

C库协同开发——实验报告

一、项目要求

- 1、用C语言为某个基因组的genbank文件写两个动态共享库libgenbank.so、libfasta.so。
genbank.h: 解读genbank文件，提取CDS序列。
fasta.h: 将提取的序列保存为FASTA格式的文件。
- 2、为库和应用程序编写Makefile，或用autotools完成这一个过程。
- 3、在github中建立账号，并将库程序和对应的提取工具程序通过git上传到建立的repository中

二、genbank 和 FASTA文件格式分析

1、genbank格式

对genbank文件格式的理解参考了该网站：<https://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html>

(1) CDS:

1. 一个genbank文件中可能有多个CDS1
2. 编码起点到终点有3中表示方式：
 - 完整的: 2345...5823
 - 从5'开始: <1...240
 - 结束在3': 234...6928>
3. complement表示反向互补链是编码部分
4. join(b1...e1,b2...e2)表示将多个exon连接起来

(2) 基因序列:

位于origin到//之间

2、FASTA格式

对fasta文件格式的理解参考了该网站:<https://phaster.ca/input>

FASTA格式中的一条完整序列，包括开头的单行描述符和多行序列数据。

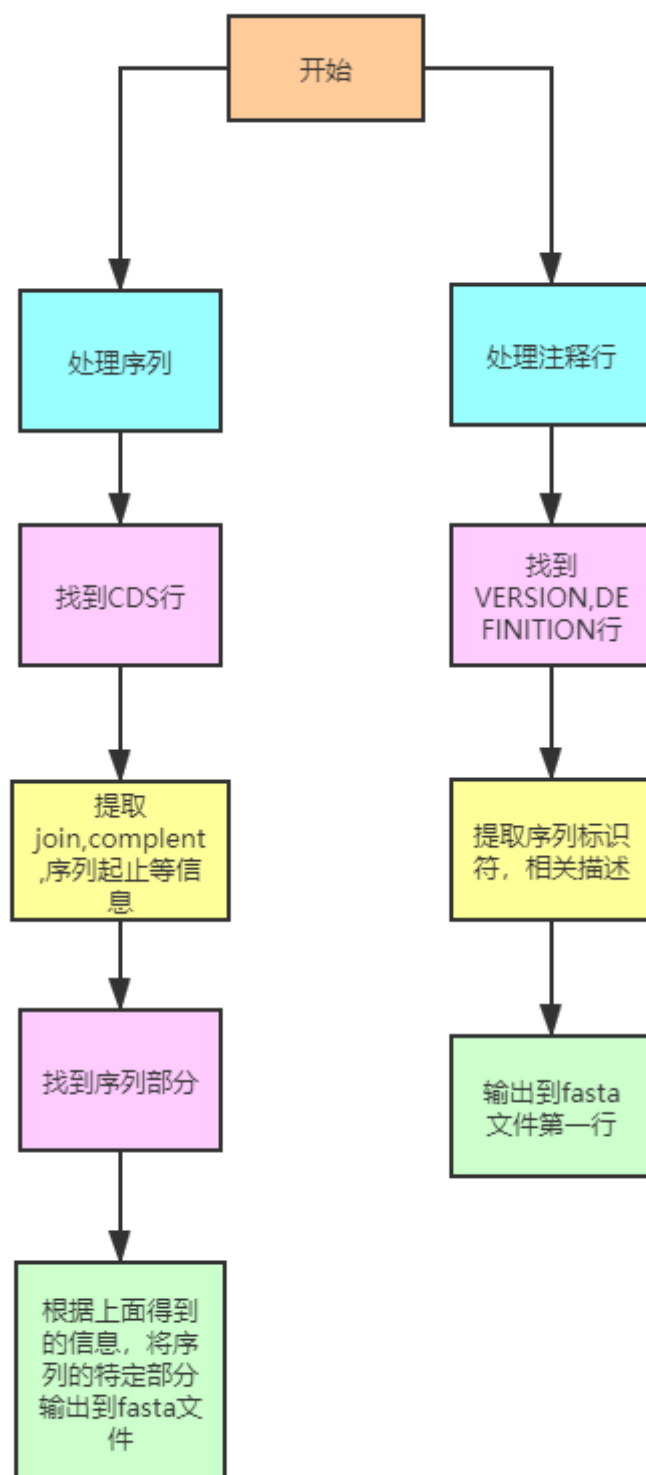
(1) 描述行行首前置半角大于号(">")以和数据行区分。">"后进阶的内容为该序列的标识符，剩余部分为序列的描述。标识符和描述均非必须。

