Memory Management 1 : Pointer

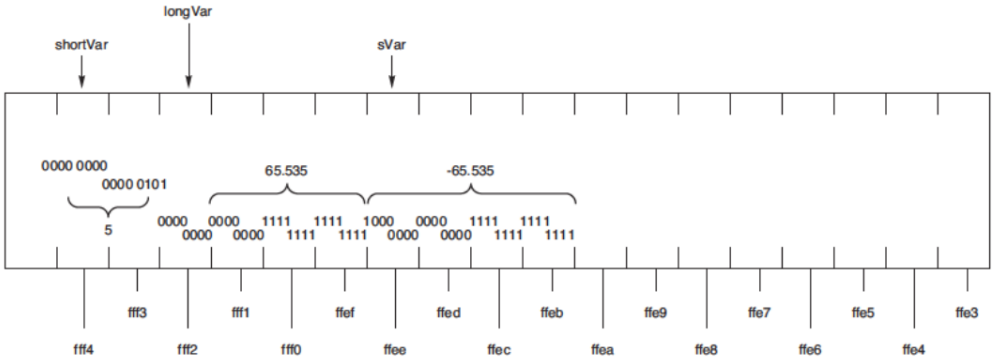
Pertemuan 6

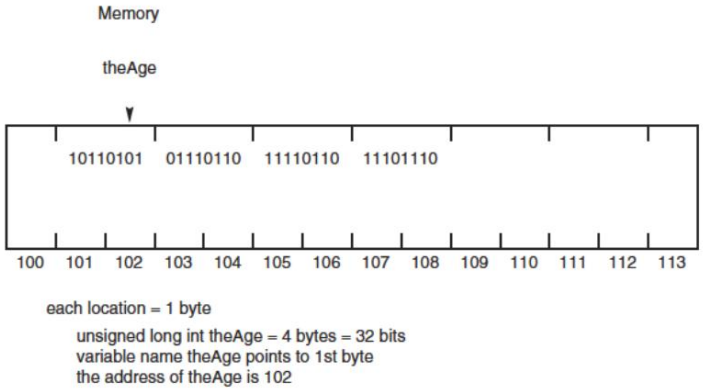
Outline

- Memory Management 1 : Pointer
  - Understanding Pointer
  - Manipulating Data using Pointer
  - Examining Addresses Stored in Pointers
  - The Heap
  - Creating Object on the Heap
  - Accessing Data Members Using Pointers
  - Member Data on the Heap
  - The **this** Pointer
  - **const** Pointers and **const** Member Functions

Definisi Pointer

- Variabel adalah obyek yang dapat menahan nilai. Variabel integer memegang bilangan. Variabel character memegang huruf. Sebuah pointer adalah variabel yang menyimpan alamat memori. Pointer adalah kekuatan yang dimiliki oleh bahasa pemrograman C++. Pointer menyediakan kemampuan untuk memanipulasi memori komputer secara langsung.





Menyimpan Alamat di Pointer

- Pointer pAge dideklarasikan untuk menahan sebuah alamat. Hal ini menyatakan halaman menjadi pointer ke int. Artinya, halaman dinyatakan untuk menahan alamat int. Jika Anda menginisialisasi pointer ke 0 atau NULL, Anda harus secara khusus menetapkan alamat howOld ke pAge.

```
int *pAge = NULL;

int howOld = 50;     // make a variable
int *pAge = 0;       // make a pointer
pAge = &howOld;      // put address in pAge
```

Area Program Memory

- Mengapa repot-repot menggunakan pointer ketika Anda dapat menggunakan variabel untuk mengakses ke nilai itu? Pointer digunakan paling sering untuk 3 pekerjaan:
  - Mengelola data pada heap
  - Mengakses data dan fungsi anggota class
  - Melewati variabel dengan mengacu pada fungsi
- Programmer menangani 5 hal dengan memori:
  - *Global name space* : menyimpan variabel global
  - *Heap* : tempat penyimpanan yg bebas
  - *Registers* : menangani fungsi internal
  - *Code space* : menyimpan kode program
  - *Stack* : menyimpan variabel lokal

Stack vs Heap : **new**

- *Stack* dibersihkan secara otomatis ketika fungsi mengembalikan nilai. Semua variabel lokal keluar dari ruang lingkup, dan dikeluarkan dari stack.
- *Heap* tidak dibersihkan sampai program berakhir, dan itu adalah tanggung jawab Anda untuk membebaskan setiap memori yang telah disediakan ketika Anda selesai dengan itu.
- Anda mengalokasikan memori pada *stack* di C ++ dengan menggunakan kata kunci “new” dan diikuti oleh jenis objek yang Anda ingin mengalokasikan sehingga compiler tahu berapa banyak memori yang diperlukan.

