

# Podstawy Informatyki

**Katedra Telekomunikacji, EiT**

dr inż. Jarosław Bułat (c)

[kwant@agh.edu.pl](mailto:kwant@agh.edu.pl)

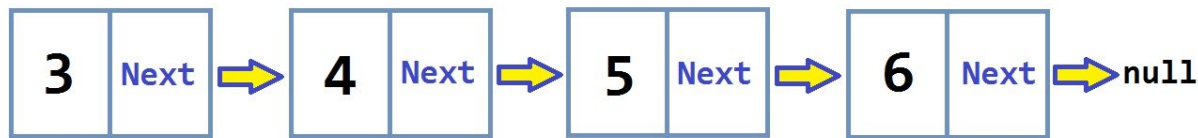
# Plan prezentacji

- » Linked list - przykład
- » Debugger
- » Profiler
- » Refactoring

# Linked list

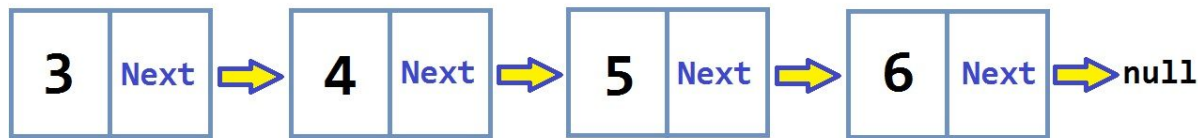
jeszcze raz

# lista jednokierunkowa



- » Każdy bloczek to dowolnie złożona struktura
- » Każda struktura zawiera **wskaźnik na następną**
- » “Łańcuch” takich struktur **można w trakcie działania programu modyfikować**:
  - dostawić nowy element na końcu
  - wyrzucić [5] a [4]->next przypisać do [6]
  - zamienić [3] z [6] bez kopiowania, tylko przepisując odpowiednie [\*]->next
  - zmienić wartość każdego elementu
- » Dostęp jest sekwencyjny

# lista jednokierunkowa



» **Zalety:**

- duża elastyczność w manipulacji danymi
- zmiany nie wymagają kopiowania
- nie wymagana ciągła przestrzeń adresowa
- możliwe grafy, drzewa, dwukierunkowy listy

» **Wady:**

- większa zajętość pamięci (wymagany “next”)
- przeglądanie sekwencyjne (nie da się list[30])\*
- tworzenie listy wymaga **new** ... czyli syscall\*

# Przykład: dict

- » Utworzyć typ “słownik” (ang. *dictionary*)
  - słownik to para: (klucz, wartość)
  - (“red”, 4) (klucz string, wartość Int)
  - dict.add(“red”, 4)
  - cout << dict[“green”];
  - cout << dict[3];
- » Funkcjonalność:
  - dodawanie nowych elementów (na końcu, na początku)
  - sprawdzanie czy element istnieje
  - iterowanie po elementach
  - obliczanie rozmiaru
  - przeciążenie operatora []

# Przykład: dict

- » **Test driven development**
- » **Zaczę od próby użycia**  
(nieistniejącej jeszcze implementacji)

# Przykład: dict

```
#include <iostream>
#include "dict.h"
```

```
using namespace std;
```

```
int main() {
    Dict dict;

    cout << dict.size() << endl;           // 0
    dict.push_back("red", 4);
    dict.push_back("green", 7);
    dict.push_back("black", 10);
    dict.push_front("white", 1);
    cout << dict.size() << endl;           // 4

    cout << dict["red"] << endl;           // 4
    cout << dict["white"] << endl;        // 1
    cout << dict.front() << endl;         // 1
    cout << dict.back() << endl;          // 10

    for (size_t i=0; i<dict.size(); ++i) {
        cout << dict[i] << endl;         // 1 4 7 10
    }

    return 0;
}
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)



# Przykład: dict

Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)

# Przykład: dict

Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)



```
cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;           // 1
cout << dict.back() << endl;           // 10
```

```
for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

# Przykład: dict

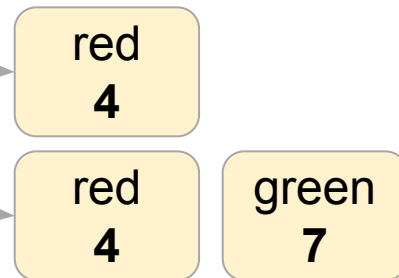
Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4
```

```
cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10
```

```
for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)



# Przykład: dict

Dict dict;