(2) 标识符可从genbank文件VERSION行获得，描述信息可从DEFINITION行获得。

(3) 序列单行限制在80字符以内。

三、程序的实现

1、程序设计思想



2、核心代码的解析

(1)提取CDS序列

```
#define MAXCOL 5000
#define SECTION 100

typedef struct {
    char **str;        //the PChar of string array
    size_t num;        //the number of string
}IString;

char buf[MAXCOL];
char line[MAXCOL];
IString subline;
IString data;
IString location;

char *data_join[SECTION] = {NULL};        //存储join的起始终止位置信息
char *data_complement[SECTION] = {NULL};  //存储complement的起始和终止位置信息
int num_join = 0;                          //需要正向连接的区间个数
int num_complement = 0;                    //互补链的区间个数
char defination[MAXCOL];                   //存储genbank的defination的内容
char note_fasta[MAXCOL];                   //存储fasta文件注释行输入的内容
char complement_output[MAXCOL];
int num_array_complement = 0;
int *section_start_c;
int *section_end_c;
int *section_start_j;
int *section_end_j;

int split(char *src, char *delim, IString* istr);        //分割函数
//void Reverse(char s[]);                                //倒置函数
void tackle_join(FILE *fp, FILE *target);                //处理join类型的连接
void tackle_complement(FILE *fp, FILE *target);          //处理complement类型的连接
void get_sequence(int begin, int end, FILE *fp, int* i, FILE *target);        //获取区
//间的母链序列
void get_sequence_p(int begin, int end, FILE *fp, int *i, FILE *target);        //获取区
//间的互补链序列
void reverse_basic(char basic);                          //互补链，转置字符
void get_CDS(char name[], FILE *target);                  //处理genbank的CDS内容
void get_note(FILE *fp, FILE *target);                    //处理genbank需要输入到
//fasta文件的注释行内容
void output(FILE *fp, FILE *target);

void get_note(FILE *fp, FILE *target)
{
    int i = 0;
    note_fasta[i++] = '>';
    int num_def = 0;
    while(fgets(line, MAXCOL, fp))
    {
        split(line, " ", &subline);
        if (strcmp(subline.str[0], "DEFINITION") == 0){        //把defination的
//数据拷贝下来
            for (int z = 1; z < subline.num; ++z){
                defination[num_def++] = ' ';
                strcpy(defination + num_def, subline.str[z]);
            }
        }
    }
}
```

```

        num_def += strlen(subline.str[z]);
    }
    free(subline.str);
}

else if (strcmp(subline.str[0], "VERSION") == 0){           //处理version
    for (int j = 1; j < subline.num; ++j){
        if (subline.str[j][0] == 'G'){                     //处理GI
            note_fasta[i++] = 'g';
            note_fasta[i++] = 'i';
            note_fasta[i++] = '|';
            strcpy(note_fasta + i, subline.str[j] + 3);
            i += strlen(subline.str[j] + 3);
            if (subline.str[j][strlen(subline.str[j]) - 1] == '\n')
note_fasta[i - 1] = '|'; //防止最后一个字符为\n破坏格式
            else note_fasta[i++] = '|';
        }
        else {                                             //其他的引用情况
            note_fasta[i++] = 'r';
            note_fasta[i++] = 'e';
            note_fasta[i++] = 'f';
            note_fasta[i++] = '|';
            strcpy(note_fasta + i, subline.str[j]);
            i += strlen(subline.str[j]);
            if (subline.str[j][strlen(subline.str[j]) - 1] == '\n')
note_fasta[i - 1] = '|';
            else note_fasta[i++] = '|';
        }
    }
}

else if (strcmp(subline.str[0], "DBLINK") == 0)
    break;
}
strcpy(note_fasta + i, defination);
writeFasta_note(note_fasta, target);
}

void get_CDS(char name[], FILE *target)                     //进行CDS预处理
{
    FILE *fp = fopen(name, "r");
    get_note(fp, target);
    while(fgets(line, MAXCOL, fp))                         //读取文件直至CDS那一行
    {
        split(line, " ", &subline);
        if (strcmp(subline.str[0], "CDS") == 0){
            split(subline.str[1], "<>", &data);           //划分出编码的方式以及编码的起始
终止位置，存在data
//由于\n的存在，因此，data.num
应该减一
            if (data.str[0][0] >= '0' && data.str[0][0] <= '9'){
                data_join[num_join++] = data.str[0];
            }
            else
            {
                for (int i = 0; i < data.num - 1; i++)
                {
                    if (data.str[i][0] == 'j'){           //处理CDS的join
                        i++;

