一个\_\_\_\_\_\_\_\_\_\_。

56. 常数100和3.62的数据类型分别为\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

57. 若x=5, y=10, 则计算y\*=++x表达式后，x和y的值分别为\_\_\_\_\_\_和\_\_\_\_\_\_。

58. 假定x和ch分别为int型和char型，则sizeof(x)和sizeof(ch)的值分别为\_\_\_\_\_\_和\_\_\_\_\_\_。

59. 假定x=10，则表达式x<=10?20:30的值为\_\_\_\_\_\_。

60. 表达式sqrt(81)和pow(6,3)的值分别为\_\_\_\_\_\_和\_\_\_\_\_\_。

61. 含随机函数的表达式rand()20的值在\_\_\_\_\_至\_\_\_\_\_\_区间内。

62. 在switch语句中，每个语句标号所含关键字case后面的表达式必须是\_\_\_\_\_\_\_。

63. 在if语句中，每个else关键字与它前面同层次并且最接近的\_\_\_\_\_\_\_\_关键字相配套。

64. 作为语句标号使用的C++保留字case和defaule只能用于\_\_\_\_\_\_\_\_语句的定义体中。

65. 执行switch语句时，在进行作为条件的表达式求值后，将从某个匹配的标号位置起向下执行，当碰到下一个标号位置时（停止/不停止）\_\_\_\_\_\_\_\_执行。

66. 若while循环的“头”为“while(i++<=10)”，并且i的初值为0，同时在循环体中不会修改i的值，则循环体将被重复执行\_\_\_\_\_\_次后正常结束。

67. 若do循环的“尾”为“while(++i<10)”，并且i的初值为0，同时在循环体中不会修改i的值，则循环体将被重复执行\_\_\_\_\_\_次后正常结束。

68. 当在程序中执行到\_\_\_\_\_\_\_\_语句时，将结束本层循环类语句或switch语句的执行。

69. 当在程序中执行到\_\_\_\_\_\_\_\_语句时，将结束所在循环语句中循环体的一次执行。  
    70. 在程序中执行到\_\_\_\_\_\_\_\_语句时，将结束所在函数的执行过程，返回到调用该函数的位置。

71．在程序执行完\_\_\_\_\_\_\_\_函数调用后，将结束整个程序的执行过程，返回到C++集成开发窗口。

72. 元素类型为int的数组a[10]共占用\_\_\_\_\_\_\_\_字节的存储空间。

73. 元素类型为double的二维数组a[4][6]共占用\_\_\_\_\_\_\_\_字节的存储空间。

74. 元素类型为char的二维数组a[10][30]共占用\_\_\_\_\_\_\_\_字节的存储空间。  
    75. 存储字符’a’和字符串”a”分别需要占用\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_个字节。

76. 空串的长度为\_\_\_\_\_\_\_\_，存储它需要占用\_\_\_\_\_\_\_\_个字节。

77. 字符串”\’a\’xy=4\n”的长度为\_\_\_\_\_\_\_\_。

78. 字符串”a:\\xxk\\数据”的长度为\_\_\_\_\_\_\_\_。

79. 用于存储一个长度为n的字符串的字符数组的长度至少为\_\_\_\_\_\_\_\_。  
    80. 若a是一个字符数组，则从键盘上向该数组输入一个字符串的表达式为\_\_\_\_\_\_\_\_\_\_。

81. 若a是一个字符数组，则向屏幕输出a中所存字符串的表达式为\_\_\_\_\_\_\_\_\_\_。

82. 一个二维字符数组a[10][20]能够存储\_\_\_\_\_\_\_\_个字符串，每个字符串的长度至多为\_\_\_\_\_\_\_\_。

83. 对一个二维字符数组a进行初始化的数据为{”123”,”456”,”789”}，则a[1]元素对应的字符串为\_\_\_\_\_\_\_\_\_\_。

84. strlen(”apple”)的值为\_\_\_\_\_\_\_\_，strcmp(”a”,”A”)的值为\_\_\_\_\_\_\_\_。

85. 若需要把一个字符串”aaa”赋值到字符数组a中，则需要执行\_\_\_\_\_\_\_\_函数的调用来实现。

85. 假定对数组a[]进行初始化的数据为{2,7,9,6,5,7,10}，则a[2]和a[5]分别被初始化为\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