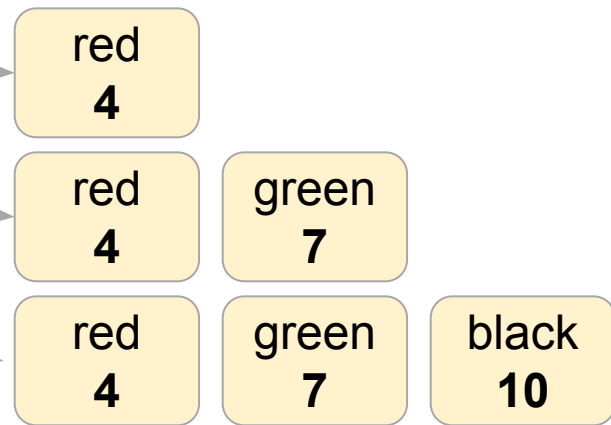
```

cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;          // 1 4 7 10
}
  
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)



# Przykład: dict

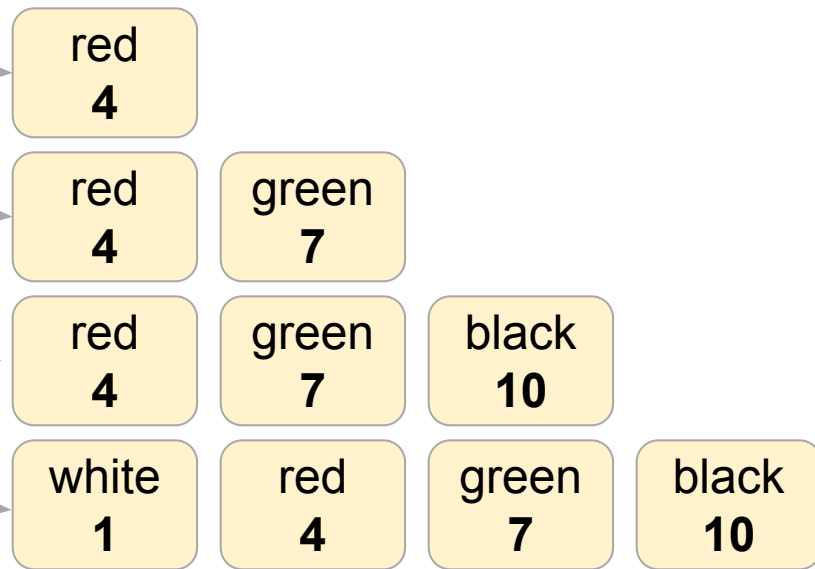
Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4
```

```
cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10
```

```
for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)



# Przykład: dict

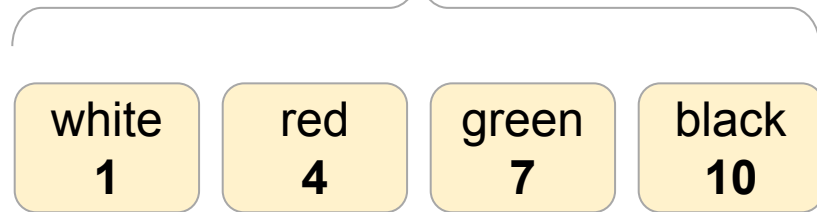
Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)
- » **liczba elementów**



# Przykład: dict

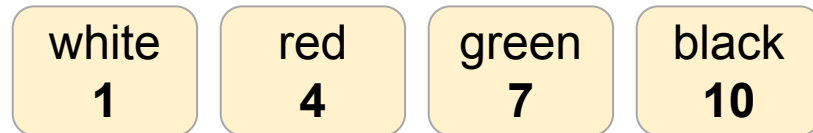
Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4
```

```
cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10
```

```
for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zacznę od próby użycia (nieistniejącej jeszcze implementacji)
- » liczba elementów
- » Indeksowanie **kluczem**



# Przykład: dict

Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4
```

```
cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10
```

```
for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)
- » liczba elementów
- » Indeksowanie **kluczem**

white  
1

red  
4

green  
7

black  
10



# Przykład: dict

Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;          // 1 4 7 10
}
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)
- » liczba elementów
- » Indeksowanie kluczem
- » Indeksowanie **pozycją**

white  
1

red  
4

green  
7

black  
10

# Przykład: dict

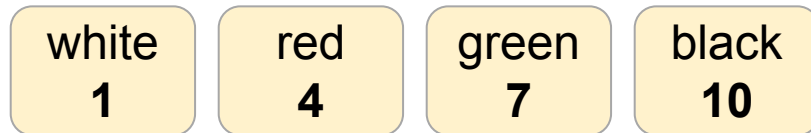
Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)
- » liczba elementów
- » Indeksowanie kluczem
- » Indeksowanie **pozycją**



# Przykład: dict

Dict dict;