```

```

        while (i < data.num - 1 && data.str[i][0] >= '0' &&
data.str[i][0] <= '9')
        {
            data_join[num_join++] = data.str[i++];
        }
    }
    else if (data.str[i][0] == 'c'){           //处理CDS的
complement
        i++;
        if (data.str[i][0] == 'j') i++;
        while (i < data.num - 1 && data.str[i][0] >= '0' &&
data.str[i][0] <= '9')
        {
            data_complement[num_complement++] = data.str[i++];
        }
    }
}

    }
    else if(strstr(subline.str[0],"ORIGIN")){
        free(subline.str);
        break;
    }

}
if (num_join != 0)
    tackle_join(fp,target);
if (num_complement != 0)
    tackle_complement(fp,target);

output(fp, target);
char end[] = "//";
writeFasta_note(end,target);
fclose(fp);
}

void output(FILE *fp,FILE *target)
{
    char sequence[] = "1";           //定位到序列行
    while(fgets(line,MAXCOL,fp))
    {
        split(line,"",&subline);
        if (strcmp(subline.str[0],sequence) == 0)
            break;
        else free(subline.str);
    }
    int line_num_ = 0;                //序列行号
    int i = 0,j = 0;
    while (i < num_join && j < num_complement){
        if (section_start_j[i] < section_start_c[j]){
            get_sequence(section_start_j[i], section_end_j[i], fp, &line_num_,
target);
            i++;
        }
        else{
            get_sequence_p(section_start_c[j], section_end_c[j], fp, &line_num_,
target);

```

```

        j++;
    }

}

if (i < num_join) {
    for(; i < num_join; ++i) get_sequence(section_start_j[i],
section_end_j[i], fp, &line_num_, target);
}
if (j < num_complement){
    for (; j < num_complement; ++j)
        get_sequence_p(section_start_c[j], section_end_c[j], fp, &line_num_,
target);
}

writeFasta_seq('\n',target);
}

void tackle_complement(FILE *fp, FILE *target)    //处理complement
{
    section_start_c = (int *)malloc(sizeof(int) * num_complement);
    section_end_c = (int *)malloc(sizeof(int) * num_complement);
    int j = 0;
    for (int i = 0; i < num_complement; i++)    //把起始终止的片段存储在数组中
    {
        split(data_complement[i], ".", &location);
        section_start_c[j] = atoi(location.str[0]);
        section_end_c[j++] = atoi(location.str[1]);
        free(location.str);
    }
}

void tackle_join(FILE *fp, FILE *target)    //处理join
{
    section_start_j = (int *)malloc(sizeof(int) * num_join);
    section_end_j = (int *)malloc(sizeof(int) * num_join);
    int j = 0;
    for (int i = 0; i < num_join; i++)    //把起始终止的片段存储在数组中
    {
        split(data_join[i], ".", &location);
        section_start_j[j] = atoi(location.str[0]);
        section_end_j[j++] = atoi(location.str[1]);
        free(location.str);
    }
}

void get_sequence(int begin, int end, FILE *fp, int* i, FILE *target)    //处理join序
列读取
{
    int line_begin = (begin - 1) / 60;
    int line_end = (end - 1) / 60;
    int col_begin = (begin - 1) % 60;
    int col_end = (end - 1) % 60;
    int z;
    do //序列读取
    {
        if ((*i) == line_begin){
            split(line, " ", &subline); //读到最后

