

## RINGKASAN: Pointer di C dan C++

### A. Pengenalan Pointer di C dan C++

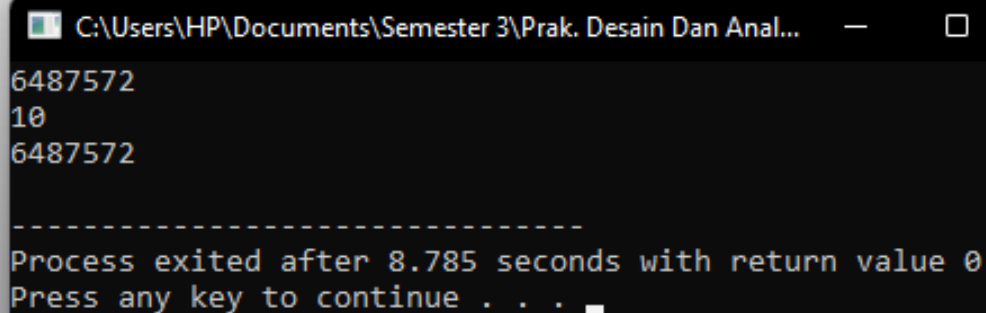
Pointer adalah variabel yang menyimpan alamat dari variabel lainnya. Cara mendeklarasi pointer dalam C/C++:

- `int a;` // kita mendeklarasikan variabel `a` bertipe integer.
- `int *p;` // dengan memberikan lambang asterisk sebelum nama variabel maka kita sudah membuat suatu variabel pointer yaitu `*p`.
- `p = &a;` // pendeklarasian seperti ini akan membuat alamat dari variabel integer `a` disimpan ke dalam variabel pointer `p` sehingga variabel `p` sekarang menyimpan alamat dari variabel `a`.

Berikut adalah contoh penggunaan pointer untuk menyimpan sebuah alamat variabel di C:

pointer1.cpp

```
1  #include <stdio.h>
2
3  int main() {
4      int a;
5      int *p; //inisialisasi variabel pointer
6      a = 10;
7      p = &a; //&a merupakan alamat dari a
8      printf("%d\n", p); //alamat yang disimpan oleh p
9      printf("%d\n", *p); //nilai pada alamat yang disimpan
10     printf("%d\n", &a);
11 }
```

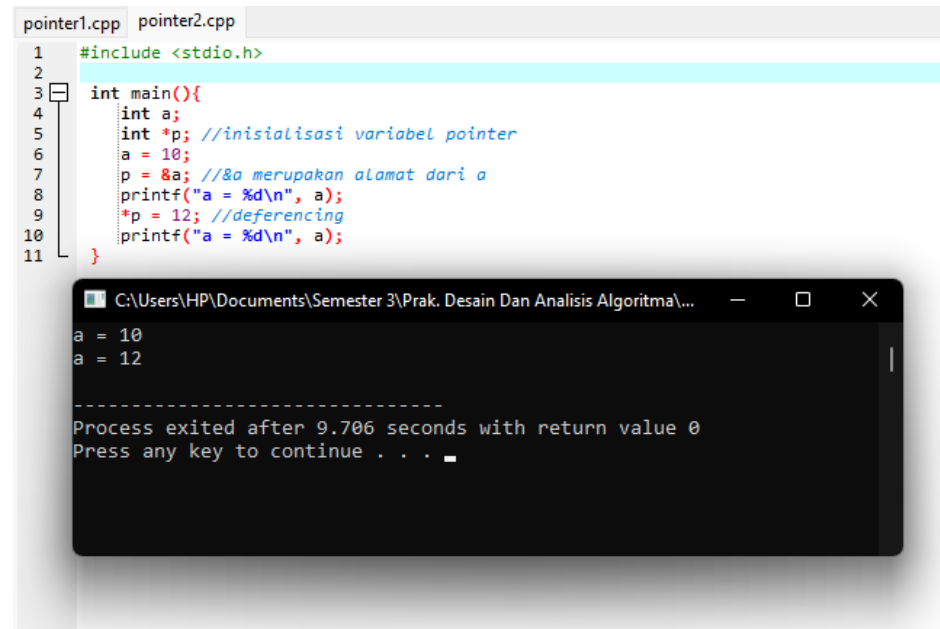


```
C:\Users\HP\Documents\Semester 3\Prak. Desain Dan Anal...
6487572
10
6487572

-----
Process exited after 8.785 seconds with return value 0
Press any key to continue . . .
```

Contoh output dari kode di atas menunjukkan alamat yang disimpan melalui pointer dan melalui akses alamat variabel, juga nilai dari variabel melalui pointer.

Pointer dapat juga memanipulasi data dari alamat variabel yang disimpan, konsep ini dikenal dengan dereferencing. Berikut contohnya:

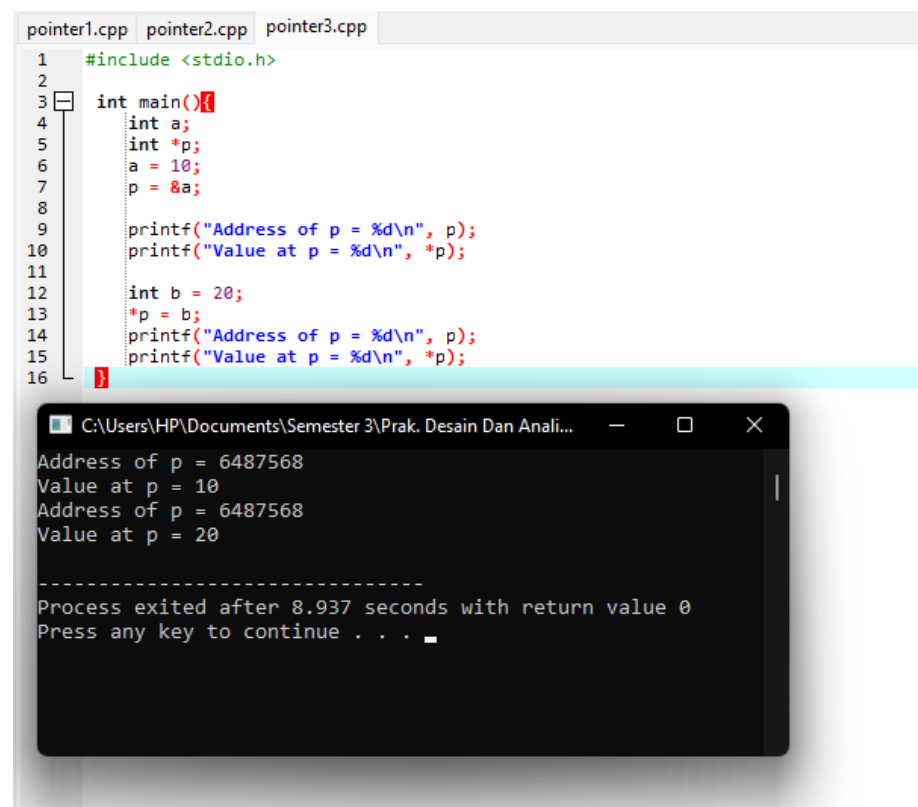


```
pointer1.cpp pointer2.cpp
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     int *p; //inisialisasi variabel pointer
6     a = 10;
7     p = &a; //&a merupakan alamat dari a
8     printf("a = %d\n", a);
9     *p = 12; //dereferencing
10    printf("a = %d\n", a);
11 }
```

```
C:\Users\HP\Documents\Semester 3\Prak. Desain Dan Analisis Algoritma\...
a = 10
a = 12

-----
Process exited after 9.706 seconds with return value 0
Press any key to continue . . .
```