```
cout << dict.size() << endl;           // 0
dict.push_back("red", 4);
dict.push_back("green", 7);
dict.push_back("black", 10);
dict.push_front("white", 1);
cout << dict.size() << endl;           // 4

cout << dict["red"] << endl;           // 4
cout << dict["white"] << endl;         // 1
cout << dict.front() << endl;          // 1
cout << dict.back() << endl;           // 10

for (size_t i=0; i<dict.size(); ++i) {
    cout << dict[i] << endl;           // 1 4 7 10
}
```

- » Test driven development
- » Zaczę od próby użycia (nieistniejącej jeszcze implementacji)
- » liczba elementów
- » Indeksowanie kluczem
- » Indeksowanie **pozycją**

white  
1

red  
4

green  
7

black  
10

# Przykład: dict - deklaracja

```
class Dict {
```

```
public:
```

```
    Dict(void);  
    void push_back(string key, int val);  
    void push_front(string key, int val);  
    size_t size(void);  
    int front(void);  
    int back(void);  
    const int operator[](string key);  
    const int operator[](size_t no);  
    ~Dict();  
};
```

- » Deklaracja klasy na podstawie “przypadków użycia”
- » Tylko metody
- » Typy mają się zgadzać
- » Na razie nie myślę jak będę implementował

# Przykład: dict - deklaracja

```
class Dict {  
    struct node{  
        string key;  
        int value;  
        node *next;  
    };  
    node *head_;  
    node *tail_;  
    size_t size_;  
public:  
    Dict(void);  
    void push_back(string key, int val);  
    void push_front(string key, int val);  
    size_t size(void);  
    int front(void);  
    int back(void);  
    const int operator[](string key);  
    const int operator[](size_t no);  
    ~Dict();  
};
```

- » Deklaracja klasy na podstawie “przypadków użycia”
- » Tylko metody
- » Typy mają się zgadzać
- » Na razie nie myślę jak będę implementował
- » Zaczynam myśleć o implementacji ;-)



AGH

# Przykład: dict - deklaracja

```
Dict::Dict(void) {  
}  
  
void Dict::push_back(string key, int val) {  
}  
  
void Dict::push_front(string key, int val) {  
}  
  
size_t Dict::size(void) {  
    return 0;  
}  
  
int Dict::front(void) {  
    return 0;  
}  
  
int Dict::back(void) {  
    return 0;  
}  
  
const int Dict::operator[](string key) {  
    return 0;  
}  
  
const int Dict::operator[](size_t no) {  
    return 0;  
}  
  
Dict::~Dict() {  
}
```

- » Deklaracja klasy na podstawie “przypadków użycia”
- » Tylko metody
- » Typy mają się zgadzać
- » Na razie nie myślę jak będę implementował
- » Zaczynam myśleć o implementacji ;-)
- » Zdefiniowałem wszystkie metody
- » Mogę pierwszy raz skompilować program

# Przykład: dict - definicje

```
Dict::Dict(void) :  
    head_(nullptr),  
    tail_(nullptr),  
    size_(0){  
}
```

- » Konstruktor
  - inicjalizacja składowych prywatnych (“stanu” obiektu)

# Przykład: dict - definicje

```
size_t Dict::size(void) {  
    return size_;  
}
```

```
int Dict::front(void) {  
    return head_>value;  
}
```

```
int Dict::back(void) {  
    return tail_>value;  
}
```

- » Trywialne metody
  - praktycznie są to “gettery”



# Przykład: dict - definicje

```
void Dict::push_back(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr){ // first
        head_ = new_node;
        tail_ = head_;
    } else { // already exist
        tail_->next = new_node;
        tail_ = new_node;
    }

    size_++;
}
```

- » Metoda push\_back()
- utwórz nowy element
  - zainicjalizuj

# Przykład: dict - definicje

```
void Dict::push_back(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr) { // first
        head_ = new_node;
        tail_ = head_;
    } else { // already exist
        tail_->next = new_node;
        tail_ = new_node;
    }

    size_++;
}
```

- » Metoda push\_back()
- utwórz nowy element
  - zainicjalizuj
  - jeżeli lista pusta
  - dodaj do head\_
  - tail\_ == head\_

# Przykład: dict - definicje

```
void Dict::push_back(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr) { // first
        head_ = new_node;
        tail_ = head_;
    } else { // already exist
        tail_>next = new_node;
        tail_ = new_node;
    }

    size_++;
}
```

- » Metoda push\_back()
- utwórz nowy element
  - zainicjalizuj
  - jeżeli lista pusta
  - dodaj do head\_
  - tail\_ == head\_
  - w kolekcji już są jakieś elementy
  - ostatni element bieżącej listy wskazuje na nowy
  - dodaj na końcu istniejącej listy

# Przykład: dict - definicje