- Nilai kembalian dari “new” adalah alamat memori. Ini harus di-assign ke pointer. Untuk membuat pendek unsigned di heap, Anda mungkin menulis sebagai berikut:

```
unsigned short int *pPointer;
```

- pPointer sekarang menunjuk ke unsigned short int di heap. Anda dapat menggunakan ini seperti pointer lainnya untuk variabel dan menetapkan nilai ke daerah itu dari memori

```
pPointer = new unsigned short int;
*pPointer = 72;
```

- Program ini berarti "menempatkan nilai 72 ke pPointer" atau "menetapkan nilai 72 ke daerah di heap yang ditunjuk oleh pPointer."

Stack vs Heap : **delete**

- Bila Anda telah selesai dengan area memori, Anda harus menghapus panggilan dari pointer untuk mengembalikan memori ke heap. Ingat bahwa pointer itu sendiri adalah variabel lokal. Untuk mengembalikan memori ke tumpukan, Anda menggunakan kata kunci *delete*.
- Ketika menghapus panggilan pointer, memori dibebaskan. Memanggil menghapus pointer itu lagi akan menyebabkan program crash. Memanggil menghapus pada pointer akan aman, ketika Anda menghapus pointer, dan set ke NULL.

```
Animal *pDog = new Animal;
delete pDog; // frees the memory
pDog = NULL; // sets pointer to null
// ...
delete pDog; // harmless
```

Menghindari *Memory Leaks*

- Cara lain yang mungkin secara tidak sengaja membuat terjadinya kebocoran memori (*memory leaks*) adalah dengan pemindahan pointer sebelum menghapus memori yang menunjuk. Perhatikan kode berikut ini:

```
1: unsigned short int *pPointer = new unsigned short int;
2: *pPointer = 72;
3: pPointer = new unsigned short int;
4: *pPointer = 84;
```

  - Baris 1 menciptakan pPointer dan memberikan alamat di heap.
  - Baris 2 menyimpan nilai 72 di area tersebut dari memori.
  - Baris 3 reassign pPointer ke area lainnya dari memori.
  - Baris 4 menyimpan nilai 84 ke area tersebut itu.

- Area di memory dengan nilai 72 kini tidak tersedia karena pointer ke area itu telah dipindahkan. Tidak ada cara untuk mengakses daerah asli dari memori, juga tidak ada cara lainnya untuk membebaskan sebelum program berakhir. Untuk mengasntispasi hal tersebut, kode harus ditulis seperti ini:

```
1: unsigned short int *pPointer = new unsigned short int;
2: *pPointer = 72;
3: delete pPointer;
4: pPointer = new unsigned short int;
5: *pPointer = 84;
```

- Sekarang memori yang awalnya ditunjuk oleh pPointer sudah dihapus.

1. Understanding Pointer

Addresser.cpp

```
1: #include <iostream>
2: int main()
3: {
4:
5:        unsigned short shortVar = 5;
6:        unsigned long longVar = 65535;
7:        long sVar = -65535;
8:
9:        std::cout « "shortVar:\t" « shortVar;
10:       std::cout « "\tAddress of shortVar:\t" « &shortVar « "\n";
11:       std::cout « "longVar:\t" « longVar;
12:       std::cout « "\tAddress of longVar:\t" « &longVar « "\n";
13:       std::cout « "sVar:\t\t" « sVar;
14:       std::cout « "\tAddress of sVar:\t" « &sVar « "\n";
15:
16:       return 0;
17: }
```