86. 假定对二维数组a[3][4]进行初始化的数据为{{3,5,6},{2,8},{7}}，则a[1][1]和a[2][3]分别被初始化为\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

87. 在C++语言中，一个函数由函数头和\_\_\_\_\_\_\_\_\_\_组成。

88. 重载一个函数的条件是：该函数必须在参数的个数或参数的\_\_\_\_\_\_\_\_\_\_上与其它同名函数有所不同。

89. 如果一个函数只允许同一程序中的函数调用，则应在该函数定义前加上\_\_\_\_\_\_\_\_\_\_C++保留字。

90. 若“double x=100;”是文件F1.CPP中的一个全局变量定义语句，若文件F2.CPP中的某个函数需要访问此x , 则应在文件F2.CPP中添加对x的声明语句为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

91. 定义一个函数模板要用到的第一个修饰符是\_\_\_\_\_\_\_\_\_\_\_\_。

92. 在函数模板的参数中，用class修饰的参数称为\_\_\_\_\_\_\_\_\_\_参数。

93. 如果一个函数直接或间接地调用自身，这样的调用称为\_\_\_\_\_\_\_\_\_\_\_\_调用。

94. 已知int cubin(int n){return n\*n\*n;}和double cubin(double n){return n\*n\*n;}是一个函数模板的两个实例，假定类型参数用T表示，则该函数模板的定义是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

95. 对于无返回值函数，定义函数时要用\_\_\_\_\_\_\_\_\_\_修饰函数类型。

96. 如果一个函数定义中使用了\_\_\_\_\_\_\_\_\_\_修饰，则该函数不允许被其它文件中的函数调用。

97. 如果一个函数中有多个默认参数，则默认参数必须全部处在形参表的\_\_\_\_\_\_\_\_部分。

98. 定义外部变量时，不用存储类说明符\_\_\_\_\_\_\_\_\_，而声明外部变量时用它。

99．调用系统函数时，要先使用#include命令包含该系统函数的原型语句所在的\_\_\_\_\_\_\_\_\_\_。

100. 函数形参的作用域是该函数的\_\_\_\_\_\_\_\_\_\_\_\_。

101. C++提供的预处理命令有宏定义命令，条件编译命令和\_\_\_\_\_\_\_\_\_\_\_\_。

102. 程序的编译是以\_\_\_\_\_\_\_\_\_\_为单位进行的。

103. C++程序运行时的内存空间可以分成全局数据区，堆区，栈区和\_\_\_\_\_\_\_\_\_\_。

104. 全局变量和静态局部变量具有静态生存期，存放在内存的\_\_\_\_\_\_\_\_\_\_\_\_区中。

105. 局部变量具有局部生存期，存放在内存的\_\_\_\_\_\_\_\_\_\_\_\_区中。

106. 若二维数组a有m列，设a[0][0]位于数组的第一个位置上，则计算任一元素a[i][j]在数组中位置序号的公式为\_\_\_\_\_\_\_\_\_\_\_\_。

107. 若有定义“double a[3][5];”，则a 数组中行下标和列下标的最大值分别为\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_。

108. 若有定义“struct AA {int a; char b; double c;}x;”,则x占用空间大小为\_\_\_\_\_\_\_\_字节。

109. 当定义一个结构体变量时，系统分配给该变量的内存大小等于各成员所需内存大小的\_\_\_\_\_\_\_。

110. 一个指针类型的对象占用内存的\_\_\_\_\_\_\_\_个字节的存储空间。

111. 一个指针指向一个数据对象，它保存着该数据对象的\_\_\_\_\_\_\_\_，若数据对象为DataType类型，则相应的指针类型为\_\_\_\_\_\_\_\_\_\_。