```
void Dict::push_front(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr){
        head_ = new_node;
        tail_ = head_;
    } else {
        new_node->next = head_;
        head_ = new_node;
    }

    size_++;
}
```

- » Metoda push\_front()
- » Bardzo podobna do push\_back(), zamiast na końcu, dodaje na początku

# Przykład: dict - definicje

```
void Dict::push_front(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr){
        head_ = new_node;
        tail_ = head_;
    } else {
        new_node->next = head_;
        head_ = new_node;
    }

    size_++;
}
```

```
void Dict::push_back(string key, int val)
{
    node *new_node = new node;
    new_node->key = key;
    new_node->value = val;
    new_node->next = nullptr;

    if (head_ == nullptr){
        head_ = new_node;
        tail_ = head_;
    } else {
        tail_->next = new_node;
        tail_ = new_node;
    }

    size_++;
}
```

# Przykład: dict - definicje

```
const int Dict::operator[](string key) {  
    node *curr = head_;  
    while (curr != nullptr){  
        if (curr->key == key){  
            return curr->value;  
        }  
        curr = curr->next;  
    }  
    return 0;  
}
```

- » Operator[] (string)
- » Przejrzyj całą listę

# Przykład: dict - definicje

```
const int Dict::operator[](string key) {  
    node *curr = head_  
    while (curr != nullptr){  
        if (curr->key == key){  
            return curr->value;  
        }  
        curr = curr->next;  
    }  
    return 0;  
}
```

- » Operator[] (string)
- » Przejrzyj całą listę
- » Pracuję na kopii head\_  
 żebym mógł ją modyfikować

# Przykład: dict - definicje

```
const int Dict::operator[](string key) {  
    node *curr = head_;  
    while (curr != nullptr){  
        if (curr->key == key){  
            return curr->value;  
        }  
        curr = curr->next;  
    }  
  
    return 0;  
}
```

- » Operator[] (string)
- » Przejrzyj całą listę
- » Pracuję na kopii head\_  
 żeby móc ją modyfikować
- » Jeżeli klucz się zgadza,  
 zwróć wartość i zakończ



# Przykład: dict - definicje

```
const int Dict::operator[](string key) {  
    node *curr = head_  
    while (curr != nullptr){  
        if (curr->key == key){  
            return curr->value;  
        }  
        curr = curr->next;  
    }  
  
    return 0;  
}
```

- » Operator[] (string)
- » Przejrzyj całą listę
- » Pracuję na kopii head\_  
 żeby móc ją modyfikować
- » Jeżeli klucz się zgadza,  
 zwróć wartość i zakończ
- » Typ operatora (**const**)  
 oznacza, że jest on do  
 odczytu: `x=tab[i]` a nie do  
 zapisu `tab[i]=x`;

# Przykład: dict - definicje

```
const int Dict::operator[](string key) {  
    node *curr = head_;  
    while (curr != nullptr){  
        if (curr->key == key){  
            return curr->value;  
        }  
        curr = curr->next;  
    }  
    // TODO: not found, what next??  
    return 0;  
}
```

- » Operator[] (string)
- » Przejrzyj całą listę
- » Pracuję na kopii head\_  
 żeby móc ją modyfikować
- » Jeżeli klucz się zgadza,  
 zwróć wartość i zakończ
- » Typ operatora (const)  
 oznacza, że jest on do  
 odczytu:  $x = \text{tab}[i]$  a nie do  
 zapisu  $\text{tab}[i] = x$ ;
- » **Problem:** nie mam pomysłu  
 co zrobić jeżeli nie znajdę  
 klucza

# Przykład: dict - definicje

```
const int Dict::operator[](size_t no) {  
    node *curr = head_;  
    for (size_t i=0; i<no; ++i){  
        curr = curr->next;  
    }  
    return curr->value;  
}
```

- » Operator[] (int)
- » Pracuję na kopii head\_

# Przykład: dict - definicje

```
const int Dict::operator[](size_t no) {  
    node *curr = head_  
    for (size_t i=0; i<no; ++i){  
        curr = curr->next;  
    }  
    return curr->value;  
}
```

- » Operator[] (int)
- » Pracuję na kopii head\_
- » Przesuwam się po liście “no”  
razy

# Przykład: dict - definicje

```
const int Dict::operator[](size_t no) {  
    node *curr = head_  
    for (size_t i=0; i<no; ++i){  
        curr = curr->next;  
    }  
    return curr->value;  
}
```

- » Operator[] (int)
- » Pracuję na kopii head\_
- » Przesuwam się po liście “no”  
razy
- » Zwracam wartość bieżącego  
elementu

# Przykład: dict - definicje

