Rappel des épisodes précédents

Langage C

- Types et tailles
- Lire depuis le clavier
- Écrire à l’écran
- Chaînes de caractères
- Pointeurs
- Structures
- Fonctions

Types et tailles

- **int** 4 octets (32 bits)
  - unsigned
  - short 2 octets (16 bits)
  - long 8 octets (64 bits)
- **float** 4 octets (32 bits)
- **double** 8 octets (64 bits)
- **char** 1 octet
  - unsigned
- **Pas de type « chaîne de caractères »**
  - utiliser **char[*nombreDeCaracteresPossibles*]**

Lire depuis le clavier

- **scanf**("%type",&nomDeLaVariableCorrespondantAuType) ;
  - type
    - i,d ou u pour des entiers
    - f,e ou g pour des réels
    - c pour **UN** caractère

- **scanf**("%s",nomDeLaVariableChaine) ; **Attention scanf s'arrête si un espace dans la chaîne**
- **gets**(nomDeLaVariableChaine)
- Exemple :

```
int age ;
char nom[255] ;
scanf("%d",&age) ;
scanf("%s",nom) ;
gets(nom) ;
```

Écrire à l'écran

- **printf**(" du texte %type1, du texte %type2 …",variable1,variable2) ;
- Exemple

```
int age=18 ;

float taille=1.75 ;

char note=15;

char nom[255] ;

strcpy(nom, "toto") ;

printf("bonjour %s, vous avez %d ans, vous mesurez %fm
vous avez la note de %c \n",nom,age,taille,note) ;
```

Chaînes de caractères

- **Affectation**

  strcpy(variableDestination,chaineSource)
- **Comparaison**

  strcmp(chaine1,chaine2)
- **Longueur**

  strlen(chaine)
- Exemple

```
char chaine[NBMAXCAR];
strcpy(chaine,"coucou");
if (strcmp(chaine,"coucou")==0)
{
  printf("la chaine %s est bien coucou",chaine);
}
else
{
  printf("la chaine %s n'est pas coucou, mais a une taille de %d
  caracteres",chaine,strlen(chaine));
}
```

Pointeurs

- type *nomVariablePointeur
- Deux opérateurs disponibles : & et *
  - & permet d'obtenir l'adresse d'une case mémoire
  - * permet de récupérer la valeur pointée par le pointeur

Un pointeur est une variable contenant l'adresse mémoire d'une valeur typée.

val : valeur de la variable

&val : adresse de la variable val

*ptrVal : valeur se trouvant à l'adresse donnée par ptrVal

Structures

- ```
  struct nomDeLaStructure{
     type nomDeChamps1;
     type nomDeChamps2;
     ...
  };
  ```
- Accès aux champs pour une variable statique

  ```
  nomVariable.nomDuChamp
  ```
- Accès aux champs pour une variable dynamique(pointeur)

  ```
  nomVariable->nomDuChamp
  ```
- Exemple

```
struct cercle c1;
struct cercle *c2;
c1.x=10;
c1.y=7;
c1.r=36;
c2=&c1
printf("x=%d y=%d",c2->x,c2->y);
c2->r=15;
```

```
struct cercle{
 int x;
 int y;
 int r;
};
```
```

fonctions

- `typeDeRetour nomFonction(typeParam1 nomParam1, typeParam2 nomParam2, etc) ;`
- Les paramètres sont considérés comme des variables correctement initialisées.
- Si la fonction retourne quelquechose:

```
typeDeRetour nomFonction(typeParam1 p1, typeParam2 p2, etc)
{
        typeDeRetour retour ;
        code de la fonction
        return retour ;
}
```

- Appel de la fonction:

```
typeDeRetour retour;
retour= nomFonction(...);
```