112. 若要把一个整型指针p转换为字符指针，则采用的强制转换表达式为\_\_\_\_\_\_\_\_。

113. 假定一个数据对象为int\*类型，则指向该对象的指针类型为\_\_\_\_\_\_\_。

114. 假定p是一个指向整数对象的指针，则用\_\_\_\_\_\_\_\_表示该整数对象，用\_\_\_\_\_\_\_\_表示指针变量p的地址。

115. 假定p是一个指针，则\*p++运算首先访问\_\_\_\_\_\_\_\_，然后使\_\_\_\_\_\_\_\_的值增1。

116. 假定p是一个指针，则(\*p)++运算首先访问\_\_\_\_\_\_\_\_，然后使\_\_\_\_\_\_\_\_的值增1。

117. 假定p所指对象的值为25，p+1所指对象的值为42，则\*p++的值为\_\_\_\_\_\_\_\_。

118．假定p所指对象的值为25，p+1所指对象的值为42，则\*++p的值为\_\_\_\_\_\_\_\_。

119. 假定p所指对象的值为25，p+1所指对象的值为42，则执行(\*p)++运算后，p所指对象的值为\_\_\_\_\_\_\_\_。

120．假定p所指对象的值为25，p+1所指对象的值为42，则执行\*(p++)或\*p++运算后，p所指对象的值为\_\_\_\_\_\_\_\_。

121．假定a是一个一维指针数组，则a+i所指对象的地址比a大\_\_\_\_\_\_\_\_字节。

122. 假定a是一个一维数组，则a[i]的指针访问方式为\_\_\_\_\_\_\_\_。

123. 假定a是一个一维数组，则a[i]对应的存储地址（以字节为单位）为\_\_\_\_\_\_\_\_\_\_\_\_。

124. 一个数组的数组名实际上是指向该数组\_\_\_\_\_\_\_\_元素的指针，并且在任何时候都不允许\_\_\_\_\_\_\_\_它。

125. 假定指向一维数组b[10]中元素b[4]的指针为p，则p+3所指向的元素为\_\_\_\_\_\_\_，p-2所指向的元素为\_\_\_\_\_\_\_\_。

126．若要定义整型指针p并初始指向x，则所使用的定义语句为\_\_\_\_\_\_\_\_。

127. 若p指向x，则\_\_\_\_\_\_\_\_与x的表示是等价的。

128. 在一个二维数组int a[m][n]中，包含的一维元素a[i]的类型为\_\_\_\_\_\_\_\_，访问a[i]时返回值的类型为\_\_\_\_\_\_\_\_。

129. 假定一个二维数组为c[5][8]，则c[3]的值为二维元素\_\_\_\_\_\_\_\_的地址，c[3]+2的值为二维元素\_\_\_\_\_\_\_\_的地址。

130. 假定p为指向二维数组int d[4][6]的指针，则p的类型为\_\_\_\_\_\_\_\_。

131. 假定a是一个二维数组，则a[i][j]的指针访问方式为\_\_\_\_\_\_\_\_\_\_\_\_。

132. 若要把整型变量y定义为x的引用，则所使用的定义语句为\_\_\_\_\_\_\_\_。

133. 若y是x的引用，则对y的操作就是对\_\_\_\_\_\_\_的操作。

134. 若y是x的引用，则y和x的值\_\_\_\_\_\_\_\_，即为变量\_\_\_\_\_\_\_\_的地址。

135. 执行int p=new int操作得到的一个动态分配的整型对象为\_\_\_\_\_\_\_\_。  
    136. 执行int \*p=new int[10]操作，使p指向动态分配的数组中下标为0的元素，该元素可表示为\_\_\_\_\_\_\_或\_\_\_\_\_\_\_\_。

137. 执行char \*p=new char(’a’)操作后，p所指向的数据对象的值为\_\_\_\_\_\_\_\_。

138. 执行new char[m][n]操作时的返回值的类型为\_\_\_\_\_\_\_\_。

139. 执行\_\_\_\_\_\_\_\_操作将释放由p所指向的动态分配的数据空间。

140. 执行\_\_\_\_\_\_\_\_操作将释放由p所指向的动态分配的数组空间。

141. NULL是一个符号常量，通常作为空指针值，它的具体值为\_\_\_\_\_\_\_\_。

142. 变量v定义为“double v=23.4;”，要使指针pv指向v，则定义pv的语句为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

143. 已知语句“cout<<p;”的输出是“Hello!”，则语句“cout<<\*p;”输出的是\_\_\_\_\_\_\_\_\_\_。

144. 已知语句“cout<<s;”的输出是“apple”，则执行语句“cout<<s+2;”的输出结果为\_\_\_\_\_\_\_\_\_\_。

145. 指针变量pv和pc定义为“void \*pv =”Hello, word!”; char \*pc;”，要将pv值赋给pc，则正确的赋值语句是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

145. 数组a定义为“int a[100];”，要使p[j] 与a[j] 等效，则指针p应定义为\_\_\_\_\_\_\_\_\_\_\_\_\_\_。int \*p=a;  (或int \*p=a[0];)