```
const int Dict::operator[](size_t no) {  
    node *curr = head_  
    for (size_t i=0; i<no; ++i){  
        // TODO: assume, curr->next exist  
        curr = curr->next;  
    }  
    return curr->value;  
}
```

- » Operator[] (int)
- » Pracuję na kopii head\_
- » Przesuwam się po liście “no”  
razy
- » Zwracam wartość bieżącego  
elementu
- » **Problem:** ponownie nie  
wiem co zrobić jeżeli  
no >= size

# Przykład: dict - definicje

```
Dict::~~Dict() {  
    while (head_ != nullptr){  
        node *next = head_ -> next;  
        delete head_;  
        head_ = next;  
    }  
}
```

- » Destruktor
- » Iteruję od początku do końca

# Przykład: dict - definicje

```
Dict::~~Dict() {  
    while (head_ != nullptr){  
        node *next = head_ -> next;  
        delete head_;  
        head_ = next;  
    }  
}
```

- » Destruktor
- » Iteruję od początku do końca
- » Nie muszę pracować na kopii - obiekt jest właśnie niszczone



# Przykład: dict - definicje

```
Dict::~~Dict() {  
    while (head_ != nullptr){  
        node *next = head_ -> next;  
        delete head_;  
        head_ = next;  
    }  
}
```

- » Destruktor
- » Iteruję od początku do końca
- » Nie muszę pracować na kopii - obiekt jest właśnie niszczone
- » Palę za sobą mosty ;-)

# Przykład: dict - definicje

```
Dict::~~Dict() {  
    while (head_ != nullptr){  
        node *next = head_ -> next;  
        delete head_;  
        head_ = next;  
    }  
}
```

- » Destraktor
- » Iteruję od początku do końca
- » Nie muszę pracować na kopii - obiekt jest właśnie niszczone
- » Palę za sobą mosty ;-)
- » Ale wcześniej zapisuje gdzie jest następny węzeł

# Przykład: dict TODO

- » Wymyślić jak obsłużyć błędy adresowania (brak klucza lub nieistniejący indeks)
- » Przerobić na szablon (wartość dowolna nie tylko int)
- » Testy jednostkowe (UnitTest - np. googletest, boost, CppUnit)
- » Optymalizacje: push\_back(), push\_front() + testy wydajności i porównanie z std::list
- » Spróbować implementacji “iterowalnej”:
  - każdy obiekt jest Dict jest pojedynczym węzłem w liście a nie zawiera pełną listę (jak teraz)
  - można wtedy iterować dict = dict->next();
  - nie używać iteratorów z std::

# Narzędzia

G++ to nie jedyne narzędzie którego będziesz używał!!

# Przykład: błędy pamięci

```
const int Dict::operator[](size_t no) {    » Jest ok, nie ma błędu
    node *curr = head_;
    for (size_t i=0; i<no; ++i){
        curr = curr->next;
    }
    return curr->value;
}
```

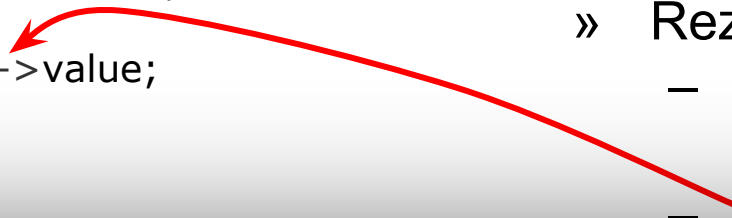
# Przykład: błędy pamięci

```
const int Dict::operator[](size_t no) {  
    node *curr = head_;  
    for (size_t i=0; i<=no; ++i){  
        curr = curr->next;  
    }  
    return curr->value;  
}
```

- » Jest ok, nie ma błędu
- » Zrobiłem typowy “off-by-one” błąd
- » Rezultat:
  - zbyt dużo razy iterowałem
  - NULL

# Przykład: błędy pamięci

```
const int Dict::operator[](size_t no) {  
    node *curr = head_;  
    for (size_t i=0; i<=no; ++i){  
        curr = curr->next;  
    }  
    return curr->value;  
}
```



- » Jest ok, nie ma błędu
- » Zrobiłem typowy “off-by-one” błąd
- » Rezultat:
  - zbyt dużo razy iterowałem
  - NULL