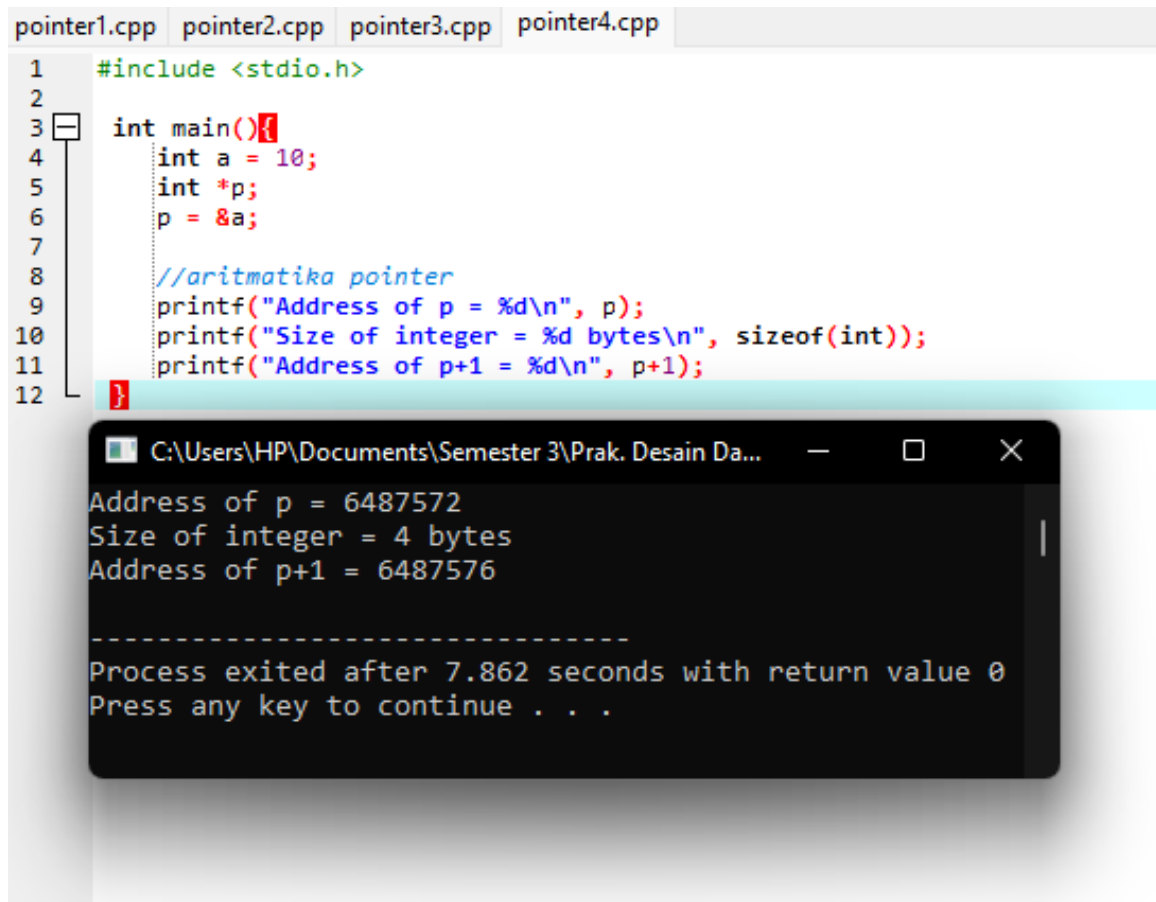
Beberapa contoh program lainnya:



```
pointer1.cpp pointer2.cpp pointer3.cpp
1 #include <stdio.h>
2
3 int main(){
4     int a;
5     int *p;
6     a = 10;
7     p = &a;
8
9     printf("Address of p = %d\n", p);
10    printf("Value at p = %d\n", *p);
11
12    int b = 20;
13    *p = b;
14    printf("Address of p = %d\n", p);
15    printf("Value at p = %d\n", *p);
16 }
```

```
C:\Users\HP\Documents\Semester 3\Prak. Desain Dan Anali...
Address of p = 6487568
Value at p = 10
Address of p = 6487568
Value at p = 20

-----
Process exited after 8.937 seconds with return value 0
Press any key to continue . . .
```

