DASAR PEMROGRAMAN

Pertemuan V


Pengantar Pemrograman
Perulangan
Subprogram II
Matriks
Fungsi Rekursif
Sorting II
Searching II
File II
Percabangan
Subprogram
Array
UTS
Sorting I
Searching I
File I
UAS

Tujuan

- Mahasiswa memahami makna dan kegunaan subprogram dalam bentuk fungsi dan prosedur
- Mahasiswa dapat menggunakan notasi fungsi dan prosedur dengan benar dan menggunakannya dalam program
- Mahasiswa dapat membuat program dengan menggunakan fungsi dan prosedur

Materi

PROSEDUR

PARAMETER INPUT DAN OUTPUT

FUNGSI OVERLOADED

PROSEDUR

PROSEDUR

- Prosedur adalah rangkaian perintah yang diberi nama untuk melakukan pekerjaan tertentu.
- Prosedur menerima argument dan tidak mengembalikan sebuah hasil.

CONTOH

- Menukar 2 buah bilangan → tidak bisa dilakukan fungsi karena hasil akhirnya ada 2 nilai yang berubah
- Menyimpan nilai ke file → hasil dari pekerjaan ini bukan nilai, tapi data menjadi tersimpan secara permanen.

MEMBUAT PROSEDUR

1. Definisikan prosedur
   - Tentukan nama prosedur
   - Tentukan formal parameter
   - Tentukan keadaan awal dan akhir
2. Buat realisasi prosedur
   - Buat algoritma untuk melakukan hal tertentu
3. Panggil prosedur di main function atau fungsi lainnya
   - Panggil prosedur menggunakan actual parameter

```cpp
#include <iostream>

using namespace std;

void fxkuadrat(int x, int *y);

int main()
{
    int x, hasil, fx;

    x = 3;
    fxkuadrat(x,&hasil);
    fxkuadrat(10,&fx);
    cout << hasil << " " << fx ;

    return 0;
}

void fxkuadrat(int x, int *y)
{
    *y = x * x + 3 * x - 5;
}
```

1. Deklarasikan prosedur
   - Tentukan nama prosedur
   - Tentukan formal parameter
   - Tentukan keadaan awal dan akhir
2. Buat realisasi prosedur
   - Buat algoritma untuk melakukan hal tertentu
3. Panggil prosedur di main function atau fungsi lainnya
   - Panggil prosedur menggunakan actual parameter

MENDEFINISIKAN PROSEDUR

Prosedur harus didefinisikan terlebih dahulu sebelum dipanggil di tempat lain.

Sebuah prosedur bisa tidak membutuhkan argument atau bisa memiliki banyak argument.

Prosedur yang memiliki banyak argument dipisahkan dengan koma.

SINTAKS MENDEFINISIKAN PROSEDUR

```
void nama_prosedur(tipe_data1 parameter_formal1,
tipe_data2 parameter_formal2, … , tipe_dataN
parameter_formalN);
```

```
void fxkuadrat(int x, int *y);
```

MEREALISASIKAN PROSEDUR

Realisasi prosedur berisi algoritma untuk melakukan pekerjaan tertentu.

Karena prosedur tidak mengembalikan nilai, jika ada nilai yang akan digunakan oleh pemanggil maka menggunakan parameter output (bahas berikutnya).

SINTAKS MEREALISASIKAN PROSEDUR

```
void nama_prosedur(tipe_data1 parameter_formal1, tipe_data2
parameter_formal2, … , tipe_dataN parameter_formalN)
{
     // algoritma untuk melakukan sesuatu

}
```

```
void fxkuadrat(int x, int *y)
{
     *y = x * x + 3 * x - 5;
}
```

MEMANGGIL PROSEDUR

Prosedur bisa dipanggil sebanyak apapun dibutuhkan.

Pemanggilan prosedur cukup dengan menuliskan nama prosedur beserta parameter-parameternya.

SINTAKS MEMANGGIL PROSEDUR

```
nama_prosedur(parameter_actual1, … , parameter_actualN)
```

```
fxkuadrat(x,&hasil);
fxkuadrat(10,&fx);
```

PARAMETER FORMAL DAN AKTUAL

- Setiap parameter aktual berasosiasi dengan parameter formal sesuai urutannya.
- Tipe data parameter actual harus sesuai dengan tipe data formal parameter yang berasosiasi

```
void fxkuadrat(int x, int *y);
```

```
fxkuadrat(x,&hasil);
```

PASSING PARAMETER

RATIONALE

Variabel Global adalah **bad practice**, sehingga sedapat mungkin dihindari penggunaanya.