```

```

        int subcol = col_begin / 10 + 1; //获取字符串所在的subline的位置
        int col_start = col_begin % 10; //获取列数
        for (;col_start < 10;++col_start)
        {
            writeFasta_seq(subline.str[subcol][col_start],target);
        }
        subcol++;
        for (;subcol < 7;++subcol)
        {
            for (z = 0;z < 10;++z)
                writeFasta_seq(subline.str[subcol][z],target);
        }
        free(subline.str);
    }

    else if ((*i) > line_begin && (*i) < line_end){ //输出整行
        int subcol = 1;
        split(line, " ",&subline); //读到最后
        for (;subcol < 7;++subcol)
        {
            for (z = 0;z < 10;++z)
                writeFasta_seq(subline.str[subcol][z],target);
        }
        free(subline.str);
    }

    else if ((*i) == line_end){ //输出到col_end
        split(line, " ",&subline);
        int subcol = col_end / 10 + 1;
        int col_over = col_end % 10;

        int col = 1;
        for (;col < subcol;++col)
        {
            for (z = 0;z < 10;++z)
                writeFasta_seq(subline.str[col][z],target);
        }
        int k = 0;
        for (;k <= col_over;++k)
        {
            writeFasta_seq(subline.str[subcol][k],target);
        }
        free(subline.str);
        break;
    }

    (*i)++;
}while(fgets(line,MAXCOL,fp));
}

void get_sequence_p(int begin,int end,FILE *fp,int *i,FILE *target) //处理
complement序列读取
{
    int line_begin = (begin - 1) / 60;
    int line_end = (end - 1) / 60;
    int col_begin = (begin - 1) % 60;
    int col_end = (end - 1) % 60;
    int z; //计数因子

```

```

do //序列读取
{
    if ((*i) == line_begin){
        num_array_complement = 0;
        split(line, " ", &subline); //读到最后
        int subcol = col_begin / 10 + 1; //获取字串所在的subline的位置
        int col_start = col_begin % 10; //获取列数
        for (; col_start < 10; ++col_start)
        {
            reverse_basic(subline.str[subcol][col_start]);
        }
        subcol++;
        for (; subcol < 7; ++subcol)
        {
            for (z = 0; z < 10; ++z)
                reverse_basic(subline.str[subcol][z]);
        }
        //complement_sequence[realnum++] = '\n'; //使整个数组中只有序列部分
        free(subline.str);
    }

    else if ((*i) > line_begin && (*i) < line_end){ //输出整行
        int subcol = 1;
        split(line, " ", &subline); //读到最后
        for (; subcol < 7; ++subcol)
        {
            for (z = 0; z < 10; ++z)
                reverse_basic(subline.str[subcol][z]);
        }
        free(subline.str);
    }

    else if ((*i) == line_end){ //输出到col_end
        split(line, " ", &subline);
        int subcol = col_end / 10 + 1;
        int col_over = col_end % 10;

        int col = 1;
        for (; col < subcol; ++col)
        {
            for (z = 0; z < 10; ++z)
                reverse_basic(subline.str[col][z]);
        }
        int k = 0;
        for (; k <= col_over; ++k)
        {
            reverse_basic(subline.str[subcol][k]);
        }
        free(subline.str);

        for (int i = num_array_complement - 1; i >= 0; --i)
            writeFasta_seq(complement_output[i], target);
        break;
    }
    (*i)++;
}while(fgets(line, MAXCOL, fp));
}

void reverse_basic(char basic)