146. 数组b定义为“int b[20][100];”，要使p[j][k] 与b[j][k] 等效，则指针p应定义为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

147. 与结构成员访问表达式p->name等价的表达式是\_\_\_\_\_\_\_\_\_\_\_\_。

148. 与结构成员访问表达式(\*fp).score等价的表达式是\_\_\_\_\_\_\_\_\_\_\_\_。

149. 已知变量a定义为“int a=5;”，要使ra成为a的引用，则ra应定义为\_\_\_\_\_\_\_\_\_\_\_\_。

150. 已知有定义“int x, a[]={5,7,9}, \*pa=a;”，在执行“x=++\*pa;”语句后，x的值是\_\_\_\_\_\_\_\_。

151．已知有定义“int x, a[]={6,10,12}, \*pa=a;”，在执行“x=\*++pa;”语句后，\*pa的值是\_\_\_\_\_\_\_\_。

152. 已知有定义“int x, a[]={15,17,19}, \*pa=a;”，在执行“x=\*pa++;”后，\*pa的值是\_\_\_\_\_\_\_\_。

153. 以面向对象方法构造的系统，其基本单位是\_\_\_\_\_\_\_\_\_\_。

154. 每个对象都是所属类的一个\_\_\_\_\_\_\_\_\_\_。

155. 对象将其大部分实现细节隐藏起来，这种机制称为\_\_\_\_\_\_\_\_\_\_。

156. 基类和派生类的关系称为\_\_\_\_\_\_\_\_\_\_。

157. 复杂对象可以由简单对象构成，这种现象称为\_\_\_\_\_\_\_\_\_\_。

158. 对象是对问题域中客观事物的\_\_\_\_\_\_\_\_，它是一组属性和在这些属性上操作的\_\_\_\_\_\_\_\_\_\_。

159. 特殊类的对象拥有其一般类的全部属性与操作，称特殊类\_\_\_\_\_\_\_\_了一般类。

160. 如果一个派生类的基类不止一个，则这种继承称为\_\_\_\_\_\_\_\_\_\_\_\_。

161. 如果一个派生类只有一个唯一的基类，则这样的继承关系称为\_\_\_\_\_\_\_\_\_\_。

162. C++支持两种多态性：\_\_\_\_\_\_\_\_\_时的多态性和\_\_\_\_\_\_\_\_\_\_时的多态性。

163. 在C++中，编译时的多态性是通过\_\_\_\_\_\_\_\_\_\_实现的，而运行时的多态性则是通过\_\_\_\_\_\_\_\_\_\_实现的。

164. 面向对象软件开发的生命周期分为三个阶段，即分析、\_\_\_\_\_\_\_\_\_\_和\_\_\_\_\_\_\_\_\_\_。

165. 面向对象的分析包括\_\_\_\_\_\_\_\_\_\_分析和\_\_\_\_\_\_\_\_\_\_分析两步。

166. 类定义中，既包含数据成员，也包含\_\_\_\_\_\_\_\_成员。

167. 类中的数据成员的访问属性通常被指明为\_\_\_\_\_\_\_\_。

168. 类中的供外部调用定义的函数成员，其访问属性通常被定义为\_\_\_\_\_\_\_\_。

169. 对于类中定义的任何成员，其隐含访问权限为\_\_\_\_\_\_\_\_。

170. 对于结构中定义的任何成员，其隐含访问权限为\_\_\_\_\_\_\_\_。

171. 为了使类中的成员不能被类外的函数通过成员操作符访问，则应把该成员的访问权限定义为\_\_\_\_\_\_\_\_。

172. 若在类的定义体中给出了一个成员函数的完整定义，则该函数属于\_\_\_\_\_\_\_\_函数。

173. 若在类的定义体中只给出了一个成员函数的原型，则在类外给出完整定义时，其函数名前必须加上\_\_\_\_\_\_\_\_和两个冒号分隔符。

174. 若在类的定义体中只给出了一个成员函数的原型，则在类外给出完整定义时，其函数名前必须加上类名和两个\_\_\_\_\_\_\_\_分隔符。

175. 若要把类外定义的成员函数规定为内联函数，则必须把\_\_\_\_\_\_\_\_关键字放到函数原型或函数头的前面。

176. 把一个类的定义体和所有成员函数的定义体所构成的程序范围叫做该类的\_\_\_\_\_\_\_\_\_\_\_\_。

177. 假定AA是一个类，“AA\* abc();”是该类中一个成员函数的原型，则在类外定义时的函数头为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