Masalah dari sebuah variabel global adalah setiap bagian dari program memiliki akses ke variabel tersebut. Hal tersebut akan mempersulit kita, terutama pada saat proses *debug,* untuk mencari bagian program mana yang melakukan manipulasi terhadap variabel tersebut.

AUTHENTIC DEFINITION

PASS BY VALUE

The function will have its own copy of the argument - its *value* is copied. The caller and callee have two independent variables with the same value. If the callee modifies the parameter variable, the effect is not visible to the caller.

PASS BY REFERENCE

The parameter inside the function refers to the same object that was passed in. The caller and the callee use the same variable for the parameter. If the callee modifies the parameter variable, the effect is visible to the caller's variable.

The "pass by value vs. pass by reference" distinction as defined in the computer science theory is now “obsolete” because the technique originally defined as "pass by reference" has since fallen out of favour and is seldom used now.

Newer languages tend to use a different (but similar) pair of techniques to achieve the same effects which is the primary source of confusion.

A secondary source of confusion is the fact that in "pass by reference", "reference" has a narrower meaning than the general term "reference" (because the phrase predates it)

REFERENCE VARIABLE

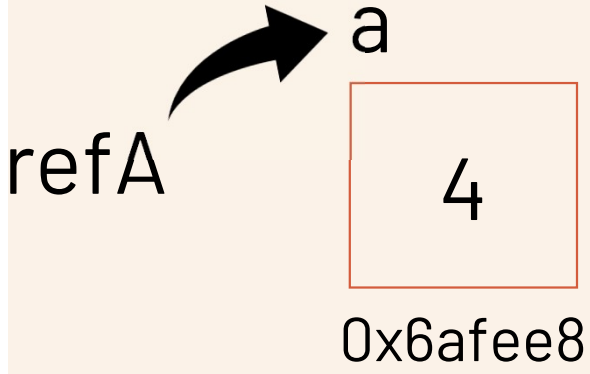
- A reference variable is an alias, that is, another name for an already existing variable.
- Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
- The operator used by the reference variable is '&'.
- Variable and its reference variable refers to the same memory location

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 4;
    int &refA = a;

    cout << "value a = " << a << endl;
    cout << "value reference a = " <<refA << endl;

    refA = 100;

    cout << "value a = " << a << endl;
    cout << "value reference a = " <<refA << endl;

    return 0;
}
```

```
value a = 4
value reference a = 4
value a = 100
value reference a = 100
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 4;
    int &refA = a;

    cout << "value a = " << a << endl;
    cout << "value reference a = " <<refA << endl;

    refA = 100;

    cout << "value a = " << a << endl;
    cout << "alamat  a = " <<&a << endl;
    cout << "value reference a = " <<refA << endl;
    cout << "alamat reference a = " <<&refA << endl;