The image shows a C++ IDE with a file named 'pointer1.cpp' selected. The code defines a main function where an integer 'a' is set to 10, a pointer 'p' is assigned the address of 'a' (&a), and three printf statements are used to display the address of 'p', the size of an integer, and the address of 'p+1'. A terminal window in the foreground shows the output: 'Address of p = 6487572', 'Size of integer = 4 bytes', and 'Address of p+1 = 6487576'. It also shows the program exiting after 7.862 seconds with a return value of 0.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int a = 10;
6      int *p;
7      p = &a;
8
9      //aritmatika pointer
10     printf("Address of p = %d\n", p);
11     printf("Size of integer = %d bytes\n", sizeof(int));
12     printf("Address of p+1 = %d\n", p+1);
13 }
```

C:\Users\HP\Documents\Semester 3\Prak. Desain Da... — □ ×

Address of p = 6487572  
Size of integer = 4 bytes  
Address of p+1 = 6487576

-----  
Process exited after 7.862 seconds with return value 0  
Press any key to continue . . .

Nilai dari alamat pointer dapat ditambah, contohnya jika kita tambahkan p dengan 1 alias  $p + 1$ , bisa dilihat bahwa nilai alamat p berubah dari 6487572 berubah menjadi 6487576. Hal itu terjadi karena variable p bertipe integer, dan integer mempunyai ukuran 4 bytes.

## B. Tipe Pointer, Aritmatika Pointer, dan Void Pointer

Variabel pointer bersifat strongly typed yang artinya variabel pointer membutuhkan tipe khusus untuk menyimpan alamat variabel dengan tipe khusus. Jadi  $\text{int}^*$  atau pointer ke integer akan dibutuhkan untuk menyimpan alamat dari variabel integer.  $\text{Char}^*$  dibutuhkan untuk variabel char. Dan begitu pula untuk struct yang didefinisikan oleh user, kita hanya membutuhkan tipe khusus tersebut.

Lalu, mengapa tidak menggunakan tipe generik untuk menyimpan semua macam alamat variabel? Jawabannya adalah kita tidak menggunakan pointer hanya untuk menyimpan alamat memori, tetapi kita juga menggunakannya untuk dereferensi alamat tersebut agar kita bisa mengakses dan mengubah nilai yang ada pada alamat tersebut Seperti yang kita ketahui, tipe data memiliki ukuran yang berbeda-beda, seperti pada compiler modern:

- int: 4 bytes
- char: 1 byte
- float: 4 bytes

Di dalam C/C++ kita dimungkinkan untuk mengoperasikan nilai dari pointer, tapi operator yang dimungkinkan hanyalah penambahan dan pengurangan nilai dari pointer.

Melakukan operasi pada pointer tidak sama seperti kita melakukan operasi pada variabel biasa. Jika operator peningkatan atau pengurangan dilakukan pada nilai dari variabel biasa maka nilai dari variabel tersebut akan ditambah atau dikurangi satu '1' angka/tingkat. Tapi hal itu akan berbeda pada nilai pointer.

Peningkatan atau penurunan pada nilai pointer tergantung pada ukuran memori tipe data yang digunakan. Jika variabel pointer menggunakan integer maka alamat akan bertambah atau berkurang 4 tingkat, jika char maka akan bertambah atau berkurang 1 tingkat. Semua itu tergantung pada tipe data yang digunakan.

Contoh aritmatika pointer dan typecasting pointer:

```
pointer1.cpp pointer2.cpp pointer3.cpp pointer4.cpp pointers.cpp
1  #include <stdio.h>
2  int main() {
3      int a = 1025;
4      int *p;
5      p = &a;
6
7      printf("Size of integer = %d bytes\n", sizeof(int));
8      printf("Address %d, value = %d\n", p,*p);
9      char *p0;
10
11     p0 = (char*)p;
12     printf("Size of integer = %d bytes\n", sizeof(char));
13     printf("Address %d, value = %d\n", p0,*p0);
14 }
15
```

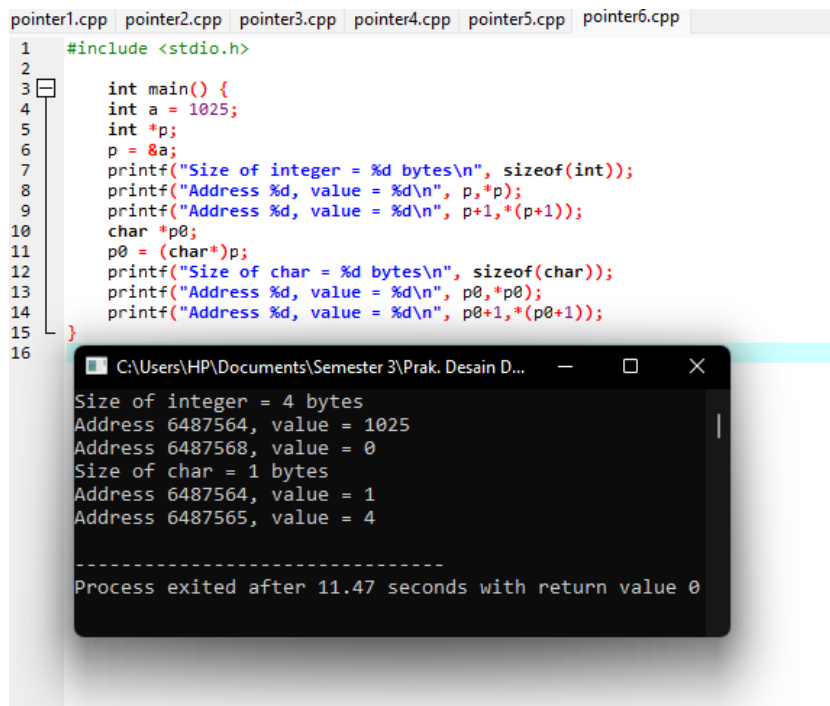
```
C:\Users\HP\Documents\Semester 3\Prak. Desain Dan Analisis Alg...
Size of integer = 4 bytes
Address 6487564, value = 1025
Size of integer = 1 bytes
Address 6487564, value = 1

-----
Process exited after 10.42 seconds with return value 0
Press any key to continue . . .
```

Kemudian muncul pertanyaan kenapa nilai dari p0 adalah 1 bukan 1025, meskipun dia sudah memiliki alamat yang sama dengan p. Untuk itu mari lihat kembali jika dituliskan 1025 ke dalam angka biner menggunakan 32 bits:

1025 = 00000000 00000000 00000100 00000001

Hal tersebut terjadi karena p0 merupakan char dan hanya bisa menyimpan 1 byte. Maka ketika dipanggil nilai dari p0, dia hanya akan menunjukkan nilai dari alamat paling kanan dari 102, yakni 00000001 yang ketika diubah ke nilai sebenarnya adalah 1.



