IF 5021 Algoritma dan Pemrograman Kuliah Minggu ke-5
Topik : konsep pointer dan pointer dalam bahasa C

Objektif :
Setelah mengikuti kuliah ini, mahasiswa :

1. Memahami tentang nilai dan alamat
    a. Type Pointer (nama, domain nilai, nilai konstanta, operator)
    b. Pointer arithmetik dalam bahasa C
2. Memahami "type" pointer:
    a. Tiga tahap pemakaiannya : deklarasi, alokasi dan inisialisasi, sehingga dapat memakai type pointer tanpa menimbulkan persoalan memory.
    b. Pointer to int, to const, to functions, dan kombinasinya
    c. Array of pointers
3. Memahami dan mampu memakai pointer untuk indirrect addressing
4. Memahami pemakaian pointer sebagai sarana parameter passing by reference, mekanisme eksekusinya, dan memakainya dengan tepat
5. Memahami implementasi array dalam bahasa C:
    a. Deklarasi array (satu dimensi, dua dimensi), secara "statik"/fix maupun dinamik (ditentukan kemudian ukurannya saat alokasi) /pointer
    b. Untuk array dengan deklarasi pointer, alokasi dan inisialisasi/pengisian nilai
    c. Mampu melakukan proses pemakaian array dinamik dengan baik.
6. Reserved :
    a. Mengkode atau memakai algoritma search dan sort nilai dalam array
    b. Memahami konsep linked list dan dapat memakai sebagai salah satu struktur data dinamik jika diperlukan.

Bacaan :

1. Buku Kernighan & Ritchie : C Programming Language, Prentice Hall Software Series
2. Diktat Algoritma dan Pemrograman, topik terkait
3. Diktat struktur data, topik terkait (pointer, linked list)

Deklarasi, alokasi, inisialisasi pointer, static variable

```
/* file : pointer.c */
#include <stdlib.h>
#include <stdio.h>
int main () {
  int x=0;   /* deklarasi,alokasi otomatis dan inisialisasi */

  int y; /* deklarasi dan alokasi memori dilakukan */
  y=0; /* instruksi assignment untuk inisialisasi nilai*/

  static int z=0; /* sekali diinisialisasi */
  int * ptrx; /* pointer ke integer */
  ptrx = (int *) malloc (sizeof (int)); /* alokasi */
  *ptrx=4; /* mengisi nilai - inisialisasi nilai */
  printf("%d\n", *ptrx);

  int *ptr= (int *) malloc (sizeof (int));
  *ptr=10;
  printf("%d\n", *ptr);

  return 0;
}
```

Pointer dan passing parameter

Berikut ini resume mengenai pointer (address) dan nilai, dan dampak dari passing parameter berupa nilai serta address

Menukar dua buah nilai

Fungsi	Pemanggilan

`void swap1(int a, int b) {`
`  int temp = a;`
`  a = b;`
`  b = temp;`
`/* nilai a dan b tertukar */`
`}`
	`int x = 42;`
	`int y = 17;`
	`swap1(x, y);`
	`/* nilai x dan y tetap*/`

`void swap2(int* a, int* b) {`
`  int temp = *a;`
`  *a = *b;`
`  *b = temp;`
`/* nilai *a dan *b tertukar */`
`}`
	`int x = 42;`
	`int y = 17;`
	`swap2(&x, &y);`
	
	`/* nilai x dan y tertukar */`

Menukar dua buah address

Fungsi	Pemanggilan

`void swap3(int* a, int* b) {`
`  int* temp = a;`
`  a = b;`
`  b = temp;`
`/* nilai a (address) dan b (address) tertukar */`
`}`
	`int x = 42;`
	`int y = 17;`
	`int* xp = &x;`
	`int* yp = &y;`
	`swap3(xp, yp);`
	`/* xp dan yp tidak tertukar */`

`void swap4(int* a, int* b) {`
`  int temp = *a;`
`  *a = *b;`
`  *b = temp;`
 	
`}`
	`int x = 42;`
	`int y = 17;`
	`int* xp = &x;`
	`int* yp = &y;`
	`swap4(xp, yp);`
	`/* yang ditukar :nilai yang disimpan dalam xp dan yp */`

`void swap5(int** a, int** b) {`
`  int* temp = *a;`
`  *a = *b;`
`  *b = temp;`
 	
`}`
	`int x = 42;`
	`int y = 17;`
	`int* xp = &x;`
	`int* yp = &y;`
	`swap5(&xp, &yp);`
	`/* xp dan yp tertukar */`

Pointer dan Array

Bacalah dari buku Kernighan & Ritchie dan contoh=contoh array dalam Diktat program kecil

```
int *T;
T = (int *) malloc (10*sizeof (int));

/* cara mengacu elemen array */
/*  *T                T[0] */
/*  *(T+1)            T[1] */
/*  *(T+2)            T[2] */
/*  *(T+3)            T[3] */
```

Lihat contoh Program kecil (akan didemokan) : deklarasi array, array sebagai parameter, persoalan test alokasi

Array of pointers
An array of pointers can be declared as :