    return 0;
}
```

```
value a = 4
value reference a = 4
value a = 100
alamat  a = 0x6afee8
value reference a = 100
alamat reference a = 0x6afee8
```



POINTER VARIABLE

- A “pointer” is a variable that holds the memory location of another variable.
- The operators used by the pointer variable is * .
- The declaration of pointer variable contains the base data type followed by the ‘*’ sign and the variable name.

Name	Type	Value
sum	int (4 bytes)	255 (000000FF)
age	short int (2 bytes)	10 (000A)
average	double int (8 bytes)	2305843009213693951 (1FFFFFFFFFFFFFFF)
pointerSum	int *	90000000

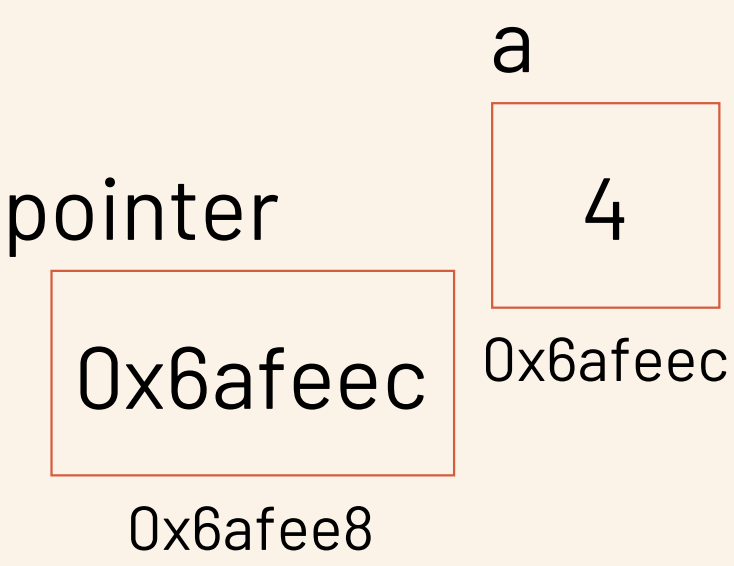
	ADDRESS	CONTENT
sum	**90000000**	00
	90000001	00
	90000002	FF
	90000003	FF
age	**90000004**	00
	90000005	0A
average	**90000006**	FF
	90000007	FF
	90000008	FF
	90000009	FF
	9000000A	FF
	9000000B	FF
	9000000C	FF
	9000000D	FF
pointerSum	**9000000E**	90
	9000000F	00
	90000010	00
	90000011	00

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 4;
    int *pointer = &a;

    cout << a << endl;
    cout << pointer << endl;
    cout << *pointer << endl;

    return 0;
}
```

```
4
0x6afee8
4

Process returned 0 (0x0)   execution time : 0.202 s
Press any key to continue.
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 4;
    int *pointer = &a;
    int b = 5;

    cout<<"value a = "<< a << endl;
    cout<<"alamat a = "<< &a << endl;
    cout<<"value pointer = "<<pointer << endl;
    cout<<"alamat pointer = "<< &pointer << endl;
    cout<<"value yang ditunjuk pointer = "<<*pointer << endl;

    pointer = &b;

    cout<<"value a = "<< a << endl;
    cout<<"alamat a = "<< &a << endl;
    cout<<"value b = "<< b << endl;
    cout<<"alamat b = "<< &b << endl;
    cout<<"value pointer = "<< pointer << endl;
    cout<<"alamat pointer = "<< &pointer << endl;
    cout<<"value yang ditunjuk pointer = "<<*pointer << endl;

    return 0;
}
```

```
value a = 4
alamat a = 0x6afeec
value pointer = 0x6afeec
alamat pointer = 0x6afee8
value yang ditunjuk pointer = 4
value a = 4
alamat a = 0x6afeec
value b = 5
alamat b = 0x6afee4
value pointer = 0x6afee4
alamat pointer = 0x6afee8
value yang ditunjuk pointer = 5
```



REFERENCE

- The reference is an alias for a variable.
- The reference variable returns the address of the variable preceded by the reference sign ‘&’.
- The operator for reference is &
- The reference variable can never refer to NULL.
- An uninitialized reference can never be created.
- The reference variable can only be initialized at the time of its creation.
- The reference variable can never be reinitialized again in the program.

POINTER

- The pointer is the memory address of a variable.
- The pointer variable returns the value located at the address stored in pointer variable which is preceded by the pointer sign '*’.
- The operator for pointer is ‘*”.
- The pointer variable can refer to NULL.
- An uninitialized pointer can be created.
- The pointer variable can be initialized at any point of time in the program.
- The pointer variable can be reinitialized as many times as required.

PASSING PARAMETER IN C++

Pass by Value

Pass by Reference

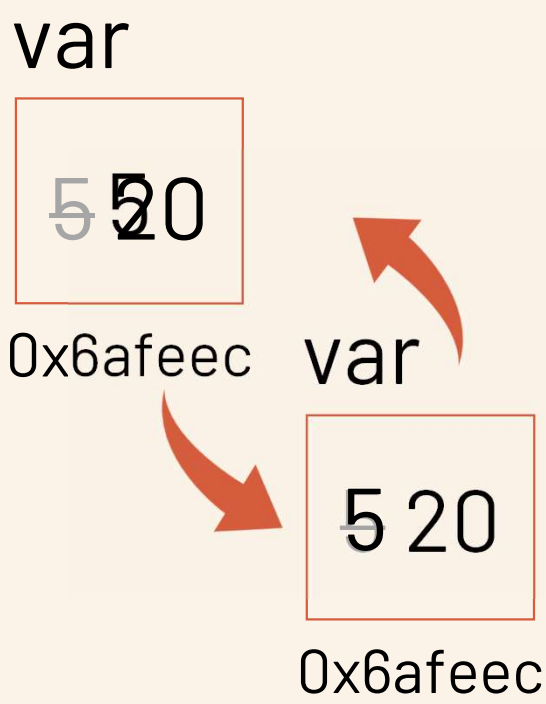
Pass by Pointer

```cpp
#include <iostream>

using namespace std;

void passByValue (int var);
void passByReference (int &refVal);
void passByPointer (int *pointer);

int main()
{
    int var = 5;
    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;

    passByValue(var);
    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;

    return 0;
}

void passByValue (int var)
{
    var = 10;
    cout << "value var di subprogram= "<< var << endl;
    cout << "alamat var di subprogram= "<<&var << endl;

}
```

```
value var di main program= 5
alamat var di main program= 0x6afeec
value var di subprogram= 10
alamat var di subprogram= 0x6afed0
value var di main program= 5
alamat var di main program= 0x6afeec
```

```cpp
#include <iostream>

using namespace std;

void passByValue(int var);
void passByReference(int &var);
void passByPointer(int *pointer);

int main()
{
    int var = 5;
    int &refVar = var;

    cout << "value var di main program = " << var << endl;
    cout << "alamat var di main program = " << &var << endl;

    passByReference(refVar);

    cout << "value var di main program = " << var << endl;
    cout << "alamat var di main program = " << &var << endl;

    return 0;
}

void passByReference(int &var)
{
    var = 20;
    cout << "value var di subprogram = " << var << endl;
    cout << "alamat var di subprogram = " << &var << endl;
}
```