2. Manipulating Data using Pointer

Pointer.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:         int myAge; // a variable
6:         int *pAge = NULL; // a pointer
7:
8:         myAge = 5;
9:         pAge = &myAge; // assign address of myAge to pAge
10:        std::cout << "myAge: " << myAge << "\n";
11:        std::cout << "*pAge: " << *pAge << "\n\n";
12:
13:        std::cout << "*pAge = 7\n";
14:        *pAge = 7; // sets myAge to 7
15:        std::cout << "*pAge: " << *pAge << "\n";
16:        std::cout << "myAge: " << myAge << "\n\n";
17:
18:        std::cout << "myAge = 9\n";
19:        myAge = 9;
20:        std::cout << "myAge: " << myAge << "\n";
21:        std::cout << "*pAge: " << *pAge << "\n";
22:
23:        return 0;
24: }
```

3. Examining Addresses Stored in Pointers

PointerCheck.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:        unsigned short int myAge = 5, yourAge = 10;
6:        unsigned short int *pAge = &myAge; // a pointer
7:
8:        std::cout << "myAge:\t" << myAge;
9:        std::cout << "\t\tyourAge:\t" << yourAge << "\n";
10:       std::cout << "&myAge:\t" << &myAge;
11:       std::cout << "\t&yourAge:\t" << &yourAge <<"\n";
12:
13:       std::cout << "pAge:\t" << pAge << "\n";
14:       std::cout << "*pAge:\t" << *pAge << "\n\n";
15:
16:       pAge = &yourAge; // reassign the pointer
17:
18:       std::cout << "myAge:\t" << myAge;
19:       std::cout << "\t\tyourAge:\t" << yourAge << "\n";
20:       std::cout << "&myAge:\t" << &myAge;
21:       std::cout << "\t&yourAge:\t" << &yourAge <<"\n";
22:
23:       std::cout << "pAge:\t" << pAge << "\n";
24:       std::cout << "*pAge:\t" << *pAge << "\n\n";
25:
26:       std::cout << "&pAge:\t" << &pAge << "\n";
27:       return 0;
28: }
```

4. The Heap

…

```
1: #include <iostream>
2:
3: int main()
4: {
5:        int localVariable = 5;
6:        int *pLocal= &localVariable;
7:        int *pHeap = new int;
8:        if (pHeap == NULL)
9:        {
10:                std::cout << "Error! No memory for pHeap!!";
11:                return 1;
12:       }
13:       *pHeap = 7;
14:       std::cout << "localVariable: " << localVariable << "\n";
15:       std::cout << "*pLocal: " << *pLocal << "\n";
16:       std::cout << "*pHeap: " << *pHeap << "\n";
17:       delete pHeap;
18:       pHeap = new int;
19:       if (pHeap == NULL)
20:       {
21:                std::cout << "Error! No memory for pHeap!!";
22:                return 1;
23:       }
24:       *pHeap = 9;
25:       std::cout << "*pHeap: " << *pHeap << "\n";
26:       delete pHeap;
27:       return 0;
28: }
```

5. Creating Object on the Heap

HeapCreator.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:          public:
6:                      SimpleCat();
7:                      ~SimpleCat();
8:                      private:
9:                      int itsAge;
10: };
11:
12: SimpleCat::SimpleCat()
13: {
14:         std::cout << "Constructor called\n";
15:         itsAge = 1;
16: }
17:
18: SimpleCat::~SimpleCat()
19: {
20:         std::cout << "Destructor called\n";
21: }
22:
23: int main()
24: {
25:         std::cout << "SimpleCat Frisky ...\n";
26:         SimpleCat Frisky;
27:
28:         std::cout << "SimpleCat *pRags = new SimpleCat ...\n";
29:         SimpleCat *pRags = new SimpleCat;
30:
31:         std::cout << "delete pRag s ...\n";
32:         delete pRags;
33:
34:         std::cout << "Exiting, watch Frisky go ...\n";
35:         return 0;
36: }
```

6. Accessing Data Members Using Pointers

HeapAccessor.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:        public:
6:                  SimpleCat() { itsAge = 2; }
7:                  ~SimpleCat() {}
8:                  int GetAge() const { return itsAge; }
9:                  void SetAge(int age) { itsAge = age; }
10:       private:
11:                 int itsAge;
12: };
13:
14: int main()
15: {
16:       SimpleCat *Frisky = new SimpleCat;
17:       std::cout << "Frisky is " << Frisky->GetAge()
18:       << " years old" << "\n";
19:
20:       Frisky->SetAge(5);
21:       std::cout << "Frisky is " << Frisky->GetAge()
22:       << " years old\n";
23:
24:       delete Frisky;
25:       return 0;
26: }
```

7. Member Data on the Heap

…

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:         public:
6:                   SimpleCat();
7:                   ~SimpleCat();
8:                   int GetAge() const { return *itsAge; }
9:                   void SetAge(int age) { *itsAge = age; }
10:
11:                  int GetWeight() const { return *itsWeight; }
12:                  void setWeight (int weight) { *itsWeight = weight; }
13:
14:        private:
15:                  int *itsAge;
16:                  int *itsWeight;
17: };
18:
```

...