```
pointer1.cpp pointer2.cpp pointer3.cpp pointer4.cpp pointer5.cpp pointer6.cpp
1 #include <stdio.h>
2
3 int main() {
4     int a = 1025;
5     int *p;
6     p = &a;
7     printf("Size of integer = %d bytes\n", sizeof(int));
8     printf("Address %d, value = %d\n", p, *p);
9     printf("Address %d, value = %d\n", p+1, *(p+1));
10    char *p0;
11    p0 = (char*)p;
12    printf("Size of char = %d bytes\n", sizeof(char));
13    printf("Address %d, value = %d\n", p0, *p0);
14    printf("Address %d, value = %d\n", p0+1, *(p0+1));
15 }
16
```

```
C:\Users\HP\Documents\Semester 3\Prak. Desain D...
Size of integer = 4 bytes
Address 6487564, value = 1025
Address 6487568, value = 0
Size of char = 1 bytes
Address 6487564, value = 1
Address 6487565, value = 4

-----
Process exited after 11.47 seconds with return value 0
```

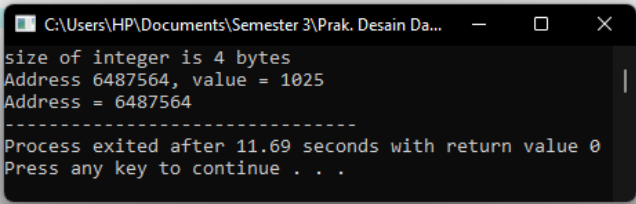
Ketika kita mencetak alamat dari p0+1 akan memunculkan 6487565 yang mana sesuai dengan pertambahan 1 byte dari alamat p0 yaitu 6487564 + 1, namun nilai dari p0+1 berubah menjadi 4. Kenapa hal ini bisa terjadi?

1025 = 00000000 00000000 00000100 00000001

Dari hasil angka biner 1025 yang telah kita punya ketika ditambahkan 1 ke alamat p0 maka sekarang nilai dari p0 + 1 akan memakai 00000100 yang mana jika kita terjemahkan ke desimal akan menghasilkan angka 4.

Terdapat tipe pointer yang merupakan tipe pointer generik yang tidak sesuai dengan tipe data tertentu yang dinamakan void pointer. Tipe data ini dapat menunjuk ke variabel dari tipe data apa pun. Tetapi void pointer memiliki batasan tertentu juga, yang salah satunya adalah nilainya tidak dapat direferensikan secara langsung.

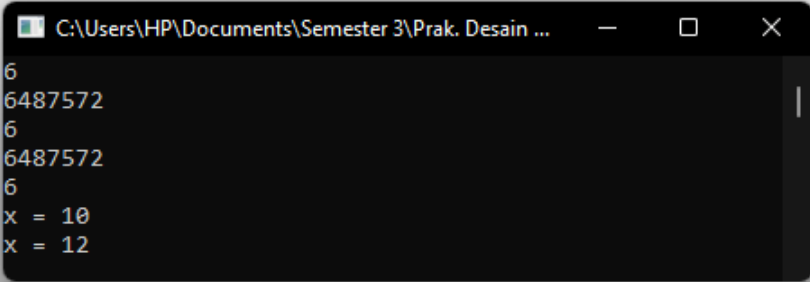
```
pointer1.cpp pointer2.cpp pointer3.cpp pointer4.cpp pointer5.cpp pointer6.cpp pointer7.cpp
1  #include <stdio.h>
2
3  int main() {
4      int a = 1025;
5      int *p;
6      p = &a;
7      printf("size of integer is %d bytes\n", sizeof(int));
8      printf("Address %d, value = %d\n", p,*p);
9
10     // void pointer - generic pointer
11     void *p0;
12     p0 = p;
13     printf("Address = %d",p0 );
14 }
15
```



### C. Pointer ke Pointer

Pointer ke pointer adalah bentuk rantai pointer. Biasanya, pointer berisi alamat variabel. Saat kita mendefinisikan pointer ke pointer, pointer pertama berisi alamat pointer kedua, yang menunjuk ke lokasi yang berisi nilai sebenarnya seperti yang ditunjukkan di bawah ini.

```
pointer1.cpp pointer2.cpp pointer3.cpp pointer4.cpp pointer5.cpp pointer6.cpp pointer7.cpp pointer8.cpp
1  #include <stdio.h>
2
3  int main() {
4      int x = 5;
5      int *p = &x;
6      *p = 6;
7      int **q = &p;
8      int ***r = &q;
9      printf("%d\n",*p);
10     printf("%d\n",*q);
11     printf("%d\n",**q);
12     printf("%d\n",**r);
13     printf("%d\n",***r);
14     ***r = 10;
15     printf("x = %d\n",x);
16     **q = *p + 2;
17     printf("x = %d\n",x);
18 }
```



Kita bisa lihat jika kita mendeklarasikan pointer `**q` yang merujuk ke pointer `*p` maka ketika kita memanggil `*p` akan menghasilkan alamat dari `x` yang sama saja ketika kita memanggil `p`, kemudian

ketika kita memanggil `**p` maka akan menghasilkan nilai dari `x` yaitu 5 sama seperti jika kita memanggil `*p` begitu pula akan sama terhadap pointer `***r` yang merujuk ke `**q`. Kita juga bisa merubah nilai melalui pointer yang merujuk ke pointer lainnya. Pada program kita mendeklarasikan `***r = 10` hal ini akan mengubah nilai `x` menjadi 10 karna pointer `r` merujuk ke pointer `q` merujuk ke pointer `p` dan merujuk ke `x`.

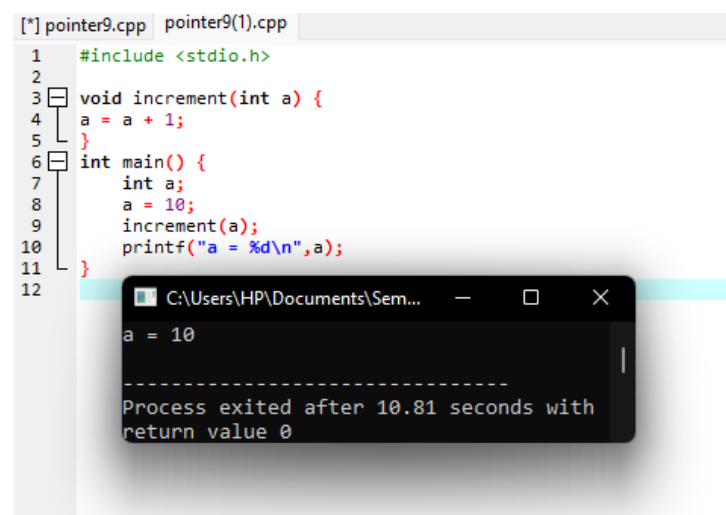
#### D. Pointer Sebagai Argumen Fungsi – Call by Reference

Fungsi dapat dipanggil dengan dua cara: Call by Value atau Call by Reference. Kedua cara ini umumnya dibedakan berdasarkan jenis nilai yang diteruskan kepada fungsi sebagai parameter.

Parameter yang diteruskan ke fungsi disebut parameter aktual sedangkan parameter yang diterima oleh fungsi disebut parameter formal.

Call by Value: Dalam metode pemanggilan fungsi ini, nilai parameter aktual disalin ke parameter formal fungsi dan kedua jenis parameter disimpan di lokasi memori yang berbeda. Jadi setiap perubahan yang dibuat di dalam fungsi tidak tercermin dalam parameter sebenarnya dari pemanggil.

Contoh Call by Value:



