תוכן עניינים

Huffman 3
גנריות 10
חיתוך מערכים 10
מיון מצביעים 11
איחוד מערכים 12
מיון מצביעים גרסה נוספת 13
החזרת מערך למצביעים של איברי מערך בלי חזרות 13
מציאת ערך מינמום ומקסימום במערך(דוגמא של פונקצית השוואה של פונקציות) 14
חישוב דליים 15
החזרתי מערך של תתי מערכים של מצביעים שכל תת מערך מכיל מצביעים לאיברים השווים אחד לשני 16
החזרת מערך של טווחים 17
מקבלת מספר לא ידוע של מערכים ומחזירה מערך של האיברים המינימליים של מערכי הקלט(איבר אחד מכל מערך) 18
החזרת מערך של האיברים המקסימליים של מערך לפי כמה פונקציות השוואה, מקבל מערך של פונקציות השוואה כקלט 18
החלפה כל הערך הישנים בחדשים והחזרת מערך של כתובת שבהם בוצעה החלפה 19
מבני נתונים 20
פונקציה שמקבלת עץ ומחזירה מערך של מצביעים לרשימות שכל רשימה היא מסלול אחר בעץ מהשורש לעלה אחר. 20
פונקציה שמקבלת מערך של רשימות של מסלול משורש לעלה ומחזירה עץ 22
החזרת מערך של מצביעים לרשימות שכל רשימה היא רמה אחרת בעץ 23
להפוך עץ לרשימה דו כיוונית מעגלית 24
מיון שני רשימות ממוינות לרשימה אחת (רשימה חדשה לא מוקצת בזיכרון, בשניה כן) 25
מקבלת רשימה של תוים וקידוד ההופמן שלהם ומייצר עץ הופמן 26
מקבל שתי רשימות אחת שמייצגת preoreder ואחת inorder של עץ ומחזירה את העץ 28
מקבלת מספר לא ידוע של איבר מסוג לא ידוע ומצביע לפונקצית השוואה ומחזירה מצביע לרשימה ממוינת של האיברים 30
עבודה עם קבצים 31
מקבל מערך של מצביעים לקבצים ומחזיר מערך של מצביעים למערכים כאשר מכל קובץ יצרנו מערך חדש 31
שאלה עם הקובץ של יוניקס – פתרון של אחמד 32
שאלה עם מבנה שמייצג אנשים עדכונים נתונים והוספה לקובץ 35
מחיקת הערות מקובץ מקור של תוכנית C 38
קריאת הכותרת של קובץ GZ לא נבדק 39
משחק החיים עם קבצים לא נבדק 40

פונקציה שמקבלת מצביע לקובץ עם איברים ממוינים בסדר עולה והשני בסדר יורד ומבצעת מיזוג אל תוך קובץ חדש ... 41
מקבל קובץ מקור בלי הערות וקובץ הערות ומבצע שחזור ... 42
מקבל קובץ עם 81 מספרים ובודק אם מייצגים סודוקו – לא נבדק ... 43
Read Files ... 43
מצביעים בלבד ... 45
שחלוף מטריצה משיעורי בית ... 45
שכפול מערך של מערכים בלי מקומות מיותרים ... 45
איחוי מערך של מערכים צמצום מקומות מיותרים ... 46
Mystery ... 46
משיעורי בית ... 48
משחק החיים ... 48
פעולות על ביטים ... 51
מיון מערך של מערכים של מחרוזות ומציאת המחרוזת הארוכה ביותר ... 52
הפונקציה מחזירה מערך של מבנה שמחזיק את הערך של עלה ואת הרמה שבא נמצאה ... 54
מילוי מערך ... 58
מיון הדפסה ושחרור מהזיכרון ... 59
Complicated declaratrion ... 61

Huffman

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
typedef struct {
	int x;
	int y;
} point;
typedef struct {
	char temp;
	int counter;
} Buffer;
typedef  struct {
	char chr;
	int counter;
}Symbol;
typedef  struct  HNode{
	Symbol  s;
	struct HNode *left, *right;
} HNode,*HNodePtr;
typedef struct LNode{
	HNodePtr root;
	struct LNode * next;
}LNode;
typedef struct {
	Symbol* arr;
	int length;
} Symbol_arr;
Symbol createSymbol(char c,int n){
	Symbol temp={c,n};
	return temp;
}
int getBit(int x,int p){
	unsigned int i = 1<<p;
	return x&i? 1:0;
}
char setBit(char x,int p){
	int i = 1<<p;
	return x|=i;
}
int setBitInt(int x,int p){
	int i = 1<<p;
	return x|=i;
}
HNodePtr createHNode(Symbol st,HNodePtr l,HNodePtr r){
	HNodePtr temp = (HNodePtr)malloc(sizeof(HNode));
	temp->s=st;
	temp->left=l;
	temp->right=r;
	return temp;
}
LNode* createLNode(HNodePtr st){
	LNode* temp= (LNode*)malloc(sizeof(LNode));
	temp->root=st;
```

```
	temp->next=NULL;
	return temp;
}
LNode* add(LNode* list,HNodePtr c){
	if(list==NULL)
		return createLNode(c);
	if(c->s.counter<=list->root->s.counter){
		LNode* temp = createLNode(c);
		temp->next=list;
		return temp;
	}
	else
		list->next=add(list->next,c);
	return list;
}
int Lsize(LNode* list){
	int i=0;
	LNode* temp=list;
	while(temp){
		i++;
		temp=temp->next;
	}
	return i;
}
void printList(LNode* list){
	while(list){
		printf("%c %d ",list->root->s.chr,list->root->s.counter);
		list=list->next;
	}
}
HNodePtr createTree(HNodePtr f,HNodePtr s){
	HNodePtr temp =(HNodePtr) malloc(sizeof(HNode));
	temp->s.chr=0;
	temp->s.counter=f->s.counter+s->s.counter;
	return temp;
}
HNodePtr makeHufTree(LNode* list){
	while(Lsize(list)>1){
		HNodePtr first = list->root;
		list=list->next;
		HNodePtr seccond = list->root;
		list=list->next;
		HNodePtr temp = createHNode(createSymbol(0,seccond->s.counter+first-
>s.counter),first,seccond);
		list= add(list,temp);
	}

	return list->root;
}
Symbol_arr newSymbolArr(){
	Symbol_arr temp;
	temp.arr=NULL;
	temp.length=0;
	return temp;
}
Symbol_arr  createFreq(FILE* f1){
	Symbol_arr temp = newSymbolArr();
	rewind(f1);
```

```
        int n=0;
        while((n=getc(f1))!=EOF){
                char c = (char)n;
                int   flag=0;
                for(int i=0;i<temp.length;i++)
                        if(temp.arr[i].chr==c){
                                temp.arr[i].counter++;
                                flag=1;
                        }
                        if(flag==0){

        temp.arr=(Symbol*)realloc(temp.arr,sizeof(Symbol)*(temp.length+1));
                                temp.arr[temp.length].chr=c;
                                temp.arr[temp.length].counter=1;
                                temp.length++;
                        }
        }
        return temp;
}
int isLeaf(HNodePtr root){

        if(!root->left && !root->right)
                return 1;
        else return 0;
}
int numofLeaf(HNodePtr root){
        if(!root)
                return 0;
        else if(isLeaf(root))
                return 1;
        return numofLeaf(root->left)+numofLeaf(root->right);
}
void printTree(HNodePtr root){
        if(root==NULL)
                return;
        if(isLeaf(root))
                printf("%c ",root->s.chr);
        printTree(root->left);
        printTree(root->right);
}

char* getCode(HNode* root, char c)
{
        if(isLeaf(root) && root->s.chr!=c){
                return NULL;
        }
        if(root->left->s.chr==c){
                char* temp = (char*)malloc(sizeof(char)*2);
                temp[0]='1';
                temp[1]=0;
                return temp;
        }
        if(root->right->s.chr==c){
                char* temp = (char*)malloc(sizeof(char)*2);
                temp[0]='0';
                temp[1]=0;
                return temp;
        }
```

```
        char* temp;
        if((temp=getCode(root->right,c))){
                int n=strlen(temp);
                char* temp2=(char*)malloc(sizeof(char)*(n+1));
                temp2[0]='0';
                temp2[1]=0;
                strcat(temp2,temp);
                return temp2;
        }
        if((temp=getCode(root->left,c))){
                int n=strlen(temp);
                char* temp2=(char*)malloc(sizeof(char)*(n+1));
                temp2[0]='1';
                temp2[1]=0;
                strcat(temp2,temp);
                return temp2;
        }

}
char decode(HNodePtr root,char * code){
        if(!code){
                return -1;
        }
        if(isLeaf(root)){
                return root->s.chr;
        }
        if(strcmp(code,"")==0){
                return -1;
        }
        if(*code=='1')
                return decode(root->left,++code);
        else return decode(root->right,++code);
}
int bitLen(int x){
        int length =0;
        for(int i=0;i<32;i++)
                if(getBit(x,i)==1)
                        length=i;
        return length;
}
void createCompressedFile(FILE* out,FILE* in,HNodePtr root){
        Buffer b = {0,0};
        int n;
        rewind(in);
        fseek(out,0,SEEK_END);
        while((n=getc(in))!=EOF){
                char* code = getCode(root,n);
                for(int i=0;i<strlen(code);i++){
                        if(code[i]=='1')
                                b.temp=setBit(b.temp,b.counter);
                        b.counter++;
                        if(b.counter==8){
                                fwrite(&b.temp,sizeof(char),1,out);
                                b.counter=0;
                                b.temp=0;
                        }
                }
        }
```

```
	if(b.counter!=0)
		fwrite(&b.temp,sizeof(char),1,out);

