KMP算法，即Knuth-Morris-Pratt算法，是一种用于字符串匹配的有效算法。其中的next数组在求解KMP算法过程中发挥着非常重要的作用。下面将介绍如何以C++98标准写一份KMP算法中求解next数组的函数。

1.函数原型：

void get\_next(const char \*p, int next[]);

2.函数参数说明：

p：匹配模式串

next[]：保存每个字符对应的next值的数组

3.函数实现：

void get\_next(const char \*p, int next[])

{

int len = strlen(p);

int j = 0;

next[0] = 0;

for (int i = 1; i < len; i++)

{

while (j > 0 && p[i] != p[j])

{

j = next[j - 1];

}

if (p[i] == p[j])

{

j++;

}

next[i] = j;

}

}

4.函数说明：

该函数主要是采用了KMP算法中的核心思想，即通过next数组来实现匹配模式串时的跳跃。其基本思路是先设置j=0，然后一次比较p[i]和p[j]所对应的字符，如果相等，则设置next[i]=j+1；否则，利用next[j-1]的值，重新设定j的值。其实现过程就是在一个while循环中不断地更新j的值，直到找到相等的字符或者j的值为0。最后将当前的j值保存到next[i]数组中。在循环结束时，得到了一个完整的next数组。

5.函数测试：

为了测试get\_next函数是否正确实现，可以输入一些需要匹配的字符串，然后通过计算其对应的next数组来判断程序是否能够正确处理。具体代码如下：

int main()

{

char p[30];

int next[30];

while (1)

{

printf("请输入一个模式串：");

scanf("%s", p);

get\_next(p, next);

printf("next数组为：\n");

for (int i = 0; i < strlen(p); i++)

{

printf("%d ", next[i]);

}

printf("\n");

}

return 0;

}

6.总结：

KMP算法是一种高效的字符串匹配算法，其求解过程中next数组的作用非常重要。本文就是以C++98标准为基础，介绍了如何编写一份求解next数组的函数。当然，这只是KMP算法中的一个小步骤，实际应用中还需要去实现匹配的过程。

KMP算法是字符串匹配算法的一种实现方式，它的全称是Knuth-Morris-Pratt算法，主要用于在一个文本串S内查找一个模式串P的出现位置。该算法由Donald Knuth、Vaughan Pratt和James Morris三位计算机科学家于1977年共同提出。

KMP算法的核心思想是尽可能地利用已经匹配过的信息，通过对前面匹配失败的子串进行分析来确定下一步匹配的位置，从而减少匹配次数，提高匹配效率。具体实现可以采用类似动态规划的方式，在匹配失败时判断前缀和后缀的相似程度，从而确定下一步匹配的位置。

下面是KMP算法的具体实现流程：

1. 预处理模式串P，构建一个next数组。具体实现过程中，先定义一个next[0]=-1，令j=0，i=1，然后从头开始逐个比较P[j]和P[i]，如果匹配成功，则将j和i同时向后移动一位；如果匹配失败，则需要回溯到前一个next值的位置，即令j等于next[j]，继续比较P[j]和P[i]，直到找到一个next值，使得P[j]与P[i]匹配成功或者j回溯到了0位置。

2. 在文本串S中查找模式串P。具体实现过程中，定义两个指针i和j，i指向文本串S的开头位置，j指向模式串P的开头位置，然后从头开始顺序比较S[i]和P[j]，如果匹配成功，则i和j同时向后移动一位；如果匹配失败，则需要根据next数组回溯到前面已经匹配过的位置，即令j等于next[j]，继续比较S[i]和P[j]。

3. 如果在匹配过程中j回溯到了0位置，则说明当前匹配失败，需要将i和j都向后移动一位，从下一个字符重新开始匹配。

4. 当匹配完成时，如果j等于模式串长度，则说明完全匹配成功，返回i-j即为模式串在文本串中出现的位置。

KMP算法的时间复杂度是O(m+n)，其中m为文本串长度，n为模式串长度。因此，在匹配较大文本串和模式串时，KMP算法具有较高的效率和可扩展性。