<type> *<name>[<number-of-elements];

For example : **`char *ptr[3];`**

```
/* File ArrPointers.c */
#include<stdio.h>
int main(void)
{
    char *p1 = "ITB";
    char *p2 = "UI";
    char *p3 = "ITS";

    char *arr[3];
    arr[0] = p1;
    arr[1] = p2;
    arr[2] = p3;

   printf(" p1 = [%s] \n",p1);
   printf(" p2 = [%s] \n",p2);
   printf(" p3 = [%s] \n",p3);

   printf(" arr[0] = [%s] \n",arr[0]);
   printf(" arr[1] = [%s] \n",arr[1]);
   printf(" arr[2] = [%s] \n",arr[2]);

   return 0;
}
```

Pointer to constant
A pointer to a constant is declared as :
`const <type-of-pointer> *<name-of-pointer>;`

```
/* File PtrToConst.c */
#include<stdio.h>
int main(void)
{
  char ch = 'c';
  const char *ptr = &ch;
  /* A constant pointer 'ptr' pointing to 'ch' */

*ptr = 'a';//Wrong,Cannot change the value at address pointed by 'ptr'.

    return 0;
}
```

Constant pointer

<type-of-pointer> *const <name-of-pointer>

A pointer is said to be constant pointer when the address its pointing to cannot be changed
A constant pointer, if already pointing to an address, cannot point to a new address.

```
/* File : ConstPtr.c */
#include<stdio.h>
int main(void)
{
    char ch = 'c';
    char c = 'a';

    char *const ptr = &ch; // A constant pointer
    ptr = &c; // Trying to assign new address to a constant pointer. WRONG!!!!

    return 0;
}
```

Pointer to function

A function pointer can be declared as :

<return type of function> (*<name of pointer>) (type of function arguments)

```
/* File PtrToF.c */

#include<stdio.h>
int func (int a, int b)
{
    printf("a = %d\n",a);
    printf("b = %d\n",b);
    return 0;
}

int main(void)
{
    int(*fptr)(int,int); // Function pointer
    fptr = func; // Assign address to function pointer

    printf ("%d \n", func (2,3));
    printf ("%d \n", fptr(2,3));

    return 0;
}
```

Array of Pointer to function

```
/* File : pointf1.c */
/* Pointer ke function */
/* prototype */
void f1 (void);
void f2 (void);
void f3 (void);
void f4 (void);
/* KAMUS GLOBAL */
#define true 1
#define false 0
int quit = false;
int main ()
{/* KAMUS LOKAL */
/* Definisi tabel yang elemennya adalah pointer ke fungsi */
/* Elemen tabel yang ke-i akan mengakses fungsi ke-i */

  void (*tab[4]) () =   {f1, f2, f3, f4};
  printf ("Pointer to function \n");
  tab[0] ();
  tab[1] ();
  tab[2] ();


/* BODY FUNGSI - tidak dituliskan di teks ini */
/* [cut] */
```

Array Dua dimensi

Array dua dimensi : lihat contoh-contoh program kecil

```
/* File matriks.c */
#include <stdio.h>

int main() {
int Mx [3][2]= {{1,2},{3,4},{5,6} };
int i,j;
  for (i=0; i<3; i++)
  {
   for (j=0; j<2; j++)
   { printf ("Mx[%d][%d] = %d",i,j, Mx[i][j]);
   }
   printf("\n");
  }
 return 0;
}
```

```c
/* file : matriks3.c */
#include <stdlib.h>
#include <stdio.h>
int main (){
/* deklarasi matriks dengan baris dan kolom variabel */
/* dengan elemen bertype int*/
  int ** Mat;
  int NBARIS;
  int i,j;

  /* baca banyaknya baris, alokasi baris */
  printf("Masukkan banyaknya baris "); scanf("%d", &NBARIS);
  Mat = (int **)malloc (NBARIS * sizeof (int *));
  for(i=0;i< NBARIS ;i++){
      Mat[i] = (int *)malloc ((i+1) * sizeof (int));
      for(j=0; j<= i; j++)
      {
        Mat [i][j] = i+j;
        printf ("Mat[%d][%d] : %d - \n",i,j,(Mat [i][j]));
      }
  }
 return 0;
}
```

```c
/* File matriks4.c */
/* hati-hati matriks dapat “dilinearkan” */

#include <stdlib.h>
#include <stdio.h>
int main (){
int i,j;
   int M[4][4] = { {1,2,3,4},
                   {5,6,7,8},
                   {9,10,11,12},
                   {13,14,15,16}
                 } ;

  for (i=0;i<4;i++) {
  for (j=0;j<4;j++) {
   {printf("%d ", M[i][j]); }}
   printf ("\n");
  }

  for (i=0;i<16;i++) {
   printf("%d - %d\n",i, *M[0]+i);
   }

 return 0;
}
```

Tambahan konvensi pemrograman :

Saat membuat deklarasi array dinamik yang belum dialokasi, sebutkan bahwa deklarasi tersebut adalah array

```c
int * T; /* array yg akan dialokasi dari nilai N yang dibaca */
```