}
void readFile(FILE * in,FILE* out,HNodePtr root,int numOfChars,int sizeOfFreqAr){
	int n,n2 ;
	int k=1;
	//	fseek(in,7+sizeof(Symbol)*sizeOfFreqAr,SEEK_SET);
	char * temp=(char*)malloc(sizeof(char));
	int count=0;
	temp[0]=0;
	while((n=getc(in))!=EOF){
		for(int i=0;i<8;i++){
			temp=(char*)realloc(temp,8*k*sizeof(char));
			if(getBit(n,i))
				strcat(temp,"1");
			else
				strcat(temp,"0");
			if((n2=decode(root,temp))!=-1)
			{
				putc(n2,out);
				strcpy(temp,"");
				count++;
				if(count==numOfChars){
					free(temp);
					return;
				}
				k=1;
				continue;
			}
		}
		k++;
	}
	free(temp);
}
void createHeaderBedoreCompression(FILE* in,Symbol_arr freq,int fSize){
	putc('H',in);
	putc('S',in);
	fwrite(&fSize,sizeof(int),1,in);
	fwrite(&freq.length,sizeof(int),1,in);
	for(int i=0;i<freq.length;i++){
		fwrite(freq.arr+i,sizeof(Symbol),1,in);
	}
}
int readHeaderOrigFsize(FILE* i){
	fseek(i,3,SEEK_SET);
	int n;
	fread(&n,sizeof(int),1,i);
	return n;
}
Symbol_arr readFreqArr(FILE* i){
	fseek(i,6,SEEK_SET);
	int n;
	fread(&n,sizeof(int),1,i);
	Symbol_arr temp;
	temp.length=n;
	temp.arr=(Symbol*)calloc(n,sizeof(Symbol));
	fread(temp.arr,sizeof(Symbol),n,i);
```

```c
	return temp;
}
LNode* makeHufList(Symbol_arr a){
	LNode* list = NULL;
	for(int i=0;i<a.length;i++)
		list=add(list,createHNode(a.arr[i],NULL,NULL));
	return list;
}
int checkIfHuf(FILE * f){
	int n;
	n=getc(f);
	if(n!='H')
		return 0;
	n=getc(f);
	if(n!='S')
		return 0;
	return 1;
}
int main(int argc,char ** argv)
{
	if(argc != 3){
		fprintf(stderr,"error! invailid num of paramerters");
	}
	if(!argv[1][0]=='-' && ( !argv[1][1]=='c' || !argv[1][1]=='d' ) ){
		fprintf(stderr,"error! invailid paramerters");
		exit(EXIT_FAILURE);
	}
	else if(argv[1][1]=='c'){
		printf("Start compressing");
		FILE * toCompress = fopen(argv[2],"r+");
		if(!toCompress){
			perror("terminated cause -> ");
			exit(EXIT_FAILURE);
		}
		Symbol_arr a = createFreq(toCompress);
		LNode* list = makeHufList(a);
		HNodePtr root = makeHufTree(list);
		char * compressFileName = (char*)malloc((strlen(argv[2])+5)*sizeof(char));
		compressFileName[0]=0;
		strcpy(compressFileName,argv[2]);
		strcat(compressFileName,".huf");
		FILE * com = fopen(compressFileName,"wb+");
		int fsize;
		fseek(toCompress,0,SEEK_END);
		fsize=ftell(toCompress);
		createHeaderBedoreCompression(com,a,fsize);
		createCompressedFile(com,toCompress,root);
		int fsizecom;
		fseek(com,0,SEEK_END);
		fsizecom=ftell(com);
		printf("original file size was %d , compreessed size %d",fsize,fsizecom);
		exit(EXIT_SUCCESS);
	}
	else if(argv[1][1]=='d'){
		printf("Starting to decompress");
		FILE* compressed = fopen(argv[2],"rb+");
		if(!compressed){
			perror("treminated cause->");
```

```
				exit(EXIT_FAILURE);
			}
			if(!checkIfHuf(compressed))
			{
				fprintf(stderr,"Error not huf File");
				exit(EXIT_FAILURE);
			}
			int origsize= readHeaderOrigFsize(compressed);
			Symbol_arr a= readFreqArr(compressed);
			LNode* l = makeHufList(a);
			HNodePtr root = makeHufTree(l);
			FILE* f = fopen("decomp.txt","wb+");
			readFile(compressed,f,root,origsize,a.length);
			exit(EXIT_SUCCESS);
		}
		exit(EXIT_FAILURE);
	}
```

חיתוך מערכים

```
int gInclude(void* A,void* B,int n,int m,int (*cmp)(void*,void*)){
	for(int i=0;i<n;i++){
		if(cmp((char*)A+i*m,B)==0 )
			return 1;
	}
	return 0;
}
void* gIntersect(int n,int m,int (*cmp)(void* ,void*),void* A,...){
	char** B=(char**)malloc(sizeof(char*));
	B[0]=(char*)A;
	va_list param;
	va_start(param,A);
	int length=2;
	char* temp=NULL;
	while(temp=va_arg(param,char*)){
	B=(char**)realloc(B,sizeof(char*)*length);
	B[length-1]=temp;
	length++;
	}
	B[length-1]=NULL;
	int k=0;
	char* p=NULL;
	char **C=B;
	while(*C){
		char* D = *C;
		for(int t=0;t<n;t++){
		int flag=0;
		for(int i=0;i<length-1;i++){
			if(!gInclude(B[i],D,n,m,cmp)){
				flag=1;
				break;
			}
		}
			if(!flag){
				if(gInclude(p,D,k,m,cmp)==0){
					k++;
					p=(char*)realloc(p,m*k);
					memcpy(p+(k-1)*m,D,m);
				}
			}
		D=D+m;
		}
		C++;
	}
	return (void*)p;
}
```

מיון מצביעים

```
int genericFindingIndexOfMaxElement(void* A,int size,int typeSize, int (*fP)( void*,
void*)){
	int max=0;
	for(int i=1;i<size;i++){
		if(fP((char*)A+max*typeSize,(char*)A+typeSize*i)<0)
			max = i ;
	}
	return max;
}

void** gSort(void* base,int n,int size,int (*cp)(void*,void*)){
	int k=0;
	char** A = (char**)calloc(n,sizeof(char*));
	for(int i=0;i<n;i++){
		int count=0;
		for(int j=0;j<n;j++){
			if(cp((char*)base+i*size,(char*)base+j*size)>0)
				count++;
		}
		A[count]=(char*)base+i*size;
	}
	return (void**)A;
}
```

```
void**  gUnion (int n,int m,  int  (*cmp) (void *, void*), void *base,... ){
	int length =0;
	char** A=NULL;
	for(int i=0;i<n;i++){
		int	flag=1;
		for(int j=0;j<length;j++){
			if(cmp((char*)base+i*m,*(A+j))==0)
				flag=0;
		}
		if(flag){
			length++;
			A=(char**)realloc(A,length*(sizeof(char*)));
			A[length-1]=(char*)base+i*m;
		}
	}
	va_list param;
	va_start(param,base);
	char *temp=NULL;
	while(temp=va_arg(param,char*)){
		for(int i=0;i<n;i++){
			int	flag=1;
			for(int j=0;j<length;j++){
				if(cmp((char*)temp+i*m,*(A+j))==0)
					flag=0;
			}
			if(flag){
				length++;
				A=(char**)realloc(A,length*(sizeof(char*)));
				A[length-1]=(char*)temp+i*m;	}
		}
	}
	length++;
	A=(char**)realloc(A,length*(sizeof(char*)));
	A[length-1]=NULL;
	return (void**)A;

}
```

מיון מצביעים גרסה נוספת

```
typedef  struct  {
  void   *elm;
  int       rnk;
}Rank;
Rank* gRank(void* first,void * last,int sizeOfElement,int (*cp)(void*,void*)){
       int numOfElement = (((char*)last-(char*)first)/sizeOfElement)+1;

       Rank* A = (Rank*)calloc(numOfElement,sizeof(Rank));
       for(int i=0;i<numOfElement;i++){
              int count=0;
              for(int j=0;j<numOfElement;j++){
                     if(cp((char*)first+j*sizeOfElement,(char*)first+i*sizeOfElement)<0)
                     count++;
              }
              Rank temp;
              temp.elm=(char*)first+i*sizeOfElement;
              temp.rnk=count;
              A[i]=temp;
       }
       return A;
}
```

החזרת מערך למצביעים של איברי מערך בלי חזרות

```
void** gUniq(void* first,void* last,int sizeOfField,int cmp(void*,void*)){
int temp =1;
char** A;
int k=0;
A= (char**)malloc(sizeof(char*));
A[k]=(char*)first;
k++;
       for(int i=1;(char*)first+i*sizeOfField<=last;i++){
              int flag=0;
              for(int j=i-1;j>=0;j--)
              {
                     if(cmp((char*)first+i*sizeOfField,(char*)first+j*sizeOfField)==0)
                            flag=1;
              }
              if(!flag){
                     A=(char**)realloc(A,(k+1)*sizeof(char*));
                     A[k] = (char*)first+sizeOfField*i;
                     k++;
              }
       }
       return (void**)A;
}
```

מציאת ערך מינמום ומקסימום במערך(דוגמא של פונקצית השוואה של פונקציות)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef  struct {
  void * min ;
  void *max ;
} MinMax;
int   f()  {return 1;}
int   g() {return 2 ;}
int   h()  {return 3;}
int functionCmp(const void* f,const void* f2){
       int n = (*(int (**)())f)();
       int n2 = (*(int (**)())f2)();
       return n-n2;
}
int genericStrCmp(const void*  P1,const void*  P2){
       return strcmpi((char*)P1,(char*)P2);
}
int genericStrPointerCmp(const void*   P1,const void*  P2){
       return strcmpi(*(char**)P1,*(char**)P2);
}
int genericFindingIndexOFMaxElement(void* A,int size,int typeSize, int (*fP)(const void*,
const void*)){
       int max=0;
       for(int i=1;i<size;i++){
       if(fP((char*)A+max*typeSize,(char*)A+typeSize*i)<0)
              max = i ;
       }
       return max;
}

int genericFindingIndexOFMinElement(void* A,int size,int typeSize, int (*fP)(const void*,
const void*)){
       int max=0;
       for(int i=1;i<size;i++){
       if(fP((char*)A+max*typeSize,(char*)A+typeSize*i)>0)
              max = i ;
       }
       return max;
}

MinMax   gMinMax (void *A, int size, int sizeOfField, int  (*cp) (const void  *, const
void  *) ){
       int min = genericFindingIndexOFMinElement(A,size,sizeOfField,cp);
       int max = genericFindingIndexOFMaxElement(A,size,sizeOfField,cp);
       MinMax b;
       b.max=(char*)A+max*sizeOfField;
       b.min=(char*)A+min*sizeOfField;
       return b;
}
int main(void){
       char    a[][10] = {"xyz","abc","aaaa"};
char   *b[] = {"xyz","abc","aaaa","xyz"};
Circle   c[] = { {1,2,1}, {3,4,7} ,{7,1,3}};
MinMax str1 = gMinMax(a,sizeof(a)/sizeof(a[0]),sizeof(a[0]),genericStrCmp);
 int   (*d[]) () = { f ,g , h };
```

```
 MinMax fnc2 = gMinMax(d,sizeof(d)/sizeof(d[0]),sizeof(d[0]),functionCmp);
 printf("%d %d",(*(int (**)())fnc2.min)(),(*(int (**)())fnc2.max)());

}
```

