Exercice 1

1. inserer_debut, inserer_fin : pas forcément, dépend de malloc ou pas
inserer_place : connaître la donné pour trouver la place
chercher : pareil
detruire_liste : desallouer la donnée
afficher_liste : oui, format %d, %s
ecrire_liste, lire_liste : idem

2. lire
ecrire
compare
copier
dupliquer
afficher
detruire

```
void *dupliquer(const void *src);
void copier(const void *src, void *dst);
void detruire(void *data);
void afficher(const void *data);
int compare(const void *a, const void *b);
int ecrire(const void *data, FILE *f);
void *lire(FILE *f);
```

3. PElement

```
typedef struct _element *PElement;
typedef struct _element {
void *data;
PElement suivant;
} Element;
```

4. pointeur sur fonction (pour associer a un pointeur)
on définit un ensemble de fonctions de manipulation de donnée (question 2)

5. PListe

```
typedef struct _liste *PListe;

typedef struct _liste {
	PElement elements;
	void *(*dupliquer)(const void *src);
	void (*copier)(const void *src, void *dst);
	void (*detruire)(void *data);
	void (*afficher)(const void *data);
	int (*comparer)(const void *a, const void *b);
	int (*ecrire)(const void *data, FILE *f);
	void *(*lire)(FILE *f);
} Liste;
```

Exercice 3 :

```
void detruire_int(void *data) {
	free(data);
}

void afficher_int(const void *data) {
	int *idata=(int *)data;
	printf("%d",*idata);
}

int ecrire_int(const void *data, FILE *f) {
	const int *idata=(const int *)data;
	return fprintf(f, "%d", *idata);
}
```

```
void * lire_int(FILE *f) {
	int i;
	int r=fscanf(f," %d",&i);
	if (r<1) return NULL;
	int *pi=(int *)malloc(sizeof(int));
	*pi=i;
	return pi;
}

int comparer_int(const void *a, const void *b) {
	int *ia=(int *)a;
	int *ib=(int *)b;
	return (*ia > *ib)-(*ia < *ib); // si *ia plus grand : 1
//si *b plus grand : 1 ensuite : soit -1, 0,1
}

void copier_int(const void *src, void *dst) {
	int *isrc=(int *)src;
	int *idst=(int *)dst;
	*idst=*isrc;
}


void *dupliquer_int(const void *src) {
	int *isrc=(int *)src;
	int *idst=malloc(sizeof(int));
	if (idst==NULL) {
		affiche_message("Erreur d'allocation");
		return NULL;
	}
	*idst=*isrc;
	return (void *)idst;
}
```

Exercice 2 :

1.

```
void inserer_debut(PListe pliste, void *data) {
        PElement newe=malloc(sizeof(Element));
        if (newe==NULL) {
        affiche_message("Erreur d’allocation");
        }
        newe->data=pliste->dupliquer(data);
        newe->suivant=pliste->elements;
        pliste->elements=newe;
        }
```

2.

```
void inserer_fin(PListe pliste, void *data){
        PElement newe=malloc(sizeof(Element));
        if (newe==NULL) {
        affiche_message("Erreur d’allocation");
        }
        newe->data=pliste->dupliquer(data);
        newe->suivant=NULL;
        }
        PElement tmp=pliste->elements;
        if(tmp==NULL) {
        pliste->elements=newe;
        }
        else {
        while(tmp->suivant!=NULL)
                tmp=tmp->suivant;
        tmp->suivant=newe;
        }
}
```

3.

```
void inserer_place(PListe pliste, void *data){
	PElement newe=malloc(sizeof(Element));
	if (newe==NULL) {
	affiche_message("Erreur d’allocation");
	}
	newe->data=pliste->dupliquer(data);
	newe->suivant=NULL;
	PElement tmp=pliste→elements;

	// si la liste est vide: ajout immediat
	if((tmp==NULL)||(pliste->comparer(data, tmp->data)<0)) {
	newe->suivant=pliste->elements;
	pliste->elements=newe;
	}
	else {
	while(tmp->suivant!=NULL) {
	int comp=pliste->comparer(data,tmp->suivant->data);
	if (comp<=0) {

	// insertion avant le suivant
	newe->suivant=tmp->suivant;
	tmp->suivant=newe;
	return;
	}
	// sinon, on passe au suivant
	tmp=tmp->suivant;
	}
// on est arrive a la fin sans trouver la bonne place: la bonne place est donc a
la fin
	tmp->suivant=newe;
	}
	return;
}
```

4.

```
PElement chercher_liste(PListe pliste, void *data) {
PElement tmp=pliste->elements;
        while(tmp) {
        if (pliste->comparer(data, tmp->data)==0)
                return tmp;
        }
        tmp=tmp->suivant;
        }
        return NULL;
}
```

5.

```
void detruire_liste(PListe pliste) {
        PElement tmp=pliste->elements;
        PElement tmp2;

        while (tmp) {
                tmp2=tmp->suivant;
                pliste->detruire(tmp->data);
                free(tmp);
                tmp=tmp2;
        }
        free(pliste);
}
```

6.

```
void afficher_liste(PListe pliste) {
PElement tmp=pliste->elements;
while (tmp) {
	pliste->afficher(tmp->data);
	printf("\n");
	tmp=tmp->suivant;
	}
}
```

7.

```
void ajouter_liste(PListe pliste, int nb_data, ...) {
	va_list args;
	va_start(args,nb_data);
	int i;
	for (i=0;i<nb_data;i++) {
		void *data=va_arg(args,void *);
	inserer_fin(pliste,data);
	}
	va_end(args);
}
```

8.

```
void map(PListe pliste, void (*fonction)(void *data, void *oa), void *optarg)
{
	PElement elt=pliste->elements;
	while(elt) {
	fonction(elt->data,optarg);
	elt=elt->suivant;
	}
}
```

9.

```
int ecrire_liste(PListe pliste, const char *nom_fichier) {
	FILE *f=fopen(nom_fichier,"w");
	if (f==NULL) {
	printf("Erreur lors de l’ouverture du fichier %s\n",nom_fichier);
	return 0;
	}

	PElement tmp=pliste->elements;
	while (tmp) {
	pliste->ecrire(tmp->data,f);
		fprintf(f,"\n");
		tmp=tmp->suivant;
	}
	fclose(f);
	return 1;
}
```

10.

```
int lire_liste(PListe pliste, const char * nom_fichier) {
	FILE *f=fopen(nom_fichier, "r");
	if (f==NULL) {
	printf("Erreur lors de l’ouverture du fichier %s\n",nom_fichier);
	return 0;
	}
	void *data=pliste->lire(f);
	while(data!=NULL) {
	inserer_fin(pliste,data);
	free(data);
	data=pliste->lire(f);
	}
	fclose(f);
	return 1;
	}
```