178. 成员函数的参数表在类作用域中，成员函数的返回值类型\_\_\_\_\_\_\_\_类作用域中。

179. 为了避免在调用成员函数时修改对象中的任何数据成员，则应在定义该成员函数时，在函数头的后面加上\_\_\_\_\_\_\_\_关键字。

180. 若只需要通过一个成员函数读取数据成员的值，而不需要修改它，则应在函数头的后面加上\_\_\_\_\_\_\_\_关键字。

181. 若采用x.abc(y)表达式调用一个成员函数，在成员函数中使用的\_\_\_\_\_\_\_\_就代表了类外的x对象。

182. 若采用p->abc(y)表达式调用一个成员函数，在成员函数中使用的\_\_\_\_\_\_\_\_就代表了类外的p指针。

183. 内联函数的定义模块与\_\_\_\_\_\_\_\_模块必须放在同一个文件中。  
    184．假定AA是一个类，“AA\* abc()const;”是该类中一个成员函数的原型，在该函数体中（能够/不能够）\_\_\_\_\_\_\_\_\_\_向\*this或其成员赋值。  
    185．在一个用数组实现的队列类中，包含有两个数据成员，一个指明队首元素位置，另一个指明\_\_\_\_\_\_\_\_\_\_\_\_。

186. 在一个用数组实现的队列类中，包含有两个数据成员，一个指明队列长度，另一个指明\_\_\_\_\_\_\_\_\_\_元素的位置。

187. 在一个用数组实现的队列类中，假定数组长度为MS，队首元素位置为first，队列长度为length，则插入一个新元素的位置为\_\_\_\_\_\_\_\_\_\_。

188. 在一个用数组实现的队列类中，假定数组长度为MS，队首元素位置为first，队列长度为length，则删除一个元素后队首的位置为\_\_\_\_\_\_\_\_\_\_。

189. 在一个用数组实现的队列类中，假定数组长度为MS，队首元素位置为first，队列长度为length，则队列为空的条件为\_\_\_\_\_\_\_\_\_\_。

190. 在一个用数组实现的队列类中，假定数组长度为MS，队首元素位置为first，队列长度为length，则队列为满的条件为\_\_\_\_\_\_\_\_\_\_。

191. 当一个队列为空时，不能对其做\_\_\_\_\_\_\_\_元素的操作。

192. 当一个队列为满时，不能对其做\_\_\_\_\_\_\_\_元素的操作。

193．从一个队列中删除元素就是删除\_\_\_\_\_\_\_\_位置上的元素。

194．向一个队列中插入元素就是把该元素放到\_\_\_\_\_\_\_\_元素的后一位置上。

195．在一个用链表实现的队列类中，假定每个结点包含的值域用elem表示，包含的指针域用next表示，链队的队首指针用elemHead表示，队尾指针用elemTail表示，当链队非空时，\_\_\_\_\_\_\_\_\_\_\_\_\_指向队首结点的后继（即下一个）结点。

196. 在一个用链表实现的队列类中，假定每个结点包含的值域用elem表示，包含的指针域用next表示，链队的队首指针用elemHead表示，队尾指针用elemTail表示，当链队非空时，新插入结点的地址应当赋给\_\_\_\_\_\_\_\_\_\_\_所指结点的next域。

197. 在一个用链表实现的队列类中，队尾结点的指针域的值为\_\_\_\_\_\_\_\_。

198. 在一个用链表实现的队列类中，若链队中只含有一个结点，则队首指针的值与队尾指针的值\_\_\_\_\_\_\_\_。

199. 在一个用链表实现的队列类中，若链队为空或只含有一个结点，则队首指针的值与队尾指针的值\_\_\_\_\_\_\_\_。

200. 在一个用链表实现的队列类中，若队首指针与队尾指针的值不同，则说明链队中至少包含有\_\_\_\_\_\_\_\_个结点。

201．一个类的\_\_\_\_\_\_\_\_\_\_函数实现对该类对象的初始化功能。

202．一个类的\_\_\_\_\_\_\_\_\_\_函数通常用于实现释放该类对象中指针成员所指向的动态存储空间的任务。

203．当用户为一个类定义有\_\_\_\_\_\_\_\_\_\_\_\_时，则系统不会为该类再自动生成一个默认构造函数。

204．假定用户没有给一个名为AB的类定义构造函数，则系统为其定义的构造函数为\_\_\_\_\_\_\_\_\_\_。

205．假定用户没有给一个名为AB的类定义析构函数，则系统为其定义的析构函数为\_\_\_\_\_\_\_\_\_\_。

206．定义类动态对象数组时，其元素只能靠自动调用该类的\_\_\_\_\_\_\_\_\_\_\_\_来进行初始化。

207．在一个类中定义拷贝构造函数的目的，是为了当利用该类的一个对象初始化另一个对象时，能够避免这两个对象的同一指针同时指向同一块\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