```



```

{
    if (basic == 'a') complement_output[num_array_complement++] = 't';
    else if (basic == 't') complement_output[num_array_complement++] = 'a';
    else if (basic == 'c') complement_output[num_array_complement++] = 'g';
    else if (basic == 'g') complement_output[num_array_complement++] = 'c';
}
int split(char *src, char *delim, IString* istr)    //split src, 将分割后的子串存入结构体的str中
{
    int i;
    char *str = NULL, *p = NULL;

    (*istr).num = 1;
    str = (char*)calloc(strlen(src)+1,sizeof(char));
    if (str == NULL) return 0;
    (*istr).str = (char**)calloc(1,sizeof(char *));
    if ((*istr).str == NULL) return 0;
    strcpy(str,src);

    p = strtok(str, delim);
    (*istr).str[0] = (char*)calloc(strlen(p)+1,sizeof(char));
    if ((*istr).str[0] == NULL) return 0;
    strcpy((*istr).str[0],p);
    for(i=1; p = strtok(NULL, delim); i++)
    {
        (*istr).num++;
        (*istr).str = (char**)realloc((*istr).str,(i+1)*sizeof(char *));
        if ((*istr).str == NULL) return 0;
        (*istr).str[i] = (char*)calloc(strlen(p)+1,sizeof(char));
        if ((*istr).str[i] == NULL) return 0;
        strcpy((*istr).str[i],p);
    }
    free(str);
    str = p = NULL;

    return 1;
}

```

(2)保存为fasta格式文件

```

//fasta中一行序列的长度取70
int currentcol = 0;

void writeFasta_seq(int c,FILE *fp);    //写入fasta文件的序列部分
void writeFasta_note(char note[],FILE *fp);

void writeFasta_seq(int c,FILE *fp)
{
    if (currentcol < 70){    //如果序列长度 + 原有的小于70, 直接输入
        putc(c,fp);
        currentcol += 1; //记录该行的长度
    }
    else{//如果已经超过70就输出到下一行
        putc('\n',fp);    //光标下一行
    }
}

```

```

        putc(c, fp);
        currentcol = 1; //长度重新设为1
    }
}

void writeFasta_note(char note[], FILE *fp) //自带换行符，输入注释信息
{
    fputs(note, fp);
}

```

3、debug

1. 在debug的过程中，出现最多的问题就是输不出结果
 - 首先利用printf确保每个模块是否能够执行。
 - 在确定模块全部正确执行后，查看两个输出函数 get_sequence 和 get_sequence_p 是否有问题。
 - 在确保上述两个函数没有问题后，确定每一段区间的起始和终止是否正确。在本程序中，问题就是首先从这里开始的，即区间的起始和终止位点个数不正确或者根本没有。这里采用的方式就是printf输出每个区间的起始和终止位置，从而发现了问题。
 - 发现区间有问题，首先回到处理区间的函数，tackle_complement与tackle_join，发现这两个函数仅仅是对输入的字符串进行拆分与整形处理，不会有大问题。
 - 所以，问题只可能出现在处理genbank的CDS序列模块。发现，原来的程序，为了处理CDS位于不同行的情况（当然序列还是要在同一行的，本程序在这一点做了简化），采用了while循环，但是break的位置设置的并不对，因而造成了那一行之后的CDS序列无法读取。
2. debug过程中还出现了注释行输出不到一行、输出函数output有问题等情况，但是这些情况都是小问题，按照上述思路很轻松就可以解决。
3. 我们的测试文件采用的是eg1.txt, eg2.txt和linux.txt,发现比较复杂的eg1.txt和eg2.txt跑出了正确的结果，这说明我们的代码本身是没有问题的。但是比较简单的linux.txt却跑出了没有序列的空文件。为了查找原因，我们把跑出的fasta文件从linux拷贝到windows后，发现格式有差别。

```

liuxinyi — root@85da6d850f33:/home/bio/bi296/lab6/lab6 — docker • rundocker.sh — 91x...
>ref|NC_000086.7| Mus musculus strain C57BL/6J chromosome X, GRCh38.p6 C57BL/6J.
atgctgctgctggcagacatggacgtggatcagctgggtggctgggggtcagttccgggtggtcaagg
agcccttggtctcgtgaaggtgctgcagtggtctttgccatcttcgcctttgctacgtgcggcagcta
caccggagagcttcggctgagcgtggagtgtgccaacaagacggagagtgccctcaacatcgaagtcgaa
tttgagtacccattcaggctgcaccaagtgtactttgatgcaccctcctgcgttaaagggggcactacca
agatcttcttagttggtgactactcctcctcggtgaattctttgtcaccgtggctgtgtttgccttct
ctactccatggggccctggccacgtacatcttctcgcagaacaagtaccgagagaacaacaagggccca
atgatggacttcctggccacagcagtggttcgtttcatgtggctagttagctcatccgcctgggccaag
gcctgtccgatgtgaagatggccactgaccagagaaacattatcaaggagatgcctatgtgccgcagac
aggaaacacatgcaaggaaactgaggaccctgtgacttcaggactcaacacctcggtggtgtttggcttc
ctgaacctggtgctctgggttggcaacctatggttcgtgttcaaggagacaggctgggcccgcattca
tgcgcgcacctccaggcgccccagaaaagcaaccagctcctggcgatgcctacggcgatgcgggctatgg
gcagggcccgaggctatgggcccaggactcctacgggcctcaggggtggttatcaaccgattacggg
cagccagccagcggtggtggtggtacgggcctcagggcgactatgggcagcaaggctacggccaac
agggtgcgccacctcttctccaatcagatgtaa
//
sequence.fasta (END)