```
(...)  
1  
10  
4  
7  
10  
fish: './dict' zakończony sygnałem SIGSEGV (Address boundary error)
```

# Przykład: błędy pamięci

- » W przypadku błędu krytycznego, zrób sekcję zwłok :-)  
“postmortem debugging”
- » Program skompiluj z flagą “-g” (debugger)
- » Linux: w konsoli ustaw “**ulimit -c unlimited**”
- » System zapisze do pliku “**core**” zrzut pamięci w chwili zatrzymania programu



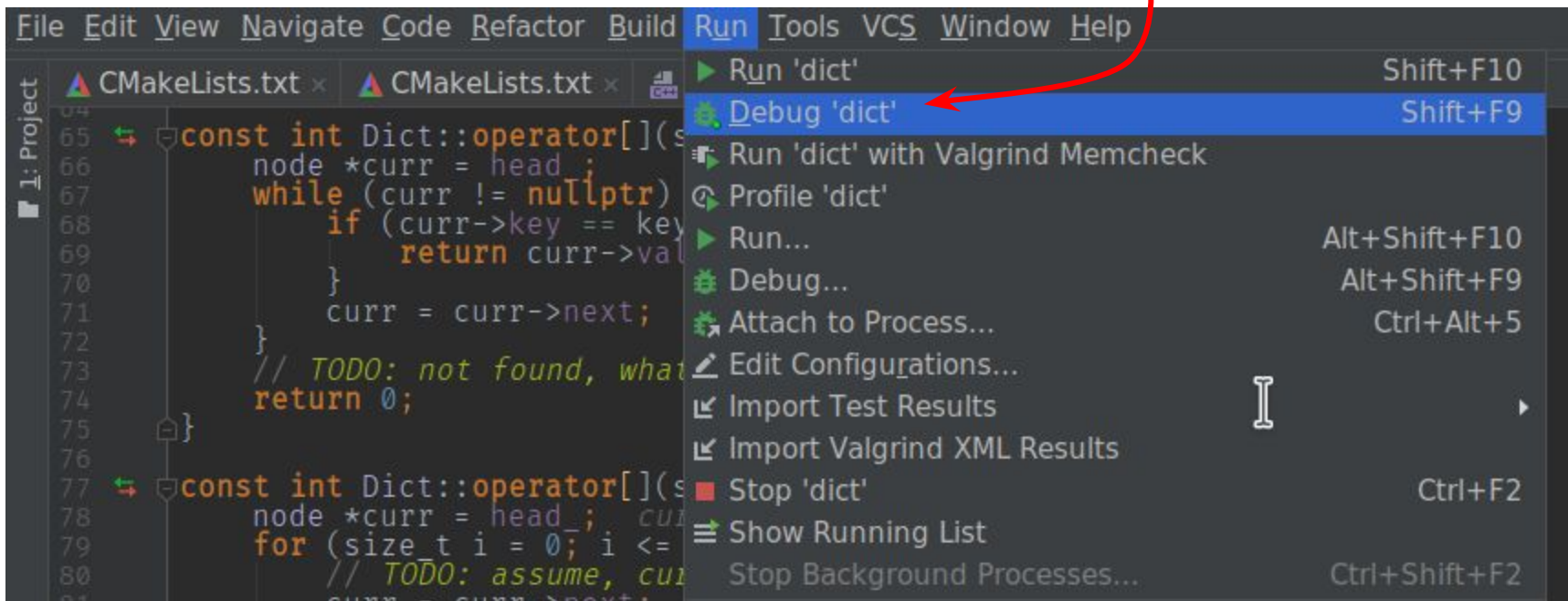
# Przykład: błędy pamięci

- » W przypadku błędu krytycznego, zrób sekcję zwłok :-)  
“postmortem debugging”
- » Program skompiluj z flagą “-g” (debugger)
- » Linux: w konsoli ustaw “**ulimit -c unlimited**”
- » System zapisze do pliku “**core**” zrzut pamięci w chwili zatrzymania programu

```
>gdb ./path/to/executable ./path/to/core
(...)
Reading symbols from ./dict...done.
[New LWP 14235]
Core was generated by `./dict'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055b80f52d624 in Dict::operator[] (this=0x7ffeef26a520, no=3) at
/home/kwant/Dropbox/PodstawyInformatyki/lab_14_dict/src/ dict.cpp:83
83          return curr->value;
```

# Przykład: błędy pamięci

» To samo można wywołać z IDE



Miejsce problemu