חישוב דליים

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
typedef  struct {
  void * elm;
  int  counter;
} Felm;
int genericStrCmp(void*  P1, void*  P2){
       return strcmpi((char*)P1,(char*)P2);
}
Felm*   calcFreq (void * A, int length, int SizeOfElem, int  (cp) (void *, void*) ){
       int count = 0;
       for ( int i = 0; i < length ; i++ ){
              int flag = 0;
              for(int j=i-1;j>=0;j--){
                     if(cp((char*)A+SizeOfElem*i,(char*)A+SizeOfElem*j)==0)
                            flag=1;
                     }
              if(!flag)
                     count++;
       }
       Felm * B = (Felm*)calloc(count,sizeof(Felm));
       count=0;
       for ( int i = 0; i < length ; i++ ){
              int flag2=0;
              for(int j=0;j<count;j++)
                     if(cp((char*)A+SizeOfElem*i,B[j].elm)==0){
                            B[j].counter++;
                            break;
                     }
                     B[count].elm=(char*)A+i*SizeOfElem;
                     B[++count].counter=1;
       }
       return B;
}
int main(void){
       char    a[][10] = {"xyz","abc","aaaa"};
       Felm* B = calcFreq(a,sizeof(a)/sizeof(a[0]),sizeof(a[0]),genericStrCmp);
}
```

החזרתי מערך של תתי מערכים של מצביעים שכל תת מערך מכיל מצביעים לאיברים השווים אחד לשני

```
void*** gEquals (void *base, int n, int  size, int  (*cmp) (void *, void*) ){
	char*** A = (char***)malloc(sizeof(char**));
	int length=0;
	int count=0;
	A[0]=NULL;
	for(int i=0;i<n;i++){
		int flag=0;
		for(int k=0;k<length;k++)
			if(cmp((char*)base+size*i,**(A+k))==0){
				flag=1;
				break;
			}
			if(!flag){
				count=1;
				for(int j=i+1;j<n;j++)
					if(cmp((char*)base+i*size,(char*)base+j*size)==0)
						count++;
				A[length]=(char**)calloc(count+1,sizeof(char*));
				int temp=0;
				for(int j=i;i<n;j++){
					if(cmp((char*)base+i*size,(char*)base+j*size)==0){
						A[length][temp]=(char*)base+j*size;
						temp++;
						A[length][temp]=NULL;
						if(temp==count)break;
					}
				}
				A[length][temp]=NULL;
				length++;
				A=(char***)realloc(A,sizeof(char**)*(length+1));
			}
	}
	A[length]=NULL;
	return (void***)A;
}
```

```
typedef  struct {
	int  from, to;
} Range;
int checkOkev(void* a,void*b){
	if(*(int*)a+1==*(int*)b)
		return 1;
	else
		return 0;
}
Range newRange(int f,int t){
	Range temp;
	temp.from=f;
	temp.to=t;
	return temp;
}
Range* gRange (void *first, void *last, int size , int  (*check) (void *, void*) ){
	int start=0;
	int flag=0;
	int length=1;
	Range* Arr = (Range*) malloc(sizeof(Range));
	Arr[length-1]=newRange(-1,-1);
	int i;
	for(i=0;(char*)first+i*size<(char*)last;i++){
		if(check((char*)first+i*size,(char*)first+(i+1)*size)){
			flag++;
		}
		else
		{
			Arr[length-1]=newRange(start,i);
			length++;
			Arr=(Range*)realloc(Arr,sizeof(Range)*(length));
			Arr[length-1]=newRange(-1,-1);
			start=i+1;
		}
	}
	Arr[length-1]=newRange(start,i);
	length++;
	Arr=(Range*)realloc(Arr,sizeof(Range)*(length));
	Arr[length-1]=newRange(-1,-1);
	start=i+1;

	return Arr;
}
```

מקבלת מספר לא ידוע של מערכים ומחזירה מערך של האיברים המינימליים של מערכי הקלט(איבר אחד מכל מערך)

```
void*  gMin (int length,int size,  int  (*cmp) (void *, void*),...){
       int sizeArr = 0;
       char* Answer = NULL;
       va_list param;
       va_start(param,cmp);
       while(char* temp =  va_arg(param,char*)){
              sizeArr++;
              Answer=(char*)realloc(Answer,size*sizeArr);
              int idx=genericFindingIndexOfMinElement(temp,length,size,cmp);
              memcpy(Answer+(sizeArr-1)*size,temp+idx*size,size);
       }
       va_end(param);
       return Answer;
}
```

החזרת מערך של האיברים המקסימליים של מערך לפי כמה פונקציות השוואה, מקבל מערך של פונקציות השוואה כקלט

```
int genericFindingIndexOFMaxElement(void* A,int size,int typeSize, int (*fP)( void*,
void*)){
       int max=0;
       for(int i=1;i<size;i++){
              if(fP((char*)A+max*typeSize,(char*)A+typeSize*i)<0)
                     max = i ;
       }
       return max;
}

maxElm*   gAllMax(void *base , int n, int size , int  (*cmp[])(void *, void *)){
       int count=0;
       int (**temp)(void*,void*)=cmp;
       for(;*temp;count++,temp++);
       maxElm* A = (maxElm*)calloc(count+1,sizeof(maxElm));
       int n2=0;
       for(int i=0;i<count;i++){
              n2= genericFindingIndexOFMaxElement(base,n,size,cmp[i]);
              A[i].elm=(char*)base+n2*size;
              A[i].ptr=cmp[i];
       }
       return A;
}
```

החלפה כל הערך הישנים בחדשים והחזרת מערך של כתובת שבהם בוצעה החלפה

```
void**  gReplace (void *start, void *end ,int size,void *old, void  *new2 , int  (*cmp)
(void *, void*) ){
	char** A;
	int count = 0;
	for(int i=0;(char*)start+i*size<=end;i++){
		if(cmp((char*)start+i*size,old)==0)
			count++;
	}

	A = (char**)calloc(count,sizeof(char*));
	count=0;
	for(int i=0;(char*)start+i*size<=end;i++){
		if(cmp((char*)start+i*size,old)==0){
			A[++count]=(char*)start+i*size;
			for(int j=0;j<size;j++){
				*((char*)start+i*size+j)=*((char*)new2+j);
			}
		}
	}
	return (void**)A;
}
```

פונקציה שמקבלת עץ ומחזירה מערך של מצביעים לרשימות שכל רשימה היא מסלול אחר בעץ מהשורש לעלה אחר.

```
struct TNode {
	int info;
	struct TNode *left,*right;
};
typedef struct TNode TNode;
struct LNode {
	int info;
	struct LNode *next;
};
typedef struct LNode LNode;
int numOfLeaf(TNode* root){
	if(!root) return 0;
	if(!root->left && !root->right){
		return 1;
	}
	return numOfLeaf(root->right)+numOfLeaf(root->left);
}
LNode* listOfLeaf(TNode* root){
	if(!root)
		return NULL;
	if(!root->left&& !root->right)
	{
		LNode* head = (LNode*)malloc(sizeof(LNode));
		head->info=root->info;
		head->next=NULL;
		return head;
	}
	LNode* left = listOfLeaf(root->left);
	LNode* right = listOfLeaf(root->right);
	if(left){
		LNode* temp=left;
		while(temp->next){
			temp=temp->next;
		}
		temp->next=right;
		return left;
	}
	return right;
}
LNode* getPath(TNode* root,int info){
	if(!root)
		return NULL;
	if(root->info==info){
		LNode* head = (LNode*)malloc(sizeof(LNode));
		head->info=info;
		head->next=NULL;
		return head;
	}
	LNode* l=getPath(root->left,info);
	LNode* r=getPath(root->right,info);
	if(l){
		LNode* head = (LNode*)malloc(sizeof(LNode));
		head->info=root->info;
```

```
		head->next=l;
		return head;
	}
	if(r){
		LNode* head = (LNode*)malloc(sizeof(LNode));
		head->info=root->info;
		head->next=r;
		return head;
	}
}
TNode* add(TNode* root,int info){
	if(!root){
		root=(TNode*)malloc(sizeof(TNode));
		root->info=info;
		root->left=NULL;
		root->right=NULL;
		return root;
	}
	if(info<root->info){
		root->left=add(root->left,info);
	}
	else{
		root->right=add(root->right,info);
	}
	return root;
}
LNode** getPathes(TNode* root){
	int n= numOfLeaf(root);
	LNode**	Answer=(LNode**)calloc(n+1,sizeof(LNode*));
	LNode* head = listOfLeaf(root);
	LNode* temp=head;
	for(int i=0;i<n;i++){
		Answer[i]=getPath(root,temp->info);
		temp=temp->next;
	}
	Answer[n]=NULL;
	return Answer;
}
```

פונקציה שמקבלת מערך של רשימות של מסלול משורש לעלה ומחזירה עץ

```
TNode* add(TNode* root,int info){
	if(!root){
		root=(TNode*)malloc(sizeof(TNode));
		root->info=info;
		root->left=NULL;
		root->right=NULL;
		return root;
	}
	if(info<root->info){
		root->left=add(root->left,info);
	}
	else{
		root->right=add(root->right,info);
	}
	return root;
}
TNode* Search(TNode* root,int info){
	if(!root) return NULL;
	if(root->info==info) return root;
	if(root->info<info){
		return Search(root->right,info);
	}
	else
		return Search(root->left,info);
}
TNode* createTreeFromPathes(LNode** arrL){
	TNode* root=NULL;
	while(*arrL){
		LNode* head=*arrL;
		while(head){
			TNode* t=Search(root,head->info);
			if(!t)
				root=add(root,head->info);
			head=head->next;
		}
		arrL++;
	}
	return root;
}
```

החזרת מערך של מצביעים לרשימות שכל רשימה היא רמה אחרת בעץ

