**STRING.H EMULADA**

***void \*mi\_memchr(const void \*s, int c, size\_t n)***{  
    char \* ps = s;  
    int cont = 0;  
    while((\*ps) && (\*ps != c) && (cont < n))  
    {  
        ps++;  
        cont++;  
    }  
    if ((\*ps) && (cont < n))  
    {  
        return ps;  
    }  
    return NULL;  
  
}  
  
***int mi\_memcmp(const void \*s1, const void \*s2, size\_t n)***  
{  
    int cont = 1;  
    char \*p1 = s1,  
         \*p2 = s2;  
    while ((\*p1) &&(\*p2) && (cont < n) &&(\*p1 == \*p2))  
    {  
        p1++;  
        p2++;  
        cont++;  
    }  
    if((!\*p1) && (\*p2) && (cont<n))  
        return (-1);  
    if((\*p1) && (!\*p2) && (cont<n))  
        return (1);  
    if (\*p1 == \*p2)  
        return 0;  
    if(\*p1 > \*p2)  
    {  
        return 1;  
    }  
    else  
        return -1;  
}

***char \* mi\_strcat(char \* s1, const char \* s2)***  
{  
    char \*retorno = s1;  
    while (\*s1)  
        s1++;  
    while (\*s2)  
    {  
        \*s1 = \*s2;  
        s1++;  
        s2++;  
    }  
    \*s1 = '\0';  
    return retorno;  
}

***char \* mi\_strchr(const char \* s1, const char c)***  
{  
    char \* p = s1;  
    while ((\*p) && (\*p != c))  
        p++;  
    if(!\*p)  
        return NULL;  
    return p;  
  
}  
  
***int mi\_strcmp(const char \*s1, const char \*s2)***{  
    char \* p1 = s1,  
         \* p2 = s2;  
    while ((\*p1) && (\*p2) && (\*p1 == \*p2))  
    {  
        p1++;  
        p2++;  
    }  
    if ((!\*p1) && (!\*p2))  
        return 0;  
    if(!\*p1)  
        return -1;  
    if(!\*p2)  
        return 1;  
    return (\*p1 - \*p2);  
}

***char \* mi\_strcpy(char \* s1, const char \* s2)***  
{  
    char \*p1 = s1,  
         \*p2 = s2;  
    while (\*p2)  
    {  
       \*p1 = \*p2;  
        p1++;  
        p2++;  
    }  
    \*p1 = '\0';  
    return s1;  
}  
  
***size\_t mi\_strlen(const char \* s1)***  
{  
    int cont = 0;  
    char \* p1 = s1;  
    while(\*p1)  
    {  
        cont++;  
        p1++;  
    }  
    return (size\_t)cont;  
}  
  
***size\_t mi\_strcspn(const char \*s1, const char \*s2)***  
{  
    char \* p1 = s1,  
         \* p2 = s2;  
    int cont = 0;  
    while (\*p1)  
    {  
        while (\*p2)  
        {  
            if (\*p2 == \*p1)  
                return (size\_t)cont;  
            p2++;  
        }  
        p2 = s2;  
        cont++;  
        p1++;  
    }  
    return (size\_t)cont;  
}

***char \*mi\_strncat(char\*s1, const char \*s2, size\_t n)***  
{  
    char \* destino = s1,  
         \* origen = s2;  
    int i;  
    while(\*destino)  
        destino++;  
    for(i=0; i<n; i++)  
    {  
        \*destino = \*origen;  
        destino++;  
        origen++;  
    }  
    \*destino = '\0';  
    return s1;  
}  
  
  
***int mi\_strncmp(const char \*s1, const char \*s2, size\_t n)***  
{  
    char \* p1 = s1,  
         \* p2 = s2;  
    int i = 0;  
    while ((\*p1) && (\*p2) && (i < n) && (\*p1 == \*p2))  
    {  
        p1++;  
        p2++;  
        i++;  
    }  
    if ((!\*p1) && (!\*p2))  
        return 0;  
    if(!\*p1)  
        return -1;  
    if(!\*p2)  
        return 1;  
    return (\*p1 - \*p2);  
}  
  
***char \*mi\_strncpy(char \*s1, const char \*s2, size\_t n)***  
{  
    char \*p1 = s1,  
         \*p2 = s2;  
    int i = 0;  
    while (\*p2 && i < n)  
    {  
       \*p1 = \*p2;  
        p1++;  
        p2++;  
        i++;  
    }  
    \*p1 = '\0';  
    return s1;  
}

***char \*mi\_strpbrk(const char \*s1, const char \*s2)***  
{  
    char \* cad1 = s1,  
         \* cad2 = s2;  
    while(\*cad1)  
    {  
        while((\*cad1) && (\*cad2) && (\*cad1 != \*cad2))  
            cad2++;  
        if (\*cad2 == \*cad1)  
            return cad1;  
        cad2 = s2;  
        cad1++;  
    }  
    return NULL;  
}  
  
***char \*mi\_strrchr(char \*s, int c)***  
{  
    char \* cad = s;  
    while (\*cad)  
        cad++;  
    cad--;  
    while ((cad != s) && (\*cad != c))  
        cad--;  
    if (\*cad == c)  
        return cad;  
    return NULL;  
}  
  