```

```

>ref|NC_000086.7
| Mus musculus strain C57BL/6J chromosome X, GRCh38.p6 C57BL/6J.
atgtctgtctgtggcagacatggacgtggtgaatcagctgggtggctgggggtcagttccgggtggtcaagg
agcccccttggtcttctgaaggtgctgcagtggtgctttgccatcttcgccttgctacgtcggcgagcta
caccggagagcttcggctgagcgtggagtggtccaacaagacggagagtgccctcaacatcgaagtcgaa
tttgagtacccattcaggctgcaccaagtgtactttgatgcacctcctgcgttaagggggactacca
agatcttcttagttggtgactactcctcctcggtgaattctttgtcaccgtggctgtgtttgccttct
ctactccatgggggccctggccacctacatcttctcgcagaacaagtaccgagagaacaacaaggcca
atgatggacttccctggccacagcagtggttcgcttcatgtggctagttagctcatccgcctgggcaaaag
gcctgtccgatgtgaagatggccactgaccagagaacattatcaaggagatgcctatgtgccgccagac
aggaacacatgcaaggaaactgagggaacctgtgacttcaggactcaacacctcggtggtgtttggcttc
ctgaacctggtgctctgggttggaacctatggttcgtgttcaaggagacaggctgggcccgcattca
tgcgcgacacctcaggcgccccagaaaagcaaccagctcctggcgatgcctacggcgatcgggctatgg
gcaggggccccggaggctatgggccccaggactcctacgggcctcagggtggttatcaaccgattacggg
cagccagccagcggcggtggcggtggtacgggcctcagggcgactatgggcagcaaggctacggccaac
agggtgcgcccacctccttctccaatcagatgtaa
//

```

因此推测linux.txt从windows拷贝到linux的过程中格式发生了错乱，从而导致了其格式已经不符合genbank的要求了，所以输出了没有序列的文件。

查找资料后，发现windows和linux换行符不一样。windows下是\n, \r,在linux下是^M\$。在写程序时查找序 列的部分用的是查找origin跟换行符，这样在linux下是不适用的。

```

Linux.txt  X
zhan_lian > Linux.txt
20 /organism= Absidia glauca
21 /mol_type="genomic DNA"
22 /strain="CBS 101.48"
23 /sub_strain="RVII-324 met-"
24 /culture_collection="CBS:101.48"
25 /db_xref="taxon:4829"
26 /mating_type="minus"
27 complement(<9..1783)
28 gene
29 CDS
30 /gene="ABSL_05122.1 scaffold 6499"
31 join(201..310,321..450,complement(join(8214..8503,8701..8900)))
32 /gene="ABSL_05122.1 scaffold 6499"
33 /codon_start=1
34 /product="hypothetical protein, partial"
35 /protein_id="SAL99496.1"
36 /translation="MEKQPNNNMEVDVENSYQSDVARATHSLDVLKKRAVDAQGKAD
37 EALAADAPEEEYALMDVYNKKWELYQRTSNFAARFPEEGVFRGKASGGPNQGSNK
38 NPVAAPALKYVDLPFLASARERASGNRALVTQNVREFATAFEALMELHQLNINDVYQR
39 YLPICLGKYYKTFIYSKRTLGETNIETWPMVKWGLVEFTNTPRQKVKNTTAWMELTP
40 GSDTGEDF FHRVREFKEAHELANTADALLFFAVFTNCRFGWRNKISEAIRDTHQPF
41 VENFFEECAFASDLELNAGKPDADRDHNRSTSSQRTNRKRSAADNNHYGNRTW
42 TNNINASTPRRMKGTGPGYGGGGYCDNGCGEKFMPPHKGVCPAIINRETNDRRSNEDR
43 RDSKRHQSEPDQHQRRQDPVSRASAEYIDQVRADDVERLQSTFRGTTLDDDEDL
44 PCKLGQKSEGNTPIILLEHINVPLYIENKRTLALVDSGANFSSINKNFCTEHNVPIL
45 PHKKESNILLANAGISIKSYGYTPITIKYNGSYTQCQLEVMIDLALGRMTSVWFEPST
46 DAVYLQTPLKRFISIFSNAAVKLSQ"
CONTIG join(FKIZ01004832.1:1..3226)