```
struct Node {
	int info;
	struct Node *left,*right;
};

typedef struct Node Node;

struct List {
	int info;
	struct List *next;
};

typedef struct List List;
List* addToEnd(List* head,List* node){
	if(!head)
		return node;
	List* temp = head;
	while(temp->next){
		temp=temp->next;
	}
	temp->next=node;

}
List* ElemntsK(Node * root, int k){
	if(!root) return NULL;
	if(k==0){
		List* head = (List*)malloc(sizeof(List));
		head->info=root->info;
		head->next=NULL;
		return head;
	}
	List* head = ElemntsK(root->left,k-1);
	List* node = ElemntsK(root->right,k-1);
	return addToEnd(head,node);
}
int TreeH(Node* root){
	if(!root) return 0;
	int n1= TreeH(root->left);
	int n2 = TreeH(root->right);
	return (n1>n2 ? n1:n2 )+1;
}
List** TreeToArrOfList(Node* root){
	int h= TreeH(root);
	List** answer=(List**)calloc(h,sizeof(List*));
	for(int i=0;i<h;i++){
		answer[i]=ElemntsK(root,i);
	}
	return answer;
}
Node* add(Node* root,int info){
	if(!root){
		Node* temp=(Node*)malloc(sizeof(Node));
		temp->info=info;
		temp->left=NULL;
		temp->right=NULL;
		return temp;
```

```
	}
	if(info<root->info){
		root->left=add(root->left,info);
	}
	else
	{
		root->right=add(root->right,info);
	}
	return root;
}
```

להפוך עץ לרשימה דו כיוונית מעגלית

```
Node* invertTreeList(Node* root){
	if(!root) return NULL;
	if(!root->first && !root->second){
		root->first=root;
		root->second=root;
		return root;
	}
	Node* small=invertTreeList(root->first);
	Node* big=invertTreeList(root->second);
	if(small && big){
		root->first=small->first;
		small->first->second=root;
		small->first=big->first;
		big->first->second=small;
		big->first=root;

		root->second=big;
		//small->second=root;
		return small;
	}
	if(small){
		root->first=small->second;
		small->second->second=root;
		root->second=small;
		return small;
	}
	if(big){
		root->first=big->first;
		big->first->second=root;
		big->first=root;
		root->second=big;

		return root;
	}
}
```

מיון שני רשימות ממוינות לרשימה אחת (רשימה חדשה לא מוקצת בזיכרון, בשניה כן)

```
Node* mergeToNewRec(Node* l1,Node* l2){
	if(!l1 && !l2) return NULL;
	if(!l2){
		Node* head = (Node*)malloc(sizeof(Node));
		head->num=l1->num;
		head->next=mergeToNew(l1->next,NULL);
		return head;
	}
	if(!l1){
		Node* head = (Node*)malloc(sizeof(Node));
		head->num=l2->num;
		head->next=mergeToNew(l2->next,NULL);
		return head;
	}
	if(l1->num<l2->num){
		Node* head = (Node*)malloc(sizeof(Node));
		head->num=l1->num;
		head->next=mergeToNewRec(l1->next,l2);
		return head;
	}
	else{
		Node* head = (Node*)malloc(sizeof(Node));
		head->num=l2->num;
		head->next=mergeToNewRec(l1,l2->next);
		return head;

	}
}
Node* merge(Node* l1,Node* l2){
	if(!l1 && !l2)
		return NULL;
	if(!l1)
		return l2;
	if(!l2)
		return l1;
	Node* h1,*h2;
	if(l1->num>l2->num){
		h1=l2;
		h2=l1;
	}
	else
	{
		h1=l1;
		h2=l2;
	}
	Node* merged =merge(h1->next,h2);
	h1->next=merged;
	return h1;

}
```

מקבלת רשימה של תוים וקידוד ההופמן שלהם ומייצר עץ הופמן

הפונקציה הרקורסיבית buildHTree מקבלת כפרמטר רשימה מקושרת של איברים מסוג Symbol ויוצרת עץ קידוד

של הופמן בהתאם לתווים ולקודים שלהם אשר מוגדרים ברשימה המקושרת.

רמז: תווים שהקודים שלהם מתחילים ב- 0 ישויכו (ברקורסיה) לתת העץ השמאלי ותווים שהקודים שלהם מתחילים ב- 1

ישויכו (ברקורסיה) לתת העץ הימני. כלומר, יש לחלק את הרשימה המקושרת לשתי רשימות : רשימה את מכילה את

האיברים שהקוד שלהם מתחיל ב- 0 ורשימה שניה מכילה את האיברים שהקוד שלהם מתחיל ב- 1.

ניתן להניח שהרשימה ממוינת לפי הקודים של התווים בסדר מילוני.

```
typedef  struct  HNode{
	char  chr;
	struct HNode *left, *right;
} HNode;
typedef  struct {
	char chr;
	char  *code;
}Symbol;
typedef  struct LNode {
	Symbol  s;
	struct LNode *next;
}LNode;
HNode* createNode(char c){
	HNode* root = (HNode*)malloc(sizeof(HNode));
	root->chr=c;
	root->left=root->right=NULL;
	return root;
}
HNode* createTree(Symbol s){
	char* temp=s.code;
	HNode* root=createNode(0);
	HNode* tempNode=root;
	while(strlen(temp)>0){
		if(*temp=='1'){
			tempNode->left=createNode(0);
			tempNode=tempNode->left;
		}
		else{
			tempNode->right=createNode(0);
			tempNode=tempNode->right;
		}
		temp++;
	}
	tempNode->chr=s.chr;
	return root;
}
LNode* add(LNode* head,LNode* attach){
	if(!head){
		attach->next=NULL;
		return attach;
	}
```

```
        attach->next=head;
        return attach;
}
HNode* BuildTree(LNode* head){
        if(!head) return NULL;
        if(!head->next){
                return createTree(head->s);
        }
        LNode * leftTree=NULL;
        LNode * rightTree=NULL;
        LNode* temp=head;
        LNode* temp2=head;
        while(temp2){
                temp=temp2;
                temp2=temp2->next;
                if(*(temp->s.code)=='1'){
                        leftTree=add(leftTree,temp);
                        ++(leftTree->s.code);
                }
                else
                {
                        rightTree=add(rightTree,temp);
                        ++(rightTree->s.code);
                }
        }
        HNode* root=createNode(0);
        root->left=BuildTree(leftTree);
        root->right=BuildTree(rightTree);
        return root;
}
Symbol newSymbol(char c,char* code){
        Symbol n ;
        n.chr=c;
        n.code=code;
        return n;
}
LNode* newNode(Symbol s){
        LNode* head = (LNode*)malloc(sizeof(LNode ));
        head->s=s;
        head->next=NULL;
        return head;
}
LNode* addInFront(LNode* head,LNode* next){
        next->next=head;
        return next;
}
```

מקבל שתי רשימות אחת שמייצגת preoreder ואחת inorder של עץ ומחזירה את העץ

```
typedef struct LNode {
int  info;
struct  LNode *next;
} LNode;
LNode * search (LNode * list , int data){
	while(list){
		if(list->info==data)
			return list;
		list=list->next;
		}
	return NULL;
	}
void destroyList(LNode* list){
	if(!list)
		return;
	destroyList(list->next);
		free(list);
}
typedef  struct TNode  {
    int    info;
  struct TNode * left , *right;
}TNode;

typedef  struct  {
  void   *elm;
  int       rnk;
}Rank;
LNode* addList(LNode* list,int info){
	LNode* L = (LNode*)malloc(sizeof(LNode));
	L->info=info;
	if(!list){
		L->next=NULL;
		return L;
	}
	LNode* temp = list;
	while(temp->next)
		temp=temp->next;
	L->next=NULL;
	temp->next=L;
	return list;
}
TNode * cBTree(LNode* pre,LNode* in){
	if(!pre)
		return NULL;
	LNode * right =NULL;
	LNode * left = NULL;
	TNode* root=(TNode*)malloc(sizeof(TNode));
	root->info=pre->info;
	LNode* temp;
	LNode* leftin =NULL;
	LNode* rightin = NULL;
	temp=search(in,root->info);
	LNode* temp2=in;
	while(temp2->info!=temp->info){
		leftin=addList(leftin,temp2->info);
		temp2=temp2->next;
```

```
}
LNode * temp3 = temp->next;
while(temp3){
        rightin=addList(rightin,temp3->info);
        temp3=temp3->next;
}
temp=pre;
while(temp){
        if(search(leftin,temp->info))
                left=addList(left,temp->info);
        else if (search(rightin,temp->info))
                right=addList(right,temp->info);
        temp=temp->next;
}
root->left=cBTree(left,leftin);
root->right=cBTree(right,rightin);
destroyList(pre);
destroyList(in);
return root;
}
```

מקבלת מספר לא ידוע של איבר מסוג לא ידוע ומצביע לפונקצית השוואה ומחזירה מצביע לרשימה ממוינת של האיברים

```
typedef  struct Item {
	void *data;
	struct Item *next;
} Item;
typedef  struct List {
	Item *head;
	int  elmSize;
	int  (*cmp) (void *,void*);
}List ;
int intCmp( void* P1, void* P2){

	return *(int*)P1 - *(int*)P2;
}
int doubleCmp (void* P1, void* P2){
	if( *(double*)P1 - *(double*)P2 == 0)
		return 0;
	else
		if( *(double*)P1 - *(double*)P2 < 0 ) return -1;
		else
			return 1;
}
void initList(List* L,int elemSize,int (*cp)(void*,void*)){
	L->elmSize=elemSize;
	L->cmp=cp;
	L->head=NULL;
}
Item* addItem(Item* head,int sElement,void* data,int (*cp)(void*,void*)){
	if(!head){
		Item* temp=(Item*)malloc(sizeof(Item));
		temp->next=NULL;
		temp->data=malloc(sElement);
		memcpy(temp->data,data,sElement);
		return temp;
	}
	if(cp(data,head->data)<0){
		Item* temp=(Item*)malloc(sizeof(Item));
		temp->next=head;
		temp->data=malloc(sElement);
		memcpy(temp->data,data,sElement);
		return temp;
	}
	else{
		head->next=addItem(head->next,sElement,data,cp);
		return head;
	}
}
void add(List* a,...){
	va_list param;
	va_start(param,a);
	char* temp;
	int count=0;
	Item * head=a->head;
	while((temp = va_arg(param,char*))!=NULL){
		a->head=addItem(a->head,a->elmSize,temp,a->cmp);
	}
```