```
77 const int Dict::operator[](size_t no) { no: 3
78     node *curr = head_; curr: NULL
79     for (size_t i = 0; i <= no; ++i) {
80         // TODO: assume, curr->next exist (ie. "no" is correct)
81         curr = curr->next;
82     }
83     return curr->value;
84 }
85
86 Dict::~Dict() {
87     while (head_ != nullptr) {
88         node *next = head_->next;
89         delete head_;
90         head_ = next;
91     }
92 }
93
```

Dict::operator[]

Debug: dict x

Debugger Console

Frames

- Thread-1
- Dict::operator[] dict.cpp:83
- main main.cpp:31
- \_libc\_start\_main 0x00007f95d82ebb97
- \_start 0x000055f0d1b7bdea

Variables

GDB

Signal = SIGSEGV (Segmentation fault)

this = {Dict \* const | 0x7ffc27b476c0} 0x7ffc27b476c0

head\_ = {Dict::node \* | 0x55f0d2dae340} 0x55f0d2dae340

- key = {std::\_\_cxx11::string} "white"
- value = {int} 1
- next = {Dict::node \* | 0x55f0d2dae280} 0x55f0d2dae280

tail\_ = {Dict::node \* | 0x55f0d2dae300} 0x55f0d2dae300

- key = {std::\_\_cxx11::string} "black"
- value = {int} 10
- next = {Dict::node \* | 0x0} NULL
- size\_ = {size\_t} 4
- no = {size\_t} 3
- curr = {Dict::node \* | 0x0} NULL

Miejsce problemu

Jaki błąd

```
77 const int Dict::operator[](size_t no) { no: 3
78     node *curr = head_; curr: NULL
79     for (size_t i = 0; i <= no; ++i) {
80         // TODO: assume, curr->next exist (ie. "no" is correct)
81         curr = curr->next;
82     }
83     return curr->value;
84 }
85
86 Dict::~Dict() {
87     while (head_ != nullptr) {
88         node *next = head_->next;
89         delete head_;
90         head_ = next;
91     }
92 }
93
```

Dict::operator[]

Debug: dict x

Debugger Console

Frames

- Thread-1
- Dict::operator[] dict.cpp:83
- main main.cpp:31
- \_libc\_start\_main 0x00007f95d82ebb97
- \_start 0x000055f0d1b7bdea

Variables

Signal = SIGSEGV (Segmentation fault)

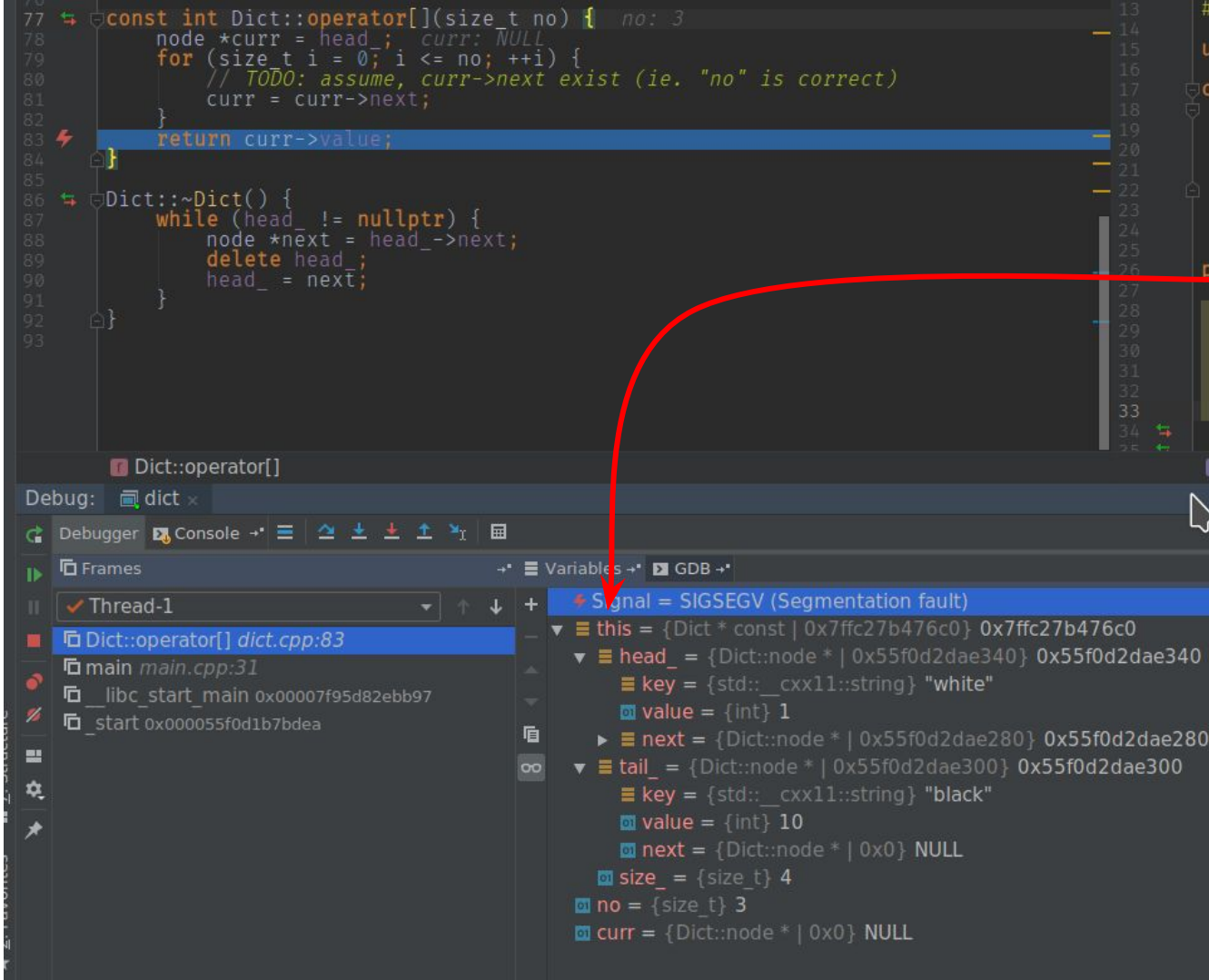
- this = {Dict \* const | 0x7ffc27b476c0} 0x7ffc27b476c0
- head\_ = {Dict::node \* | 0x55f0d2dae340} 0x55f0d2dae340
  - key = {std::\_\_cxx11::string} "white"
  - value = {int} 1
  - next = {Dict::node \* | 0x55f0d2dae280} 0x55f0d2dae280
- tail\_ = {Dict::node \* | 0x55f0d2dae300} 0x55f0d2dae300
  - key = {std::\_\_cxx11::string} "black"
  - value = {int} 10
  - next = {Dict::node \* | 0x0} NULL
  - size\_ = {size\_t} 4
- no = {size\_t} 3
- curr = {Dict::node \* | 0x0} NULL

Miejsce problemu

Jaki błąd

Stan programu:

- obiekty
- stany obiektów
- składowe klasy
- zmienne
- wartości tablic