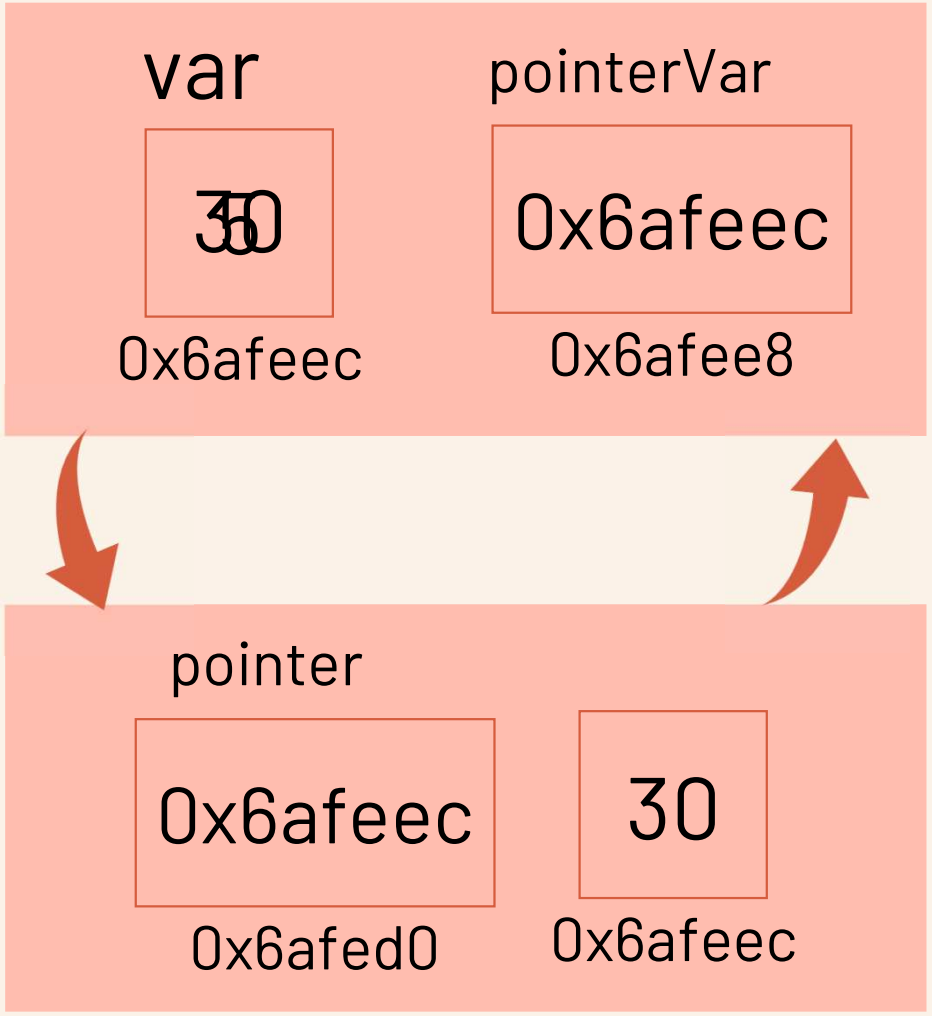
```
value var di main program= 5
alamat var di main program= 0x6afeec
value var di subprogram= 20
alamat var di subprogram= 0x6afeec
value var di main program= 20
alamat var di main program= 0x6afeec
```



```cpp
#include <iostream>

using namespace std;

void passByValue (int var);
void passByReference (int &var);
void passByPointer (int *pointer);

int main()
{
    int var = 5;
    int *pointerVar = &var;

    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;
    cout << "value pointerVar di main program= "<< pointerVar << endl;
    cout << "alamat pointerVar di main program= "<<&pointerVar << endl;

    passByPointer(pointerVar);
    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;
    cout << "value pointerVar di main program= "<< pointerVar << endl;
    cout << "alamat pointerVar di main program= "<<&pointerVar << endl;

    return 0;
}

void passByPointer (int *pointer)
{
    *pointer = 30;
    cout << "value pointer di subprogram= "<< pointer << endl;
    cout << "alamat pointer di subprogram= "<<&pointer << endl;

}
```

```
value var di main program= 5
alamat var di main program= 0x6afeec
value pointerVar di main program= 0x6afeec
alamat pointerVar di main program= 0x6afee8
value pointer di subprogram= 0x6afeec
alamat pointer di subprogram= 0x6afed0
value var di main program= 30
alamat var di main program= 0x6afeec
value pointerVar di main program= 0x6afeec
alamat pointerVar di main program= 0x6afee8
```

```cpp
#include <iostream>

using namespace std;

void passByValue (int var);
void passByReference (int &var);
void passByPointer (int *pointer);

int main()
{
    int var = 5;
    int *pointerVar = &var;

    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;
    cout << "value pointerVar di main program= "<< pointerVar << endl;
    cout << "alamat pointerVar di main program= "<<&pointerVar << endl;

    passByPointer(pointerVar);
    cout << "value var di main program= "<< var << endl;
    cout << "alamat var di main program= "<<&var << endl;
    cout << "value pointerVar di main program= "<< pointerVar << endl;
    cout << "alamat pointerVar di main program= "<<&pointerVar << endl;

    return 0;
}

void passByPointer (int *pointer)
{
    *pointer = 30;
    cout << "value pointer di subprogram= "<< pointer << endl;
    cout << "alamat pointer di subprogram= "<<&pointer << endl;

}
```