```

```

sequence_Linux.fasta
1 >ref|LT552780.1|gi|16445223| Absidia glauca strain CBS 101.48 genome assembly, sca
2 ggctttggccctgaaccctgagagggtggtggcgaggggagcagagatacggctcgtggagctggggctg
3 gggctggggctggggctaggggtctgcagcaggcaagggggtggagaaaaggagtgctcctctttgccga
4 ggaaggctgtcctcactgccgcagagggggcctccagatgccccaaaagaactgtcgacctggaggggg
5 aaaagagagattcgggtctaggaagagggttaagtatttttaaaagtagtcttttaagtgtctaggtttac
6 aggcataatattaacatccttggctcctaagtatatgcttttcttttcttttcttttatttttttaga
7 cagggtctcactctgcagccctggttgacctggaactctttttgtacaccaggctggcctcaaactctca
8 gagatccaccagctctctgcctcccaagtgtggaattaaaggcatgtgctaccatgtgccagctaagta
9 tatgttatttttttaaaacaattaaaaattacaaataaataagtgtgggattaaagggttctaggactt
10 gaacgcctgtcctcaggaagagggttaagtgtcttaaccatggagtcacccctctagccgctcttttc
11 ttttcttttttaaatagatttttattgttttgtttgttttgagacggctcttgcttttatccaaggatg
12 acctggagtttactacctaggccaatctgg
13 //

```

之后在windows下运行该测试文件linux.txt发现结果是正确的，印证了上述猜想。
为了消除这个bug,在代码中将定位ORIGIN的部分进行了更改，改用了strstr。

四、Makefile

Makefile文件内容：

```
obj = GENBANK_FASTA

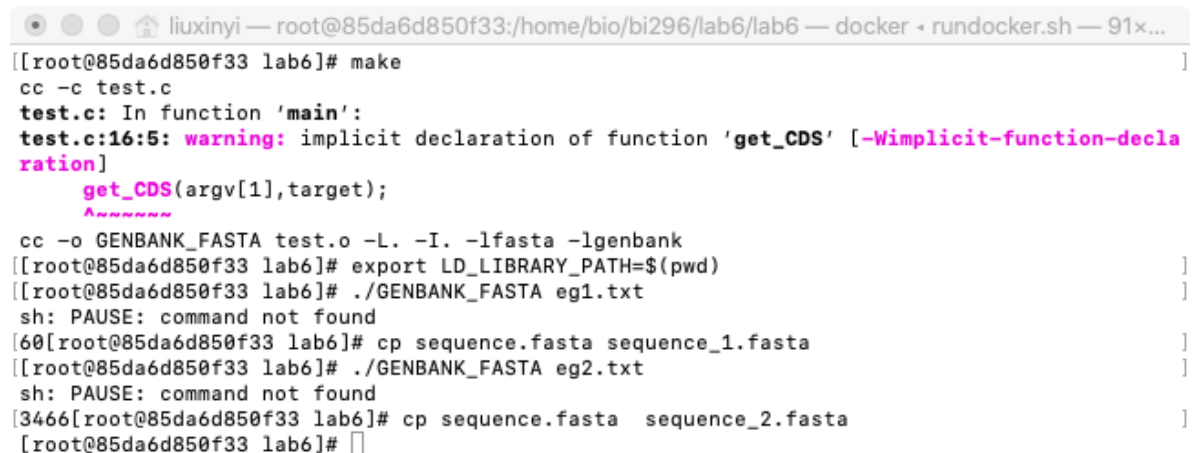
$(obj) : test.o
    cc -o GENBANK_FASTA test.o -L. -I. -lfasta -lgenbank

test.o : test.c
    cc -c test.c

clean :
    rm GENBANK_FASTA test.o
```