***size\_t mi\_strspn(const char \*s1, const char \*s2)***{  
    char \* origen = s1,  
         \* sub = s2;  
    int cont = 0;  
    while(\*origen)  
    {  
        while((\*sub) && (\*origen != \*sub))  
            sub++;  
        if(!\*sub)  
            return cont;  
        cont++;  
        sub = s2;  
        origen++;  
    }  
    return cont;  
}

***int myAtoi(const char\*num)***

{

    int entero=0,signo=1;

    if(num==NULL)

        return 0;

    while(\*num&&\*num==' ')

        num++;

    if(\*num=='-'){

        signo=-1;

        num++;

    }

    else if(\*num=='+')

        num++;

    while(ISNUM(\*num))

    {

            entero\*=10;

            entero+=(\*num-'0');

            num++;

    }

    return entero\*signo;

***char \*mi\_strstr(const char \*s1, const char \*s2)***{  
    char \* cad = s1,  
         \* sub = s2,  
         \* aux = NULL;  
    while(\*cad)  
    {  
        while((\*cad) && (\*sub) && (\*sub != \*cad))  
            cad++;  
        if (\*cad)  
            aux = cad;  
        while((\*cad) && (\*sub) && (\*cad == \*sub))  
        {  
            sub++;  
            cad++;  
        }  
        if (!\*sub)  
            return aux;  
        sub = s2;  
        aux = NULL;  
    }  
    return aux;  
}

***double myAtof(const char\*num)***

{

    double val=0;

    double pos=1;

    int signo=1;

    if(num==NULL)

        return 0;

    while(\*num&&\*num==' ')

        num++;

    if(\*num=='-'){

        signo=-1;

        num++;

    }

    else if(\*num=='+')

        num++;

    while(ISNUM(\*num))

    {

        val\*=10;

        val+=(\*num-'0');

        num++;

    }

    if(tolower(\*num)=='e')

        return val\*pow(10,myAtoi(++num));

    else if(\*num!='.')

      return val\*signo;//\*/\*num==' '?val:val\*signo;

    num++;

    while(ISNUM(\*num))

    {

        val+=(\*num-'0')\*(pos/=10);

        num++;

    }

    if(tolower(\*num)=='e')

        return  val\*pow(10,myAtoi(++num));

    return val\*signo;

}

**PARTE DEL MAIN DE CONJUNTOS**

fread(&a, sizeof(int),1,fileA);

fread(&b, sizeof(int),1,fileB);

while(!feof(fileA) && !feof(fileB))

{

while((a < b) && !feof(fileA))

{

fprintf(pUnionEstricta, "%d", a);

fprintf(pUnion, "%d", a);

fprintf(pAmenosB, "%d", a);

fread(&a, sizeof(int),1,fileA);

}

while((b < a) && !feof(fileB))

{

fprintf(pUnionEstricta, "%d", b);

fprintf(pUnion, "%d", b);

fprintf(pBmenosA, "%d", b);

fread(&b, sizeof(int),1,fileB);

}

if(a == b && !feof(fileA) && !feof(fileB))

{

fprintf(pUnion, "%d", b);

fprintf(pInterseccion, "%d", b);

fread(&a, sizeof(int),1,fileA);

fread(&b, sizeof(int),1,fileB);

}

Else

{

while(!feof(fileB))

{

fprintf(pUnionEstricta, "%d", b);

fprintf(pUnion, "%d", b);

fprintf(pBmenosA, "%d", b);

fread(&b, sizeof(int),1,fileB);

}

while(!feof(fileA))

{

fprintf(pUnionEstricta, "%d", a);

fprintf(pUnion, "%d", a);

fprintf(pAmenosB, "%d", a);

fread(&a, sizeof(int),1,fileA);

}

}

}

**PRIMITIVAS COLA ESTÁTICA:**

***void crearCola(t\_cola \*p)***  
{  
    p->ult = -1;  
    p->pri = 0;  
}  
  
***int colaVacia(const t\_cola \*p)***  
{  
    return (p->ult == -1);  
}  
  
***int colaLlena(const t\_cola \*p)***  
{  
    return ((p->ult == (p->pri)-1) && (p->ult != -1) || (p->ult == TAM-1) && (p->pri == 0));  
}  
  
int acolar(t\_cola \*p, const t\_info \*d)  
{  
    if ((p->ult == (p->pri)-1) && (p->ult != -1) || (p->ult == TAM-1) && (p->pri == 0)) ///COLA LLENA  
        return COLA\_LLENA;  
    p->ult++;  
    if(p->ult == TAM)  
        p->ult = 0;  
    p->cola[p->ult] = \*d;  
    return TODO\_OK;  
}  
  
***int desacolar(t\_cola \*p, t\_info \*d)***  
{  
    if(p->ult == -1)  
        return COLA\_VACIA;  
    \*d = p->cola[p->ult];  
    p->ult--;  
    if(p->ult == -1 && p->pri != 0)  
        p->ult = TAM-1;  
    return TODO\_OK;  
}  
  
***int verTope(const t\_cola \*p, t\_info \*d)***  
{  
    if(p->ult == -1)  
        return COLA\_VACIA;  
    \*d = p->cola[p->ult];  
    return TODO\_OK;  
}  
***void vaciarCola(t\_cola \*p)***  
{  
    p->ult = -1;  
    p->pri = 0;  
}