```
}
```

עבודה עם קבצים

מקבל מערך של מצביעים לקבצים ומחזיר מערך של מצביעים למערכים כאשר מכל קובץ יצרנו מערך חדש

```
int* readFile(FILE* f){
	fseek(f,0,SEEK_END);
	int n=ftell(f)/sizeof(int);
	int * A = (int*)malloc(ftell(f)/sizeof(int));
	rewind(f);
	int * B = A;
	while(!feof(f)){
		fread(A,sizeof(int),1,f);
		A++;
	}
	return B;
}
int** readFiles(FILE** arrF){
	FILE ** temp = arrF;
	while(*temp){
		temp++;
	}
	int** A =(int**) malloc((sizeof(int*)*((temp-arrF)+1)*2)+1);
	temp=arrF;
	while(*temp){
		A[(temp-arrF)*2]=readFile(*temp);
		A[(temp-arrF)*2+1]=A[temp-arrF]+(ftell(*temp)/sizeof(int)-1);
		temp++;
	}
	return A;
}
```

שאלה עם הקובץ של יוניקס – פתרון של אחמד

להלן המבנה של קובץ המשתמשים (/etc/passwd) במערכת יוניקס.

shell : ספריית בית : מידע כללי : מספר קבוצת המשתמש : מספר משתמש : סיסמה : שם משתמש

כל שורה בקובץ זה מתארת משתמש אחר. להלן מספר דוגמאות של שורות שנלקחו מקובץ זה:

coklaris:x:1706:1706:COHEN KLARIS:<u>/home/**bs**/bsy00/coklaris</u>:/bin/sbash

comfofer:x:1708:1708:COMFORT OFER:<u>/home/**cs**/csy00/comfofer</u>:/bin/sbash

varda:x:209:209:Varda Presman, library staf:<u>/home/**library**/varda</u>:/bin/bash

שדה מספר 6 (עם קו תחתון) בכל שורה מתאר את ספריית הבית של כל משתמש. ספריה זו מתחילה ב- /home ואחריה תיקיה שממנה ניתן ללמוד על הקבוצה שאליה משתייך המשתמש. למשל, השורה הראשונה מתארת סטודנט שלומד bs (מדעי התנהגות) ובשורה השניה מתואר סטודנט שלומד מדעי המחשב (cs). בשורה האחרונה מתוארת עובדת ספריה (library).

המטרה היא לקרוא את תוכן הקובץ passwd וליצור קובץ חדש לכל קבוצת משתמשים. שם הקובץ החדש יהיה כשם התיקיה שמופיעה בספריית הבית לאחר התיקיה home (התיקיה המודגשת bs,cs,library). **ניתן להניח שכל שורות הקובץ תואמות למבנה שתואר.**

מבנה השורות בקבצים החדשים יהיה זהה למבנה של הקובץ passwd רק שלכל משתמש יישמרו השדות הבאים: שם משתמש (שדה 1) , מספר משתמש (שדה 3), מידע כללי על המשתמש (שדה 5).

כתוב את הפונקציה writeToFiles. להלן חתימת הפונקציה ולאחר מכן הסבר על אופן פעולתה.

```
void  writeToFile ( FILE  *all[], char*  names[]) {

}
```

הפרמטר names מכיל את כל שמות הקבצים שהם שמות קבוצות המשתמשים (bs,cs,library וכו') שהופיעו בקובץ passwd.
התא האחרון במערך זה שווה NULL.

הפרמטר all מכיל מצביעים לקבצים ששמותיהם נמצאים בפרמטר names. קבצים אלה נוצרו לכתיבה והם ריקים. התא האחרון במערך זה שווה NULL. המצביע במקום ה- k במערך all תואם לשם במקום ה- k במערך names.

הפונקציה writeToFiles תקרא את תוכן הקובץ passwd ותמלא את הקבצים השונים במשתמשים המתאימים.

```
/* Exam 25/02/2009 sheela 2 */

/*********************************************/
/*  This program cannot be run in Windows OS  */
/*  (limits.h in VS is not contain LINE_MAX)  */
/*********************************************/

#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <stdlib.h>

void  writeToFile ( FILE  *all[], char*  names[])
{
	int i=0;
	int j;
	char* st;
	char line[LINE_MAX+1];
	char* login;
	char* info;
	char* uid;
	FILE *fp, *fw;
	fp = fopen("//etc/passwd","r");
	while(*all+i)
	{
		//fp = *(all+i);
		while(fgets(line,LINE_MAX,fp))
		{
			strcpy(login,strtok(line, ":"));
```

```
                        strtok(NULL,":");
                        strcpy(uid,strtok(NULL,":"));
                        strtok(NULL,":");
                        strcpy(info,strtok(NULL,":"));
        strtok(NULL,"/");
        st = strcpy(st,strtok(NULL,"/"));
        j=0;
                        while(*names && strcmp(st,*(names+j))==0)
                        {
                                fprintf(fw,puts(login),":",puts(uid),":",puts(info));
                                j++;
                        }
                }
                i++;
        }

}

int main()
{
        char *names[] = {"bs","cs","library",NULL};
        FILE *f1, *f2, *f3;
        FILE *f[4];
  f1 = fopen("//bs","w+");
        f2 = fopen("//cs","w+");
        f3 = fopen("//library","w+");
        f[0] = f1;
        f[1] = f2;
        f[2] = f3;
```

```
            f[3] = NULL;
            writeToFile(f,names);
   fclose(f1);
   fclose(f2);
   fclose(f3);
   return 1;
}
```

שאלה עם מבנה שמייצג אנשים עדכונים נתונים והוספה לקובץ

נתון המבנה Person אשר מייצג אדם עם שלושה שדות : ת.ז , שם פרטי וגיל.

```
typedef  struct {
 char  id[10];
 char  name[16];
  int  age;
} Person ;
```

נתון קובץ בשם persons.dat אשר מכיל מבנים מסוג Person . בנוסף, נתון קובץ בשם pIndex.dat שמכיל את ההיסטים של המבנים בקובץ persons.dat. **היסט** (offset) של מבנה בקובץ הוא מרחקו בבתים מתחילת הקובץ. נתון שההיסטים בקובץ pIndex.dat ממוינים בסדר עולה כאשר מפתח המיון הוא השדה age של המבנה Person. כלומר , הערך הראשון בקובץ pIndex.dat הינו ההיסט של המבנה (מתוך הקובץ persons.dat) בעל השדה age הקטן ביותר והערך האחרון בקובץ pIndex.dat הינו ההיסט של המבנה בעל השדה age הגדול ביותר.

דוגמה :

נניח שיש לנו 5 מבנים מסוג Person שמקיימים את היחס הבא :

person3<=person5<=person1<=person4<=person2

כלומר, השדה age של person3 הוא הקטן ביותר והשדה age של person2 הוא הגדול ביותר.

להלן תיאור התוכן של שני הקבצים :

persons.dat

person1person2person3person4person5

pIndex.dat

60 120 0 90 30

שים לב שהמבנים בקובץ persons מופיעים על פי סדר הוספתם לקובץ.

המספר 60 בקובץ pIndex.dat הוא ההיסט של המבנה בעל הגיל הקטן ביותר. לפי המספר 60 ניתן להבין שמדובר ב-person3. המספר 120 הוא ההיסט של המבנה person5 שמכיל את הגיל הבא בגודלו אחרי הגיל של person3 וכו'.

כתוב פונקציה בשם addItems אשר מקבלת את הפרמטרים הבאים :

- מצביע לקובץ persons.dat אשר פתוח לקריאה/כתיבה בינארית.
- מצביע לקובץ pIndex.dat אשר פתוח לקריאה/כתיבה בינארית.
- החל מהפרמטר השלישי כל פרמטר הוא מצביע למבנה מסוג Person.
- הפרמטר האחרון הוא תמיד NULL

הפונקציה תוסיף את כל המבנים שהתקבלו כפרמטרים לקובץ persons.dat ותעדכן בהתאם את הקובץ pIndex.dat .

```
typedef  struct {
 char  id[10];
 char  name[16];
  int  age;
} Person ;
typedef struct{
int age;
long offset;
} Data;
void addPerson(FILE* person,FILE* pIdx,...){
       Person* temp;
       va_list param;
       Person temp3;
       va_start(param,pIdx);
       fseek(person,0,SEEK_END);
       while(temp=va_arg(param,Person*)){
              fwrite(temp,sizeof(Person),1,person);
       }
       rewind(person);
       LNode* head = NULL;
```

```
	while(!feof(person)){
		Data temp2;
		temp2.offset=ftell(person);
		fread(&temp3,sizeof(Person),1,person);
		temp2.age=temp3.age;
		head=add(head,temp2);
	}
	rewind(pIdx);
	while(head){
		printf("age is  - %d ",head->temp.age);
		long t = head->temp.offset;
		fwrite(&t,sizeof(long),1,pIdx);
		LNode* node2=head;
		head=head->next;
		free(node2);
	}
	va_end(param);
}
Person newPerson(int age,char* name,char* id){
Person temp;
temp.age=age;
strcpy(temp.name,name);
strcpy(temp.id,id);
return temp;
}
void readPerson(FILE* f){
	rewind(f);
	Person temp;
	while(!feof(f)){
		fread(&temp,sizeof(temp),1,f);
		printf("%s %s %d\n",temp.name,temp.id,temp.age);
	}
}
void readPidx(FILE* f){
	rewind(f);
	long t;
	while(!feof(f)){
	fread(&t,sizeof(long),1,f);
	printf("%d \n",t);
	}
}
```

מחיקת הערות מקובץ מקור של תוכנית C

```c
void deleteComments(FILE* c,FILE* x,FILE* y){
	int n;
	int count=0;
	while((n=getc(c))!=EOF){
		int flag=0;
		if(n=='/')
		{
			int n2 = getc(c);
			if(n2==EOF) return;
			if(n2=='/'){
				flag=1;
				fwrite(&count,sizeof(int),1,y);
				count=0;
				putc('/',y);
				putc('/',y);
				while((n=getc(c))!=EOF && n!='\n'){
					putc(n,y);
				}
			}
			else if(n2=='*'){
				flag=1;
				fwrite(&count,sizeof(int),1,y);
				count=0;
				putc('/',y);
				putc('*',y);
				while((n2=getc(c))!=EOF){
					if(n2 == '*'){
						n=getc(c);
						if(n=='/'){
							putc('*',y);
							putc('/',y);
							break;
						}
						else{
						putc(n2,y);
						putc(n,y);
						}
					}
					putc(n2,y);
				}
			}
			else{
				putc(n,x);
				count++;
				putc(n2,x);
				count++;
				continue;
			}
		}
		if(!flag){
		count++;
		putc(n,x);
		}
	}
}
```

קריאת הכותרת של קובץ GZ לא נבדק