208．为了释放类对象中指针成员所指向的动态存储空间，则需要为该类定义\_\_\_\_\_\_\_\_\_\_\_\_。

209. 假定 AB为一个类，则执行“AB a[10];”语句时，系统自动调用该类构造函数的次数为\_\_\_\_\_\_\_\_。

210．假定一个类对象数组为A[N]，当离开它的作用域时，系统自动调用该类析构函数的次数为\_\_\_\_\_\_\_\_。

211．对类中对象成员的初始化是通过在构造函数中给出的\_\_\_\_\_\_\_\_\_\_\_\_\_\_来实现的。

212．对类中常量成员的初始化是通过在构造函数中给出的\_\_\_\_\_\_\_\_\_\_\_\_\_\_来实现的。

213．对类中引用成员的初始化只能通过在构造函数中给出的\_\_\_\_\_\_\_\_\_\_\_\_\_\_来实现。

214．对类中一般数据成员的初始化既可以通过在构造函数中给出的初始化表来实现，也可以通过构造函数中的\_\_\_\_\_\_\_\_\_\_\_\_来实现。

215. 假定要把aa定义为AB类中的一个常量整数型数据成员，则定义语句为\_\_\_\_\_\_\_\_\_\_\_\_。

216. 假定要把aa定义为AB类中的一个引用整数型数据成员，则定义语句为\_\_\_\_\_\_\_\_\_\_\_\_。

217. 假定AB类中只包含一个整型数据成员a，并且它是一个常量成员，若利用参数aa对其进行初始化，则该类的构造函数的定义为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

218. 假定AB类中只包含一个整型数据成员a，并且它是一个引用成员，若利用引用参数aa对其进行初始化，则该类的构造函数的定义为\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

219．假定指针p指向一个动态分配的类对象，则当执行“delete p;”语句时，在释放p所指向的动态存储空间之前将自动调用该类的\_\_\_\_\_\_\_\_\_\_\_\_。

220．假定用户为类AB定义了一个构造函数“AB(int aa) {a=aa;}”，则系统（会/不会）\_\_\_\_\_\_\_\_\_为该类自动定义一个无参构造函数"AB() {}"。

221. 假定用户为类AB定义了一个构造函数“AB(int aa, char \*bb=NULL):a(aa),b(bb){}”，则该类中至少包含有\_\_\_\_\_\_\_\_\_\_个数据成员。

222．假定用户为类AB定义了一个构造函数“AB(int aa) {a=aa;}”，该构造函数实现对数据成员\_\_\_\_\_\_\_\_\_的初始化。

223．假定用户为类AB定义了一个构造函数“AB(int aa=0):a(aa){}”，则定义该类的对象时，可以有\_\_\_\_\_\_\_\_种不同的定义格式。

224．假定用户为类AB定义了一个构造函数“AB(int aa):a(aa){}”，则定义该类的对象时，有\_\_\_\_\_\_\_\_种定义格式。

225．假定用户只为类AB定义了一个构造函数“AB(int aa, int bb=0) {a=aa; b=bb;}”，则定义该类的对象时，其实参表中至多带有\_\_\_\_\_\_\_\_\_个实参。

226．假定用户只为类AB定义了一个构造函数“AB(int aa, int bb=0) {a=aa; b=bb;}”，则定义该类的对象时，其实参表中至少带有\_\_\_\_\_\_\_\_\_个实参。

227．假定用户为类AB定义了一个构造函数“AB(int aa=0, int bb=0) {a=aa; b=bb;}”，则定义该类的对象时，可以有\_\_\_\_\_\_\_\_种不同的定义格式。

228．假定用户只为类AB定义了一个构造函数“AB():a(0),b(0){}”，则定义该类对象x的定义语句“AB x();”是\_\_\_\_\_\_\_\_\_(正确/错误)的。