```
[*] pointer9.cpp pointer9(1).cpp
1  #include <stdio.h>
2
3  void increment(int a) {
4      a = a + 1;
5  }
6
7  int main() {
8      int a;
9      a = 10;
10     increment(a);
11     printf("a = %d\n",a);
12 }
```

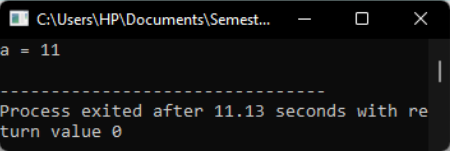
C:\Users\HP\Documents\Sem... a = 10

-----  
Process exited after 10.81 seconds with return value 0

Call by Reference: Baik parameter aktual dan formal merujuk ke lokasi yang sama, sehingga setiap perubahan yang dibuat di dalam fungsi sebenarnya tercermin dalam parameter aktual pemanggil.

### Contoh Call by Reference:

```
[*] pointer9.cpp
1  #include <stdio.h>
2
3  void increment(int *p) {
4      *p = (*p) + 1;
5  }
6
7  int main() {
8      int a;
9      a = 10;
10     increment(&a);
11     printf("a = %d\n",a);
12 }
```



Output berupa a = 11.

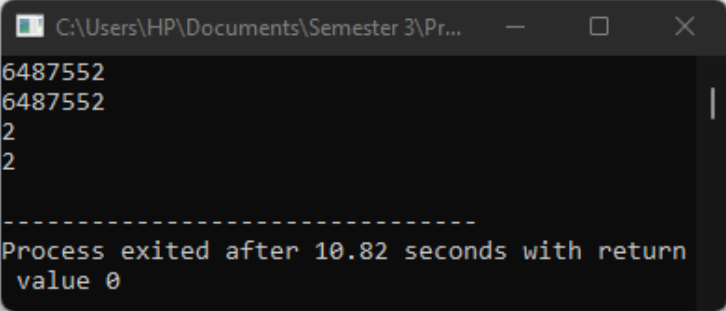
Hal ini terjadi karena ketika kita memanggil fungsi `increment(&a)`, kita mempassing alamat dari variabel `a` ke pointer `*p` dan kemudian operasi `*p = (*p) + 1` akan menambahkan nilai dari alamat variabel yang disimpan dengan 1 sehingga nilai `a` yang semula adalah 10 telah bertambah menjadi 11. Cara ini biasa dinamakan call by reference.

### E. Pointer dan Array

Elemen di index ke-i:

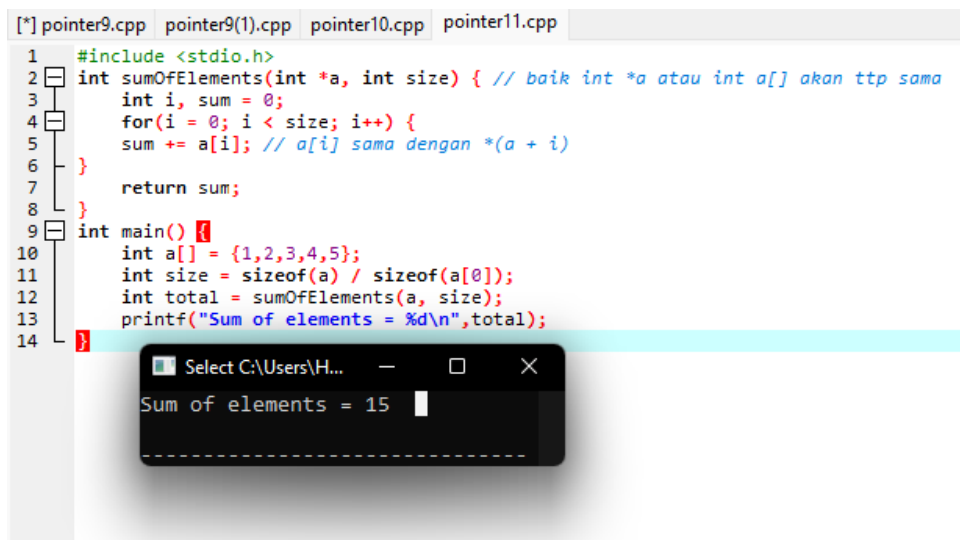
- Alamat : `&A[i]` atau `(A+i)`
- Nilai : `A[i]` atau `*(A+i)`

```
[*] pointer9.cpp  pointer9(1).cpp  pointer10.cpp
1  #include <stdio.h>
2
3  int main() {
4      int a[] = {2,4,5,8,1};
5      printf("%d\n", a); // akan mencetak alamat dari elemen array yg pertama
6      printf("%d\n", &a[0]); // akan mencetak alamat dari elemen array yg pertama
7      printf("%d\n", a[0]); // akan mencetak nilai dari elemen array yg pertama
8      printf("%d\n", *a); // akan mencetak nilai dari elemen array yg pertama
9  }
```





## F. Array Sebagai Argumen Fungsi



```
[*] pointer9.cpp pointer9(1).cpp pointer10.cpp pointer11.cpp
1  #include <stdio.h>
2  int sumOfElements(int *a, int size) { // baik int *a atau int a[] akan ttp sama
3      int i, sum = 0;
4      for(i = 0; i < size; i++) {
5          sum += a[i]; // a[i] sama dengan *(a + i)
6      }
7      return sum;
8  }
9  int main() {
10     int a[] = {1,2,3,4,5};
11     int size = sizeof(a) / sizeof(a[0]);
12     int total = sumOfElements(a, size);
13     printf("Sum of elements = %d\n", total);
14 }
```

Sum of elements = 15

Maka pada output program akan menghasilkan keluaran yakni 15 yang merupakan hasil penjumlahan dari semua elemen array. Perlu diingat bahwa karena array ini di passing by reference maka kita perlu membuat 1 variabel yg menyimpan ukuran dari array kita, jika kita tidak melakukan hal tersebut maka yang akan di passing ke fungsi penjumlahan array hanyalah elemen array yang pertama.

## G. Array Karakter dan Pointer

Beberapa karakter array dan pointer adalah sbb:

- Jika kita ingin menyimpan sebuah string dalam sebuah array maka kita harus memenuhi syarat ukuran array  $\geq$  jumlah karakter pada string + 1.
- Array dan pointer berbeda tipe namun digunakan dalam cara yang serupa.
- Array selalu di-passing ke sebuah function by reference. Artinya adalah selalu nilai alamat array lah yang dipassing dan bukan value dari array itu sendiri.

Contoh: Membuat program cetak “Hello”.