```
FILE * gzipReadHeader(char * fname){
	char line[128]=0;
		// 1. was  wasn't compressed by gz (checked by the first to bytes first needs to be 31 seccond equals to 139
		// 2. check fname flag to check if orig fname is present, flg  byte is the fourth byte, and fname flag is the third,fname stored as the seccond string after
	FILE* gz = fopen(fname,"rb+");
	//deal with Erno;
	FILE* out = fopen("out.txt","wb+");
	char n1,n2;
	fread(&n1,sizeof(char),1,gz);
	fread(&n2,sizeof(char),1,gz);
	if(n1==31&& n2==139){
		fprintf(out,"Was commpresed by gz ");
	}
	else{
		fprintf(out,"Wasn't commpresed by gz ");
		return out;
	}
	fseek(gz,4,SEEK_SET);
	fread(&n1,sizeof(char),1,gz);
	int fnamebit = getBit(n1,3);
	int fexetra = getBit(n1,2);
	if(!fnamebit){
		fprintf(out,"file name wasn't present");
	}
	fseek(gz,9,SEEK_SET);
	fread(&n1,sizeof(char),1,gz);
	switch(n1){
	case 3:
		fprintf(out,"genereted in unix os");
		break;
	case 7:
		fprintf(out,"genereted in mac os");
		break;
	case 0:
		fprintf(out,"genereted in win os");
		break;
	default:
		fprintf(out,"genereted in unkown os");
		break;
	}
	if(!fexetra){
		fseek(gz,10,SEEK_SET);
	}
	else{
		fseek(gz,12,SEEK_SET);
	}
	fread(line,sizeof(char),127,gz);
	fprintf(out,"%s",line);
	int fsize;
	fseek(gz,-4,SEEK_END);
	fread(&fsize,sizeof(int),1,gz);
	fprintf(out,"file size is  - %d \n",fsize);
	return out;
}
```

```
int getNum(FILE* f,int rows,int cols,int n,int m,int r){
	if(n>rows || n<0 ||m<0 || m>cols)
		return 0;
	int temp;
	fseek(f,(n-1)*m*r*sizeof(int)+12,SEEK_SET);
	fread(&temp,sizeof(int),1,f);
	return temp;

}
int numOfN(FILE* f,int rows,int cols,int n,int m,int r){
	int count=0;
	if(getNum(f,rows,cols,n-1,m,r)) count++;
	if(getNum(f,rows,cols,n+1,m,r)) count++;
	if(getNum(f,rows,cols,n,m+1,r)) count++;
	if(getNum(f,rows,cols,n,m-1,r)) count++;
	if(getNum(f,rows,cols,n-1,m-1,r)) count++;
	if(getNum(f,rows,cols,n+1,m+1,r)) count++;
	if(getNum(f,rows,cols,n+1,m-1,r)) count++;
	if(getNum(f,rows,cols,n-1,m+1,r)) count++;
	return count;
}
void nextGen(char* fname){
	FILE * f=fopen(fname,"rb+");
	if(!f){
		perror("terminated, reason ->");
		return;
	}
	int g,r,c;
	int one=1;
	int zero=0;
	fread(&g,sizeof(int),1,f);
	fread(&r,sizeof(int),1,f);
	fread(&c,sizeof(int),1,f);
	for(int i=0;i<r;i++)
		for(int j=0;j<c;j++){
			if(getNum(f,r,c,i,j,g)){
				switch(numOfN(f,r,c,i,j,g)){
				case 2:
				case 3:
					fseek(f,0,SEEK_END);
					fwrite(&one,sizeof(int),1,f);
					break;
				default:
					fseek(f,0,SEEK_END);
					fwrite(&zero,sizeof(int),1,f);
				}
			}
			else
			{
				if(numOfN(f,r,c,i,j,g)==3){
					fseek(f,0,SEEK_END);
					fwrite(&one,sizeof(int),1,f);
				}
				else{
					fseek(f,0,SEEK_END);
					fwrite(&zero,sizeof(int),1,f);
```

```
                    }
                }
            }
}
```

פונקציה שמקבלת מצביע לקובץ עם איברים ממוינים בסדר עולה והשני בסדר יורד ומבצעת מיזוג אל תוך קובץ חדש

```
FILE* merge(FILE* up,FILE* down){
	FILE* out=fopen("out.bin","wb+");
	int i=1,j=-1;
	int n1,n2;
	int sizeofup=sizeOfFile(up)/sizeof(int);
	int sizeofdwon=sizeOfFile(down)/sizeof(int);
	fseek(up,0,SEEK_SET);
	fseek(down,-1*sizeof(int),SEEK_END);
	fread(&n1,sizeof(int),1,up);
	fread(&n2,sizeof(int),1,down);
	j--;
	while(i<sizeofup && -j<sizeofdwon){
		if(n1<n2){
			fwrite(&n1,sizeof(int),1,out);
			fread(&n1,sizeof(int),1,up);
			i++;
		}
		else{
			fwrite(&n2,sizeof(int),1,out);
			fseek(down,j*sizeof(int),SEEK_END);
			j--;
			fread(&n2,sizeof(int),1,down);

		}
	}
	while(i<sizeofup){
		fread(&n1,sizeof(int),1,up);
		fwrite(&n1,sizeof(int),1,out);
		i++;
	}
	while(-j<sizeofdwon){
		fread(&n2,sizeof(int),1,down);
		j--;
		fseek(down,j*sizeof(int),SEEK_END);
		fwrite(&n2,sizeof(int),1,out);
	}
	return out;
}
```

מקבל קובץ מקור בלי הערות וקובץ הערות ומבצע שחזור

```c
FILE* restoreComments(FILE* s,FILE* c){
	FILE * out = fopen("3.txt","w+");
	char line[129];
	char line2[129];
	int count=0;
	while(fgets(line,128,c)){
		int numline = atoi(strtok(line,":"));
		int mode = atoi(strtok(NULL,":"));
		while(numline-count-1){
			fgets(line2,128,s);
			fputs(line2,out);
			count++;
		}
		if(mode==1){
			fgets(line2,128,s);
			strcat(line2,strtok(NULL,":"));
			fputs(line2,out);
			count++;
		}
		if(mode==0){
			strcpy(line2,strtok(NULL,":"));
			fputs(line2,out);
			count++;
		}
	}
	return out;
}
```

מקבל קובץ עם 81 מספרים ובודק אם מייצגים סודוקו – לא נבדק

```
int CheckRows(FILE * f){
	int n=0;
	int sum=0;
	int i=1;
	while(fread(&n,sizeof(int),1,f)){
		sum+=n;
		i++;
		if(i==9){
			if(sum!=45){
				return 0;
			}
			sum=0;
			i=0;
		}
	}
}
int checkColom(FILE * f,int col){
	fseek(f,0,SEEK_SET);
	int n=0;
	int i=0;
	int sum=0;
	while(fread(&n,sizeof(int),1,f)){
		i++;
		if(i==col){
			sum+=n;
		}
		if(i=9){
			i=0;
		}
	}
	if(sum==45)
		return 1;
	else
		return 0;
}
int CheckSudoko(FILE * f){
	int flag=1;
	flag=CheckRows(f);
	if(!flag) return 0;
	for(int i=1;i<=9;i++){
		flag=checkColom(f,i);
		if(!flag)
			return 0;
	}
	return 1;
}
```

Read Files

הפונקציה readFromFiles מקבלת מערך שמות של קבצים בינאריים שמכילים מספרים שלמים (NULL בתא האחרון של המערך). כמות המספרים בכל קובץ לא ידועה מראש וגם משתנה מקובץ לקובץ. על הפונקציה לקרוא את תוכן הקבצים השונים ולשמור את התוכן של כל אחד מהקבצים במערך אחר שיוקצה בצורה דינמית. המערכים המוקצים עבור הקבצים השונים יוחזקו כולם בשני מערכי מצביעים (NULL בתא האחרון של כל אחד) באופן הבא: תאי מערך המצביעים הראשון, מצביעים על התאים

הראשונים בכל מערך מספרים ותאי מערך המצביעים השני מצביעים על התאים האחרונים בכל מערך מספרים. בנוסף, שני מערכי המצביעים יוחזקו במערך מצביעים שלישי (בגודל שני תאים) שהתא הראשון בו יצביע על התא הראשון של מערך המצביעים הראשון והתא השני יצביע על התא הראשון של מערך המצביעים השני.

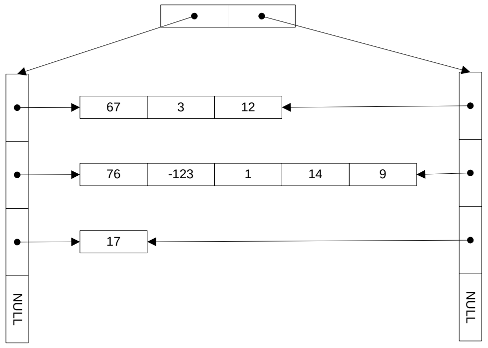
דוגמה:

נניח שיש לנו שלושה קבצים f1, f2, f3 .

הקובץ f1 מכיל את המספרים 67 , 3 , 12 .

הקובץ f2 מכיל את המספרים 76, -123 , 1 , 14 , 9

הקובץ f3 מכיל את המספרים : 17



על הפונקציה להחזיר את המבנה שנוצר באיור.