OVERLOADED FUNCTION

Function overloading is the ability to create multiple functions of the same name with different implementations.

Compiler will call appropriate function based in the parameter provided by the caller.

RULES

- The same function name is used for more than one function definition
- The functions must differ either by the number of parameters or types of their parameters

```cpp
#include <iostream>

using namespace std;

int tambah(int x, int y);
int tambah(int x, int y, int z);
double tambah(double x, double y);

int main()
{
    cout << tambah(10,20)<<endl;
    cout << tambah(10,20,30)<<endl;
    cout << tambah(10.5,20.9)<<endl;

    return 0;
}
```

```
30
60
31.4
```

```cpp
int tambah(int x, int y)
{
    int sum = x + y;
    return(sum);
}

int tambah(int x, int y, int z)
{
    int sum = tambah(tambah(x,y), z);
    return(sum);
}

double tambah(double x, double y)
{
    double sum = x + y;
    return(sum);
}
```

PARAMETER INPUT DAN OUTPUT

PARAMETER INPUT

Parameter yang pada saat pemanggilan prosedur berisi suatu nilai.

Semua parameter pada fungsi adalah parameter input

PARAMETER OUTPUT

Parameter yang pada saat pemanggilan prosedur tidak memiliki nilai.

Parameter ini menyimpan nilai hasil komputasi dalam prosedur yang akan digunakan oleh pemanggil.

Parameter output pada definisi (parameter formal) menggunakan tanda *.

Parameter output pada pemanggilan (parameter aktual) menggunakan tanda &.

```cpp
#include <iostream>

using namespace std;

void fxkuadrat(int x, int *y);

int main()
{
    int x, hasil, fx;

    x = 3;
    fxkuadrat(x,&hasil);
    fxkuadrat(10,&fx);
    cout << &hasil << " " << &fx << endl;

    return 0;
}

int fxkuadrat(int x, int *y)
{
    *y = x*x+3*x-5;
}
```

LATIHAN SOAL

LATIHAN 1

Buatlah sebuah prosedur yang menerima sebuah string, misalnya `str`, dan menghasilkan tampilan:

Hello, str

Panggil prosedur tersebut di program utama

LATIHAN 2

Buatlah prosedur untuk menukar dua harga yang disimpan dalam dua nama harga1 dan harga2

Initial State :

diberikan nilai integer a harga1 = 1 dan harga2 = 5

Final State :

harga1 = 5 dan harga2 = 1

LATIHAN 3

Dengan memanfaatkan prosedur Tukar di Latihan 2, buatlah sebuah program utama yang digunakan untuk menukar 3 buah bilangan, misalnya angka1, angka2, angka3 sehingga angka1 berisi nilai pada angka3, angka2 berisi nilai pada angka1, dan angka3 berisi nilai pada angka2. Ketiga bilangan dibaca dari keyboard

Contoh:

angka1 = 3; angka2 = 2; angka3 = 1 → ditukar menjadi : angka1 = 1; angka2 = 3; angka3 = 2