```
pointer16.cpp pointer17.cpp pointer18.cpp pointer19.cpp
1  #include <stdio.h>
2  void A() {
3      printf("Hello\n");
4  }
5  void B(void (*ptr)()) { // pointer fungsi sebagai argumen
6      ptr(); // fungsi callback yang ptr tunjuk
7  }
8  int main() {
9      B(A);
10 }
11
```

Hello

## H. Pointer dan Array Multi-Dimensional

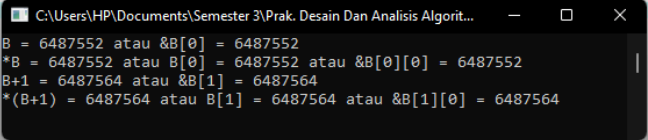
Misalnya buffer bertipe 5 array dua dimensi. Jenis ekspresi `*buffer` adalah "array of arrays (yaitu array dua dimensi)".

Berdasarkan konsep di atas menerjemahkan ekspresi `*( *( *(buffer + 2) + 1) + 2)` langkah-demi langkah membuatnya lebih jelas.

- `buffer` – Sebuah array dari 5 array dua dimensi, yaitu jenisnya adalah “array dari 5 array dua dimensi”.
- `buffer + 2` – perpindahan untuk elemen ke-3 dalam 5 array dua dimensi.
- `*(buffer + 2)` – dereferencing, yaitu tipenya sekarang adalah array dua dimensi.
- `*(buffer + 2) + 1` – perpindahan untuk mengakses elemen ke-2 dalam array 7 array satu dimensi.
- `*( *(buffer + 2) + 1)` – dereferencing (mengakses), sekarang jenis ekspresi “`*( *(buffer + 2) + 1)`” adalah array bilangan bulat.
- `*( *(buffer + 2) + 1) + 2` – perpindahan untuk mendapatkan elemen pada posisi ke-3 dalam array bilangan bulat berdimensi tunggal.
- `*( *( *(buffer + 2) + 1) + 2)` – mengakses elemen pada posisi ke-3 (tipe ekspresi keseluruhan adalah `int` sekarang).

Untuk Array 2-D:

```
[*] pointer9.cpp pointer9(1).cpp pointer10.cpp pointer11.cpp [*] pointer12.cpp pointer13.cpp
1  #include <stdio.h>
2
3  void print(char *c)
4  {
5      while(*c != '\0');
6      {
7          printf("%c", *c);
8          c++;
9      }
10     printf("\n");
11 }
12 int main(){
13     int B[2][3] = {{2, 3, 4}, {4, 5, 6}};
14     int (*p)[3] = B; //akan mengembalikan pointer ke array 1-D 3 integer
15     printf("B = %d atau &B[0] = %d\n", B, &B[0]);
16     printf("*B = %d atau B[0] = %d atau &B[0][0] = %d\n", *B, B[0], &B[0][0]);
17     printf("B+1 = %d atau &B[1] = %d\n", B+1, &B[1]); //alamat B + ukuran array 1-D 3 integer
18     printf("*(B+1) = %d atau B[1] = %d atau &B[1][0] = %d\n", *(B+1), B[1], &B[1][0]);
19 }
20
```



```
B = 6487552 atau &B[0] = 6487552
*B = 6487552 atau B[0] = 6487552 atau &B[0][0] = 6487552
B+1 = 6487564 atau &B[1] = 6487564
*(B+1) = 6487564 atau B[1] = 6487564 atau &B[1][0] = 6487564
```

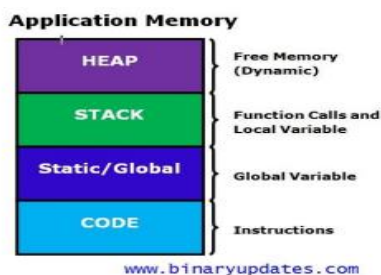
Untuk Array Multidimensional:

```
[*] pointer9.cpp pointer9(1).cpp pointer10.cpp pointer11.cpp [*] pointer12.cpp pointer13.cpp pointer14.cp
1  #include <stdio.h>
2
3  int main(){
4      int C[3][2][2] = {
5          {{2, 5}, {7, 9}},
6          {{3, 4}, {6, 1}},
7          {{0, 8}, {11, 13}}
8      };
9      printf("%d %d %d %d\n", C, *C, C[0], &C[0]);
10     printf("%d\n", C[0][0]+1);
11 }
12
```

## I. Pointer dan Memori Dinamis

Representasi memori dari program C terdiri dari bagian berikut:

- Code
- Static/Global Variable
- Stack
- Heap



**Code:** Segmen kode adalah salah satu bagian dari program dalam file objek atau dalam memori, yang berisi instruksi yang dapat dieksekusi. Biasanya, segmen teks dapat dibagikan sehingga hanya satu salinan yang perlu ada di memori untuk program yang sering dijalankan, seperti editor teks, kompiler C, shell, dan sebagainya. Juga, segmen teks sering hanya-baca, untuk mencegah program memodifikasi instruksinya secara tidak sengaja.

**Static/Global Variable:** Segmen ini adalah bagian dari ruang alamat virtual suatu program, yang berisi variabel global dan variabel statis yang diinisialisasi oleh programmer. Perhatikan bahwa, segmen ini tidak hanya-baca, karena nilai variabel dapat diubah pada waktu proses. Segmen ini dapat diklasifikasikan lebih lanjut ke dalam area baca-saja yang diinisialisasi dan area baca-tulis yang diinisialisasi.

**Stack:** Segmen stack biasanya berdampingan dengan segmen heap dan tumbuh ke arah yang berlawanan; ketika pointer stack bertemu dengan pointer heap, memori kosong telah habis. (Dengan ruang alamat modern yang besar dan teknik memori virtual, mereka dapat ditempatkan hampir di mana saja, tetapi biasanya masih tumbuh ke arah yang berlawanan.).

**Heap:** Heap adalah segmen di mana alokasi memori dinamis biasanya terjadi. Area Heap dikelola oleh malloc, realloc, dan free, yang dapat menggunakan panggilan sistem brk dan sbrk untuk menyesuaikan ukurannya (perhatikan bahwa penggunaan brk/sbrk dan satu "heap area" tidak diperlukan untuk memenuhi kontrak malloc/realloc/free; mereka juga dapat diimplementasikan menggunakan mmap untuk mencadangkan daerah memori virtual yang berpotensi tidak bersebelahan ke dalam ruang alamat virtual proses). Area Heap dibagikan oleh semua library bersama dan modul yang dimuat secara dinamis dalam suatu proses.

Alokasi memori dinamis di C:

```
[*] pointer9.cpp pointer9(1).cpp pointer10.cpp pointer11.cpp [*] pointer12.cpp pointer13.cpp pointer14.cpp pointer15.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int a; //disimpan di stack
7      int *p;
8      p = (int*)malloc(sizeof(int)); //alokasi memori di heap
9      *p = 10;
10     free(p); //dialokasi
```

Alokasi memori dinamis di C++:

```
pointer16.cpp
1  #include <iostream>
2
3  int main(){
4      int a;
5      int *p;
6      p = new int;
7      *p = 10;
8      delete p;
9  }
```

## J. Alokasi Memori Dinamis di C: malloc, calloc, realloc, free

Alokasi blok memori:

- **malloc**

Metode "malloc" atau "alokasi memori" di C digunakan untuk secara dinamis mengalokasikan satu blok memori besar dengan ukuran yang ditentukan. Ini mengembalikan pointer tipe void yang dapat dilemparkan ke pointer dalam bentuk apa pun.

Syntax: malloc(size\_t size);

- **calloc**

Metode "calloc" atau "alokasi contiguous" dalam C digunakan untuk secara dinamis mengalokasikan jumlah blok memori yang ditentukan dari tipe yang ditentukan. ini sangat mirip dengan malloc() tetapi memiliki dua titik yang berbeda dan ini adalah:

1. Menginisialisasi setiap blok dengan nilai default '0'.
2. Memiliki dua parameter atau argumen dibandingkan dengan malloc().

Syntax: calloc(size\_t num, size\_t size);

- **realloc**

Metode "realloc" atau "re-allocation" dalam C digunakan untuk secara dinamis mengubah alokasi memori dari memori yang dialokasikan sebelumnya. Dengan kata lain, jika memori yang sebelumnya dialokasikan dengan bantuan malloc atau calloc tidak mencukupi, realloc dapat digunakan untuk mengalokasikan ulang memori secara dinamis. Alokasi ulang memori mempertahankan nilai yang sudah ada dan blok baru akan diinisialisasi dengan nilai sampah default.

Syntax: realloc(void \*ptr, size\_t size);

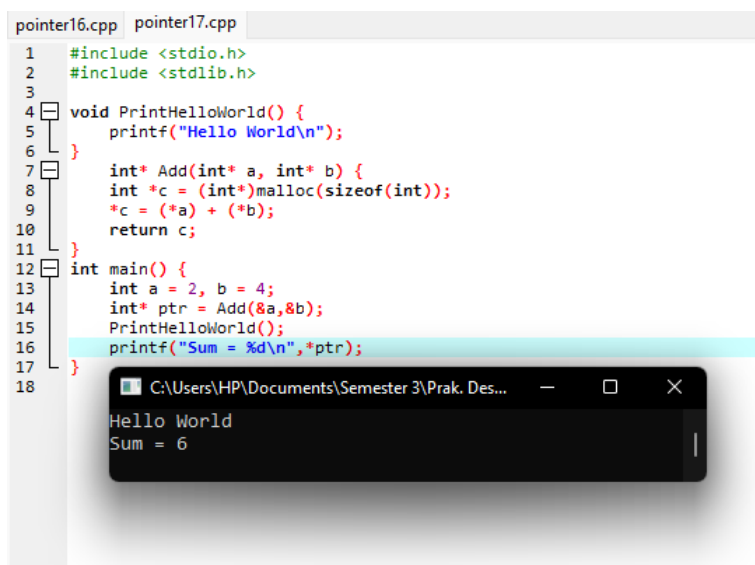
Dealokasi blok memori:

- **free**

Metode "free" dalam C digunakan untuk mengalokasikan memori secara dinamis. Memori yang dialokasikan menggunakan fungsi malloc() dan calloc() tidak dialokasikan sendiri. Oleh karena itu metode free() digunakan, setiap kali alokasi memori dinamis terjadi. Ini membantu mengurangi pemborosan memori dengan membebaskannya.

Syntax: free(ptr);

## K. Pointer Sebagai Nilai Kembali Fungsi



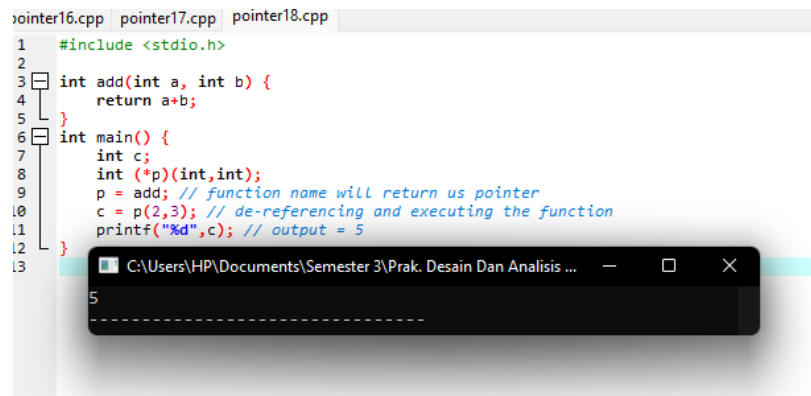
```
pointer16.cpp pointer17.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void PrintHelloWorld() {
5      printf("Hello World\n");
6  }
7  int* Add(int* a, int* b) {
8      int *c = (int*)malloc(sizeof(int));
9      *c = (*a) + (*b);
10     return c;
11 }
12 int main() {
13     int a = 2, b = 4;
14     int* ptr = Add(&a, &b);
15     PrintHelloWorld();
16     printf("Sum = %d\n", *ptr);
17 }
18
```

C:\Users\HP\Documents\Semester 3\Prak. Des...  
Hello World  
Sum = 6

Pada kode di atas, nilai kembali fungsi Add harus dialokasikan di heap agar dapat mengembalikan nilainya ke main dengan benar. Jika tidak, maka nilai yang dikembalikan akan menjadi garbage value karena memori fungsi pada stack sudah didealokasikan ketika kembali ke main.

## L. Pointer Fungsi

Di C, seperti pointer data normal (int \*, char \*, dll), kita dapat memiliki pointer ke fungsi. Berikut adalah contoh sederhana yang menunjukkan deklarasi dan pemanggilan fungsi menggunakan pointer fungsi.