```
int* readFile(FILE* f){
	fseek(f,0,SEEK_END);
	int * A = (int*)malloc(ftell(f)/sizeof(int));
	rewind(f);
	int * B = A;
	while(!feof(f)){
		fread(A,sizeof(int),1,f);
		A++;
	}
	return B;
}
int*** readFromFiles(FILE** F){
	FILE** temp = F;
	while(*temp++);
	int count = temp-F;
	int*** A = (int***)calloc(2,sizeof(int**));
	A[0]=(int**)calloc(count,sizeof(int*));
	A[1]=(int**)calloc(count,sizeof(int*));
	temp=F;
	int k=0;
	while(*temp){
		fseek(*temp,0,SEEK_END);
		int count2 = ftell(*temp)/sizeof(int);
		int* B=readFile(*temp);
		A[0][k]=B;
		A[1][k]=B+count2-1;
		k++;
		temp++;
	}
	A[1][k]=NULL;
	A[0][k]=NULL;
	return A;
}
```

מצביעים בלבד

שחלוף מטריצה משיעורי בית

```
void transpose(int** A){
int ** i = A;
int ** max = i;
while(*++i){
   max = (**max<**i)? i:max;
   }
 int* r = *max+1;

 while(r-*max<**max){
  i=A;
  do{
   if(r-*max<**i)
       printf("%3d",*(*i+ (r-*max)));
   else
       printf("   ");
  }while(*++i);
  printf("\n");
  r++;
 }
```

שכפול מערך של מערכים בלי מקומות מיותרים

```
int** duplicate(int** all){
	int** temp = all;
	while(*temp){
		temp++;
	}
	int ** Re = (int**)calloc((temp-all),sizeof(int*));
	temp=all;
	while(*temp){
		Re[temp-all]=newA(all[temp-all],all[temp-all+1]);
		Re[temp-all+1]=Re[temp-all]+(all[temp-all+1]-all[temp-all]);
		temp+=2;
	}
	return Re;
}
```

איחוי מערך של מערכים צמצום מקומות מיותרים

```
int checkRoom(int* first,int* last){
       return *first-(last-first)-1;
}
void defrag(int** all){
       int** temp=all;
       int** temp2=all+2;
       while(*(temp2)){
       while(checkRoom(*temp,*(temp+1))>0){
              if(*temp2==*(temp2+1)||*temp==*temp2)     temp2+=2;
              if(!*temp2) break;
              shift(temp,(temp+1),(temp2),(temp2+1));
              }
       temp+=2;
       }
}
```

Mystery

הפונקציה הראשית main מגדירה , בין היתר, את המערך All ששומר את כתובת ההתחלה והסוף של כל אחד מהמערכים A,B,C. (ראה ציור 1).

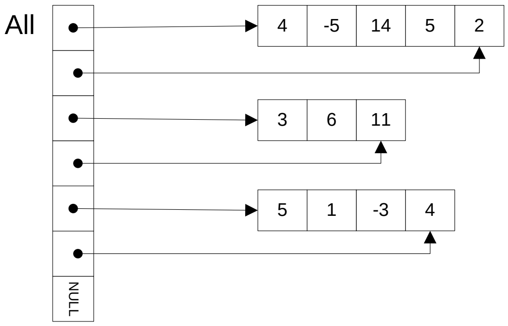
הפונקציה main מפעילה את הפונקציה mystery שמקבלת כפרמטר את המערך All ומייצרת מבנה חדש שגם הוא שומר את כתובת ההתחלה והסוף של כל אחד מהמערכים A,B,C. (ראה ציור 2).

שים לב שהמערכים A,B,C אינם מוקצים מחדש. בשני המבנים (הציורים) מדובר באותם מערכים. רק מטעמי נוחות הם צוירו פעמיים. רק המערכים העוטפים אותם מוקצים בפונקציה.

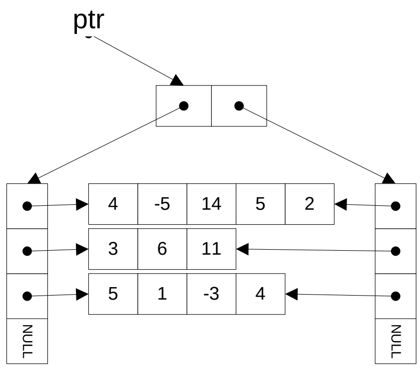
המשתנה ptr לא הוגדר בכוונה מפני שהגדרתו היא חלק מפתרון השאלה.

<u>כתוב את הפונקציה mystery תוך שימוש במצביעים בלבד.</u>

```
int main(){

 int A[]={4,-5,14,5,2};

 int B[4]={3,6,11};

 int C[6]={5,1,-3,4};

 int *All[] = {A,A+sizeof(A)/sizeof(A[0])-1, B,B+sizeof(B)/sizeof(B[0])-1, C,C+sizeof(C)/sizeof(C[0])-1,
NULL);

ptr = mystery(All);

}
```



ציור 1 (לפני הפעלת הפונקציה)



ציור 2 (אחרי הפעלת הפונקציה)

```
void*** mystery(void** B){
	char*** A= (char***)calloc(2,sizeof(char**));
	void ** temp=B;
	while(*temp){
		temp++;
	}
	A[0]=(char**)calloc((temp-B)/2,sizeof(char));
	A[1]=(char**)calloc((temp-B)/2,sizeof(char));
	temp=B;
	while(*temp){
		A[0][(temp-B)/2]=*(char**)temp;
		A[1][(temp-B)/2]=*(char**)(temp+1);
		temp+=2;
	}
	return (void***)A;
}
```

משחק החיים

```
#include <stdio.h>
#include <math.h>
#define MAXSIZE 50
typedef enum{FALSE,TRUE} boolean;
typedef  enum{O ='*', D='d',L='l'} tile ;
typedef struct
	//notice that the actual max size is 48 = 50 - 2 , because two rows and coloumns are dedecated to being the frame of the board
{
	tile box[MAXSIZE][MAXSIZE][MAXSIZE];
	int layers;
	int cur_l;
	int r;//w
	int c;//h
} Game;
void printBox(Game* A){
	for(int k=0;k<A->layers;k++){
		for(int i=0;i<=A->r+1;i++){
			for(int j=0;j<=A->c+1;j++)
				printf("%c",A->box[k][i][j]);
			printf("\n");
		}
		printf("\n\n");
	}
}

void initilizeBox(Game* A){
	int i,j,k;
	for(k=0;k<A->layers+1;k++)
		for(j=0;j<A->r+1;j++){
			for(int i=0;i<A->c+1;i++){
				A->box[k][j][i]=D;
			}
		}
			for(j=0;j<A->layers;j++){
				for(i=0;i<=A->c+1;i++){
					A->box[j][0][i]=O;
					A->box[j][A->r+1][i]=O;
				}
				for(i=0;i<=A->r+1;i++){
					A->box[j][i][0]=O;
					A->box[j][i][A->c+1]=O;
				}
			}
}
int num_of_vilable(Game* A,int i,int j){
	int sum=0;
	if(A->box[A->cur_l][i+1][j+1]==L)
		sum++;
	if(A->box[A->cur_l][i+1][j]==L)
		sum++;
	if(A->box[A->cur_l][i+1][j-1]==L)
		sum++;
	if(A->box[A->cur_l][i-1][j+1]==L)
```

```
			sum++;
		if(A->box[A->cur_l][i-1][j]==L)
			sum++;
		if(A->box[A->cur_l][i-1][j-1]==L)
			sum++;
		if(A->box[A->cur_l][i][j+1]==L)
			sum++;
		if(A->box[A->cur_l][i][j-1]==L)
			sum++;
		return sum;
}
void survival(Game* A,int i,int j){
		if( (A->box[A->cur_l][i][j]==L && ( num_of_vilable(A,i,j)==3
||num_of_vilable(A,i,j)==2 )) || ( A->box[A->cur_l][i][j]==D && num_of_vilable(A,i,j)==3
))
			A->box[A->cur_l+1][i][j]=L ;
		else
			A->box[A->cur_l+1][i][j]=D ;
}
void initData(Game* A){
		int i=1,j=1,k=0,temp=0;
		while(scanf("%d",&temp)!=EOF ){
			if(temp==0)
				A->box[k][j][i]=D;
			else
				A->box[k][j][i]=L;
			i++;
			if(i==A->c+1){
				j++;
				i=1;
			}
			if(j==A->r+1){
				return;
			}
		}
}
int initSize(Game* A){
		A->cur_l=0;
		int k=0,n=0,l=0;
		if(scanf("%d",&k)!=1){
			return 1;
			fprintf(stderr,"error");
		}
		if(scanf("%d",&n)!=1)
		{
			return 1;
			fprintf(stderr,"error");
		}
		if(scanf("%d",&l)!=1)
		{
			return 1;
			fprintf(stderr,"eror");
		}
		A->layers=k;
		A->r=n;
		A->c=l;
		return 0;
}
```

```
void generation(Game* A){
	for(int i=1;i<=A->r;i++){
		for(int j=1;j<=A->c ;j++)
			survival(A,i,j);
	}
	A->cur_l++;
}
void GamePlay(Game* A){
	while(A->cur_l+1<A->layers){
		generation(A);
	}
	printBox(A);
}
int main(){
	Game* p;
	Game A={{O},0,0,0};
	p=&A;
	if(initSize(p)==1){
		return 1;
	}
	initilizeBox(p);
	initData(p);
	GamePlay(p);
		getchar();
	return 0;
}
```

```
#include <stdio.h>
#include <math.h>
#include <string.h>
unsigned int BinaryR(char *S){
	unsigned int r=0;
	for(int i=0;i<strlen(S);i++){
		int n=S[i];
		n=n&15;
		int k=8;
		int temp=0;
		for(int j=0;j<4;j++){
			if(n&k){
				temp=1;
			}
			else
			{
				temp=0;
			}
			k=k>>1;
			r=r<<1;
			r+=temp;
		}
	}
	return r;
}
int moveBitToLeft(int a,int n){
	unsigned int x = pow(2.0,31);
	int temp=0;
	for(int i=1;i<=n;i++){
	if(x&a)
		temp=1;
	else
		temp=0;
	a=a<<1;
	a+=temp;
	}
	return a;
}
void printBit(int n){
	int x=1;
	char s[33]="";
	int k=31;
	for(int i=1;i<=8*sizeof(n);i++){
	if(x&n){
		s[k]='1';
	}
	else
	{
		s[k]='0';
	}
	x=x<<1;
	k--;
	}
	puts(s);
}
unsigned getbits(unsigned x, int p, int n){
```

```
		return (x >> (p+1-n)) & ~(~0 << n ) ;
}
unsigned setBit(unsigned x, int p,int n,unsigned y){
		unsigned temp = ~0>>32-(p+n);
		temp<<=32-(p+n);
		temp=~temp;
		x&=temp;
		y>>=n;
		y<<=n;
		return x | y ;
}
```

מיון מערך של מערכים של מחרוזות ומציאת המחרוזת הארוכה ביותר