229．假定用户只为类AB定义了一个构造函数“AB():a(0),b(0){}”，则定义该类对象x的定义语句“AB x;”是\_\_\_\_\_\_\_\_\_(正确/错误)的。

230．假定用户只为类AB定义了一个构造函数“AB():a(0),b(0){}”，则定义该类对象x的定义语句“AB x(5);”是\_\_\_\_\_\_\_\_\_(正确/错误)的。

231．假定AB为一个类，则类定义体中的“AB(AB x);”语句为该类\_\_\_\_\_\_\_\_\_\_\_\_\_\_的原型语句。

232．假定AB为一个类，则该类的拷贝构造函数的函数头为\_\_\_\_\_\_\_\_\_\_\_\_\_。

233．假定AB为一个类，该类中含有一个指向动态数组空间的指针成员pa，则在该类的析构函数中应该包含有一条\_\_\_\_\_\_\_\_\_\_\_\_\_\_语句。

234. 静态成员函数\_\_\_\_访问类的静态数据成员， \_\_\_\_访问类的非静态数据成员。

235. 静态数据成员必须在所有函数的定义体外进行\_\_\_\_\_\_\_\_\_\_。

236. 一个类的成员函数也可以成为另一个类的友元函数，这时的友元说明必须在函数名前加上\_\_\_\_\_\_\_\_\_\_的限定。

237. 重载运算符时，该运算符的\_\_\_\_\_\_\_\_\_\_、结合性以及操作符的个数不允许改变。

238. 一个单目运算符作为类的成员函数重载时有\_\_\_\_\_\_\_\_个参数；如果作为独立函数重载，则有\_\_\_\_\_\_\_\_个参数。

239. 一个双单目运算符作为类的成员函数重载时有\_\_\_\_\_\_\_\_个参数；如果作为独立函数重载，则有\_\_\_\_\_\_\_\_个参数。

240. 除了\_\_\_\_\_\_\_\_\_\_运算符外，其他重载的运算符都可以被派生类继承。

241. 作为类的成员函数重载一个运算符时，参数表中只有一个参数，说明该运算符有\_\_\_\_\_\_\_\_个操作数。

242. 在重载一个单目运算符时，参数表中没有参数，说明该运算符函数只能是类的\_\_\_\_\_\_\_\_\_\_\_\_\_。

243. 重载插入运算符<<时，其运算符函数的返回值类型应当是\_\_\_\_\_\_\_\_\_\_\_\_。

244. 重载抽取运算符>>时，其运算符函数的返回值类型应当是\_\_\_\_\_\_\_\_\_\_\_\_。

245. 重载插入运算符<<或抽取运算符>>时，其运算符函数的参数有\_\_\_\_\_\_\_\_个。

246. 重载插入运算符<<或抽取运算符>>时，其运算符函数不能被定义为类的\_\_\_\_\_\_\_\_函数。

247. 类型转换函数没有\_\_\_\_\_\_\_\_\_\_\_\_类型，而且参数表为\_\_\_\_\_\_\_\_\_\_\_\_。

248. 在一个或若干个类的基础上构造一个新类，被称为\_\_\_\_\_\_\_\_\_\_\_。

249. 派生类的成员函数可以直接访问基类的\_\_\_\_\_\_\_\_\_\_成员，不能直接访问基类的 \_\_\_\_\_\_成员。

250. 当保护继承时，基类的\_\_\_\_\_\_\_\_\_\_成员在派生类中成为保护成员，派生类对象不能直接访问基类的\_\_\_\_\_\_\_\_\_\_成员。

251. 在定义一个派生类时，使用\_\_\_\_\_\_\_\_\_\_关键字或者不显式地使用它则表示为\_\_\_\_\_\_\_\_\_\_继承。

252. 若多个基类及其派生类中都定义了同名函数成员，要访问相应函数时，就需要在函数名前加上\_\_\_\_\_\_和类区分符。

253. 若要保证一个公共的基类在派生类中只产生一个基类子对象，则必须都以\_\_\_\_\_\_\_\_\_\_\_\_的方式直接继承它。

254. 引进虚基类的根本目的是为了消除\_\_\_\_\_\_\_\_\_\_\_\_。

255. 在每个成员函数中，隐含的第一个参数的参数名为\_\_\_\_\_\_\_\_。