```
pointer16.cpp pointer17.cpp pointer18.cpp
1 #include <stdio.h>
2
3 int add(int a, int b) {
4     return a+b;
5 }
6
7 int main() {
8     int c;
9     int (*p)(int,int);
10    p = add; // function name will return us pointer
11    c = p(2,3); // de-referencing and executing the function
12    printf("%d",c); // output = 5
13 }
```

## M. Pointer Fungsi dan Callback

Callback adalah kode yang dapat dieksekusi yang diteruskan sebagai argumen ke kode lain, yang diharapkan untuk memanggil kembali (mengeksekusi) argumen pada waktu tertentu. Dalam bahasa sederhana, jika referensi suatu fungsi dilewatkan ke fungsi lain sebagai argumen untuk memanggilnya, maka itu akan disebut sebagai fungsi callback. Dalam C, fungsi callback adalah fungsi yang dipanggil melalui pointer fungsi.

Di bawah ini adalah contoh sederhana dalam C untuk menggambarkan definisi di atas agar lebih jelas:



```
pointer16.cpp pointer17.cpp pointer18.cpp pointer19.cpp
1 #include <stdio.h>
2 void A() {
3     printf("Hello\n");
4 }
5 void B(void (*ptr)()) { // pointer fungsi sebagai argumen
6     ptr(); // fungsi callback yang ptr tunjuk
7 }
8 int main() {
9     B(A);
10 }
11
```

## N. Kebocoran Memori di C

Kebocoran memori terjadi ketika programmer membuat memori di heap dan lupa untuk menghapusnya.

Konsekuensi dari kebocoran memori adalah mengurangi kinerja komputer dengan mengurangi jumlah memori yang tersedia. Akhirnya, dalam kasus terburuk, terlalu banyak memori yang tersedia dapat dialokasikan dan semua atau sebagian dari sistem atau perangkat berhenti bekerja dengan benar, aplikasi gagal, atau sistem menjadi sangat lambat.

Berikut adalah kode yang dapat menimbulkan kebocoran memori:

```
pointer16.cpp pointer17.cpp pointer18.cpp pointer19.cpp [*] pointer20.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  int cash = 100;
5  void Play(int bet) {
6      char *C = (char*)malloc(3*sizeof(char));
7      C[0] = 'J'; C[1] = 'Q'; C[2] = 'K';
8      printf("Shuffling...\n");
9      srand(time(NULL)); // seeding generator angka acak
10     int i;
11     for (i = 0; i < 5; i++) {
12         int x = rand() % 3;
13         int y = rand() % 3;
14         int temp = C[x];
15         C[x] = C[y];
16         C[y] = temp; // swap karakter di posisi x dan y
17     }
18
19     int playersGuess;
20     printf("Dimana posisi queen - 1, 2, atau 3?: ");
21     scanf("%d", &playersGuess);
22     if (C[playersGuess - 1] == 'Q') {
23         cash += 3*bet;
24         printf("Kamu menang! Hasil = \"%c %c %c\" Total Cash = $d\n", C[0], C[1], C[2],
25             cash);
26     }
27     else {
28         cash -= bet;
29         printf("Kamu kalah! Hasil = \"%c %c %c\" Total Cash = $d\n", C[0], C[1], C[2], cash);
30     }
31 }
32 int main() {
33     int bet;
34     printf("***Selamat Datang di Casino Virtual**\n\n");
35     printf("Total cash = $d\n", cash);
36     while (cash > 0) {
37         printf("Berapa taruhan Anda? $");
38         scanf("%d", &bet);
39         if (bet == 0 || bet > cash) break;
40         Play(bet);
41         printf("\n*****\n");
42     }
}
```

Ketika program dijalankan, dapat dilihat dari process viewer bahwa memori program semakin bertambah seiring program dijalankan. Hal ini dikarenakan tidak dilakukan dealokasi dengan metode free ketika memori yang dialokasikan dengan metode malloc selesai digunakan.

Kebocoran memori dapat diatasi dengan dealokasi variabel C di akhir fungsi Play:

```
pointer16.cpp pointer17.cpp pointer18.cpp pointer19.cpp [*] pointer20.cpp [*] pointer21.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  int cash = 100;
5  void Play(int bet) {
6      char *C = (char*)malloc(3*sizeof(char));
7      C[0] = 'J'; C[1] = 'Q'; C[2] = 'K';
8      printf("Shuffling...\n");
9      srand(time(NULL)); // seeding generator angka acak
10     int i;
11     for (i = 0; i < 5; i++) {
12         int x = rand() % 3;
13         int y = rand() % 3;
14         int temp = C[x];
15         C[x] = C[y];
16         C[y] = temp; // swap karakter di posisi x dan y
17     }
18     int playersGuess;
19     printf("Dimana posisi queen - 1, 2, atau 3?: ");
20     scanf("%d", &playersGuess);
21     if (C[playersGuess - 1] == 'Q') {
22         cash += 3*bet;
23         printf("Kamu menang! Hasil = \"%c %c %c\" Total Cash = $%d\n", C[0], C[1], C[2],
24         cash);
25     }
26     else {
27         cash -= bet;
28         printf("Kamu kalah! Hasil = \"%c %c %c\" Total Cash = $%d\n", C[0], C[1], C[2], cash);
29     }
30     free(C);
31 }
32 int main() {
33     int bet;
34     printf("***Selamat Datang di Casino Virtual**\n\n");
35     printf("Total cash = $%d\n", cash);
36     while (cash > 0) {
37         printf("Berapa taruhan Anda? $");
38         scanf("%d", &bet);
39         if (bet == 0 || bet > cash) break;
40         Play(bet);
41         printf("\n*****\n");
42     }
43 }
```