```
#include <stdio.h>
#include <string.h>
 char* maxLengthString(const char***  A){
		const char* max=**A;
		const   char*** C=A;

		while(*C){
		const   char** P=*C;
			while(*P){
				if(strlen(max)<strlen(*P))
					max=*P;
			P++;
			}
			C++;
		}
		return (char*)max;
}
void sort(char*** A){
		int i=0;int j=0;int k=0;int r=0;
while(A[r]!=NULL){
		i=j=0;
		while(A[i]!=NULL){
			while(A[i][j]!=NULL){
			if(strcmpi(A[i][j],A[r][k])>0){
				char* String=A[i][j];
				A[i][j]=A[r][k];
				A[r][k]=String;
			}
			j++;
			}
			j=0;
			i++;
		}
		k++;
		if(A[r][k]==NULL){
			k=0;
			r++;
		}

}
}
void printAllStrings(  const char***  A){
const   char*** C=A;
		while(*C){
			const char** P=*C;
```

```
       while(*P!=NULL){
                     puts(*P);
                     P++;
              }
       C++;
       }
}
int  main ()
{
  char* arrP1 [ ] = { "father", "mother", NULL};
  char* arrP2 [ ] = { "sister", "brother", "grandfather", NULL };
  char* arrP3 [ ] = { "grandmother", NULL};
   char* arrP4 [ ] = { "uncle", "aunt", NULL };
          char** arrPP [ ] = { arrP1, arrP2, arrP3, arrP4 , NULL};
        printAllStrings((const char***) arrPP );
               sort( arrPP );
               printf("\n");
               printAllStrings( (const char***)arrPP );
               printf("\n%s\n", maxLengthString ((const char***) arrPP ));

 return 0;

}
```

הפונקציה מחזירה מערך של מבנה שמחזיק את הערך של עלה ואת הרמה שבא נמצאה

```
#include <stdio.h>

#include <stdlib.h>


typedef  struct  HNode{

 char  chr;

 struct HNode *left, *right;

} HNode;

typedef  struct {

 char chr;

 int counter;

}Symbol;

int isLeaf(HNode* root){

if(!root->left && !root->right)

   return 1;

   else return 0;

}

Symbol createSymbol(char c,int n){

   Symbol temp;

   temp.chr=c;

   temp.counter=n;

   return temp;

   }

Symbol* getSL(HNode* root){

if(!root)

   return NULL;

if(isLeaf(root)){

   Symbol* temp = (Symbol*)malloc(sizeof(Symbol)*2);

   temp[0]=createSymbol(root->chr,0);
```

```
    temp[1]=createSymbol('\0',-1);
    return temp;
}
Symbol* L = getSL(root->left);
Symbol* R = getSL(root->right);
int count = 0;
if(L){
    Symbol* temp =L;
    for(;temp->counter!=-1;temp++)
        count++;
}
if(R){
    Symbol* temp =R;
    for(;temp->counter!=-1;temp++)
        count++;
}
Symbol* m = (Symbol*)calloc(count+1,sizeof(Symbol));
int i=0;
if(L){
    Symbol* temp = L;
    for(;temp->counter!=-1;temp++)
        m[i++]=createSymbol(temp->chr,temp->counter+1);
}
if(R){
    Symbol* temp = R;
    for(;temp->counter!=-1;temp++)
        m[i++]=createSymbol(temp->chr,temp->counter+1);
}
m[i]=createSymbol('\0',-1);
```

```
free(L);
free(R);
return m;
}
HNode* add(HNode* root,char c){
if(!root){
   root = (HNode*)malloc(sizeof(HNode));
   root->chr=c;
   root->left=NULL;
   root->right=NULL;
   return root;
   }
   if(c>root->chr)
      root->left=add(root->left,c);
   else
      root->right=add(root->right,c);
   return root;
}
void destroy(HNode* root){
if(!root) return;
if(isLeaf(root))
   free(root);
destroy(root->left);
destroy(root->right);
}
int main()
{
   HNode* root = NULL;
```

```
 root=add(root,'1');
 root=add(root,'2');
 root=add(root,'0');
 root=add(root,'3');
   Symbol* arr= getSL(root);
   for(int i=0;arr[i].counter!=-1;i++)
       printf("%c %d \n",arr[i].chr,arr[i].counter);
   free(arr);
   destroy(root);
   return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
void fill_two(int* start,int* end){
for(int* i=end;i-start<*(start-1)-1;i++)
   *i=*end;
}
fill(int** All){
int**    i=All;
   while(*i){
      fill_two(*i,*(i+1));
      i=i+2;
   }
}
        int main(){
         int A[]={5,-5,14,5,2};
         int B[4]={4,6,11};
         int C[6]={6,1,-3,4};
         int D[6]={6,2,7,1,8,2};
         int E[3]={3,15};
         int F[6]={6,4,-2};
         int *All[]={A+1,A+4,B+1,B+2,C+1,C+3,D+1,D+5,E+1,E+1,F+1,F+2,NULL};
         fill(All);
        // fill_two(B+1,B+2);
        }
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int binaryFilesCmpBySize(void* f1,void* f2){
	char c;
	FILE* P1 = *(FILE**)f1;
	FILE* P2 = *(FILE**)f2;
	rewind(P1);
	rewind(P2);
	int count1 = 0;int count2=0;
	while(!feof(P1))
	{
		fread(&c,sizeof(char),1,P1);
		count1++;
	}
	rewind(P1);
	while(!feof(P2))
	{
		fread(&c,1,1,P2);
		count2++;
	}
	rewind(P2);
	return count1-count2;

}
int filesCmpBySize(void* f1,void* f2){
	FILE* P1 = *(FILE**)f1;
	FILE* P2 = *(FILE**)f2;
	rewind(P1);
	rewind(P2);
	int count1 = 0;int count2=0;
	for(;getc(P1)!=EOF;count1++);
	rewind(P1);
	for(;getc(P2)!=EOF;count2++);
	rewind(P2);
	return count1-count2;
}
int intCmp(void* a,void* b){
	return (*(int*)a-*(int*)b);
}
int genericStrCmp( void*  P1, void*  P2){
	return strcmpi((char*)P1,(char*)P2);
}
int genericStrPointerCmp( void*   P1, void*  P2){
	return strcmpi(*(char**)P1,*(char**)P2);
}
void** gmax(void* A,void* Last,int typeSize, int (*cp)(void*, void*)){
	int max=0;
	int count=1;
	for(int i=1;(char*)A+typeSize*i<=Last;i++){
		if(cp((char*)A+max*typeSize,(char*)A+typeSize*i)<0){
			max = i ;
			count=1;
		}
		else if(cp((char*)A+max*typeSize,(char*)A+typeSize*i)==0)
			count++;
```

```
	}
	char **mat = (char **)malloc((count+1) * sizeof(char*));
	int j=0;
	for(int i=0;(char*)A+typeSize*i<=Last;i++){
		if(cp((char*)A+max*typeSize,(char*)A+typeSize*i)==0){
			mat[j]=(char*)malloc(sizeof(char*));
			mat[j++]=(char*)A+typeSize*i;
		}
		mat[j]=NULL;
	}
	return (void**)mat;
}

int main()
{

	int    a[] = {2,7,5,1,7,4,7,8};
	int** A = (int**)gmax(a,a+7,sizeof(int),intCmp);
	printf("%d ",*A[0]);
	char    b[][10] = {"xyz","abc","aaaa"};
	char** B= (char**)gmax(b,b+2,sizeof(b[0]),genericStrCmp);
	printf("%s ",B[0]);
	char    *c[] = {"xyz","abc","aaaa","xyz"};
	char*** C= (char***)gmax(c,c+3,sizeof(c[0]),genericStrPointerCmp);
	printf("%s ",*C[0]);

	FILE   *f1 , *f2 , *f3;
	f1 =fopen("1.txt","r");
	f2=fopen("2.txt","r");
	f3=fopen("3.txt","r");
	FILE  *d[] = {f1, f2, f3};
	//  printf("%d\n",filesCmpBySize(f1,f2));
	FILE *** D = (FILE***)gmax(d,d+2,sizeof(FILE*),binaryFilesCmpBySize);
	if(**D==f1)
		printf("1.txt");
	else if(**D==f2)
		printf("2.txt");
	else if(**D==f2)
		printf("3.txt");
	else
		printf("wierd");


	int i=0;
	while(A[i]!=NULL){
		free(A[++i]);
	}
	free(A);
	i=0;
	while(C[i])
		free(C[++i]);
	free(C);
	i=0;
	while(B[i])
		free(B[++i]);
	free(B);
	i=0;
	while(D[i])
```

```
            free(D[++i]);
        free(D);
        return 0;
}
```

Complicated declaratrion

```
int * (* (*p) (int) ) [10];
```

p * (int) * [10] int*

p מצביע על פונקציה שמקבל מספר שלם ומחזירה מצביעה למערך בגודל של 10 של מצביעים על שלם.

void a(int (*[])(void (*) ()));

declare a as function (array of pointer to function (pointer to function returning void) returning int) returning void

int *(*a[5])(void (*)());

a[5] * (void*) int *

a מערך בגודל חמש ל מצביעים ל void ומחזיר מצביע ל int.

int * a(int *[]);

a פונקציה שמקבל מערך של מצביעים ל int ומחזירה מצביע ל int

int *(*a[5])(void (*)());

a מערך בגודל חמש של מצביעים לפונקציה שמקבלת מצביע לפונקציה שמחזירה void ומחזירה מצביע ל int.

void (* (*p[10]) (int)) ();

p מערך בגודל 10 של מצביעים על פונקציות שמקבלות int ומחזירות מצביע לפונקציה שמקבל int ומחזירה void

```
void a(int (*[])(void  (*) ()));
```

a פונקציה שמקבל מערך של מצביעים לפונקציה שמקבלת מצביע לפונקציה (שמקבלת כלום ומחזירה כלום) ומחזירה int, a מחזירה void.

```
int   (*  a( ))[3][10];
```

a פונקציה שמחזירה מצביעה למערך בגדול 3 על 10 של int

```
void a(int (*[])(void  (*) ()));
```

a פונקציה שמקבלת מערך של מצביעים על פונקציות שמקבלות מצביע על פונקציה שלא מחזירה כלום ומחזירים int a מחזיר void