五、运行测试

编译过程截图：



```
liuxinyi — root@85da6d850f33:/home/bio/bi296/lab6/lab6 — docker • rundocker.sh — 91x...
[[root@85da6d850f33 lab6]# make
cc -c test.c
test.c: In function 'main':
test.c:16:5: warning: implicit declaration of function 'get_CDS' [-Wimplicit-function-declaration]
    get_CDS(argv[1],target);
    ^~~~~~
cc -o GENBANK_FASTA test.o -L. -I. -lfasta -lgenbank
[[root@85da6d850f33 lab6]# export LD_LIBRARY_PATH=$(pwd)
[[root@85da6d850f33 lab6]# ./GENBANK_FASTA eg1.txt
sh: PAUSE: command not found
[60][root@85da6d850f33 lab6]# cp sequence.fasta sequence_1.fasta
[[root@85da6d850f33 lab6]# ./GENBANK_FASTA eg2.txt
sh: PAUSE: command not found
[3466][root@85da6d850f33 lab6]# cp sequence.fasta sequence_2.fasta
[[root@85da6d850f33 lab6]# ]
```

测试文件及运行结果文件见附录：

测试文件：eg1.txt , eg2.txt, Linux.txt

运行结果：eg1.fasta, eg2.fasta, Linux.fasta

六、结果讨论

1、支持的情况

(1)程序支持线性linear的序列,暂不支持环形circular序列。

(2)在序列的处理中,通过查找ORIGIN定位序列时,支持ORIGIN直接加回车, ORIGIN前后有任意个空格再加回车。

(3)我们的程序支持大部分常规写法。

- 支持文件中有多个CDS的情况

但对于CDS的序列信息位于多行的情况,需要手动调整至一行再进行处理,如:

```
CDS   join(123..234,245..267,345..399,404..487,  
          555..578,613..699,876,..987,1000..2876)
```

- 支持最简单的无嵌套情况,如:

```
CDS   2341..7623
```

- 支持仅有complement或仅有join的情况,如:

```
CDS   complement(123..765)
```

```
CDS   join(234..544,786..998)
```

- 支持complement和join的嵌套:

```
CDS   complement(join(233..425,876,999))
```

大多genbank的文件都遵守后面的序列数比前面大的规则

```
CDS   join(220..350,6200..6500,complement(2300..3600))
```

```
CDS   join(200..300,450..600,complement(join(750..800,890..1100)))
```

如果前面的序列数字比后面的大,会造成输出顺序错乱(这种格式的文件很少见)

```
CDS   join(6200..6500, 220..350,complement(2300..3600))
```

2、备注

(1)我们的算法中, join(complement())主要用于先连接顺序,再连接逆序。

```
join(200..300,complement(500..800))
```

(2)join只起连接作用。complement之后都是互补链,如果还要从母链开始,需要再写一个CDS

(3)join(200..300,450..600,complement(join(750..800,890..1100)))join(complement(),complement())

这种写法主要用于连接非顺序的序列。但是,由于我们只考虑线性的序列,因此对于这种写法,前面的complement起始位置需要比后面的complement小。其实也就等价于complement(join())。

(4)complement要逆序输出，所以在写程序时将complement中的放到一个数组中。我们的程序设置的数组大小是5000，所以如果genbank中出现了1..6000这样的情况就不行。如果想比较灵活的处理的话，可以用链表。

(5)join和complement分割的字段，100..200,300..400,字段个数不能超过100个。