```
19: SimpleCat::SimpleCat()
20: {
21:       itsAge = new int(2);
22:       itsWeight = new int(5);
23: }
24:
25: SimpleCat::~SimpleCat()
26: {
27:       delete itsAge;
28:       delete itsWeight;
29: }
30:
31: int main()
32: {
33:       SimpleCat *Frisky = new SimpleCat;
34:       std::cout << "Frisky is " << Frisky->GetAge()
35:       << " years old\n";
36:
37:       Frisky->SetAge(5);
38:       std::cout << "Frisky is " << Frisky->GetAge()
39:       << " years old\n";
40:
41:       delete Frisky;
42:       return 0;
43: }
```

8. The **this** Pointer

This.cpp

```
1: #include <iostream>
2:
3: class Rectangle
4: {
5:        public:
6:                  Rectangle();
7:                  ~Rectangle();
8:                  void SetLength(int length) { this->itsLength = length; }
9:                  int GetLength() const { return this->itsLength; }
10:                 void SetWidth(int width) { itsWidth = width; }
11:                 int GetWidth() const { return itsWidth; }
12:
13:       private:
14:       int itsLength;
15:       int itsWidth;
16: };
17:
18: Rectangle::Rectangle()
19: {
20:       itsWidth = 5;
21:       itsLength = 10;
22: }
23:
24: Rectangle::~Rectangle()
25: {}
26:
```

...

```
27: int main()
28: {
29:       Rectangle theRect;
30:       std::cout << "theRect is " << theRect.GetLength()
31:       << " feet long.\n";
32:       std::cout << "theRect is " << theRect.GetWidth()
33:       << " feet wide.\n";
34:
35:       theRect.SetLength(20);
36:       theRect.SetWidth(10);
37:       std::cout << "theRect is " << theRect.GetLength()
38:       << " feet long.\n";
39:       std::cout << "theRect is " << theRect.GetWidth()
40:       << " feet wide.\n";
41:
42:       return 0;
43: }
```

9. **const** Pointers and **const** Member Functions

ConstPointer.cpp

```
1: #include <iostream>
2:
3: class Rectangle
4: {
5:public:
6:        Rectangle();
7:        ~Rectangle();
8:        void SetLength(int length) { itsLength = length; }
9:        int GetLength() const { return itsLength; }
10:
11:       void SetWidth(int width) { itsWidth = width; }
12:       int GetWidth() const { return itsWidth; }
13:
14: private:
15:       int itsLength;
16:       int itsWidth;
17: };
18:
19: Rectangle::Rectangle():
20: itsWidth(5),
21: itsLength(10)
22: {}
23:
24: Rectangle::~Rectangle()
25: {}
26:
```

…

```
27: int main()
28: {
29:       Rectangle* pRect = new Rectangle;
30:       const Rectangle *pConstRect = new Rectangle;
31:       Rectangle* const pConstPtr = new Rectangle;
32:
33:       std::cout << “pRect width: “
34:       << pRect->GetWidth() << “ feet\n”;
35:       std::cout << “pConstRect width: “
36:       << pConstRect->GetWidth() << “ feet\n”;
37:       std::cout << “pConstPtr width: “
38:       << pConstPtr->GetWidth() << “ feet\n”;
39:
40:       pRect->SetWidth(10);
41:       // pConstRect->SetWidth(10);
42:       pConstPtr->SetWidth(10);
43:
44:       std::cout << “pRect width: “
45:       << pRect->GetWidth() << “ feet\n”;
46:       std::cout << “pConstRect width: “
47:       << pConstRect->GetWidth() << “ feet\n”;
48:       std::cout << “pConstPtr width: “
49:       << pConstPtr->GetWidth() << “ feet\n”;
50:       return 0;
51: }
```

Tugas

- Modifikasi program PointerCheck.cpp untuk mengalikan "yourAge" dan *pAge, kemudian simpan hasilnya di variabel baru. Tampilkan variabel baru tersebut. Bagaimana compiler dapat menjelaskan kepada Anda perbedaan penggunaan operator * untuk perkalian dan operator * untuk pointer pAge?
- Modifikasi program PointerCheck.cpp untuk menggunakan pointer *pAge untuk merubah isi dari myAge atau yourAge!
- Tambahkan seekor kucing bernama Spooky ke program HeapCreator!
- Modifikasi program HeapAccessor agar program tersebut tidak menggunakan cara "point-to operator".