```
77 const int Dict::operator[](size_t no) { no: 3
78     node *curr = head_; curr: NULL
79     for (size_t i = 0; i <= no; ++i) {
80         // TODO: assume, curr->next exist (ie. "no" is correct)
81         curr = curr->next;
82     }
83     return curr->value;
84 }
85
86 Dict::~Dict() {
87     while (head_ != nullptr) {
88         node *next = head_->next;
89         delete head_;
90         head_ = next;
91     }
92 }
93
```

Dict::operator[]

Debug: dict x

Debugger Console

Frames

- Thread-1
- Dict::operator[] dict.cpp:83
- main main.cpp:31
- \_libc\_start\_main 0x00007f95d82ebb97
- \_start 0x000055f0d1b7bdea

Variables

Signal = SIGSEGV (Segmentation fault)

- this = {Dict \* const | 0x7ffc27b476c0} 0x7ffc27b476c0
- head\_ = {Dict::node \* | 0x55f0d2dae340} 0x55f0d2dae340
  - key = {std::\_\_cxx11::string} "white"
  - value = {int} 1
  - next = {Dict::node \* | 0x55f0d2dae280} 0x55f0d2dae280
- tail\_ = {Dict::node \* | 0x55f0d2dae300} 0x55f0d2dae300
  - key = {std::\_\_cxx11::string} "black"
  - value = {int} 10
  - next = {Dict::node \* | 0x0} NULL
  - size\_ = {size\_t} 4
- no = {size\_t} 3
- curr = {Dict::node \* | 0x0} NULL

# Przykład: błędy pamięci

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)

# Przykład: błędy pamięci

```
valgrind ./dict
```

```
==28591== Memcheck, a memory error detector
==28591== Copyright (C) 2002-2017, and GNU
GPL'd, by Julian Seward et al.
==28591== Using Valgrind-3.13.0 and LibVEX;
rerun with -h for copyright info
==28591== Command: ./dict
==28591==
0
4
(...)
7
10
==28591== Invalid read of size 4
==28591==    at 0x109624:
Dict::operator[](unsigned long) (dict.cpp:83)
==28591==    by 0x10920B: main (main.cpp:31)
==28591==   Address 0x20 is not stack'd, malloc'd
or (recently) free'd
==28591==
==28591==
```

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)
- » Uruchamianie

# Przykład: błędy pamięci

```
valgrind ./dict
==28591== Memcheck, a memory error detector
==28591== Copyright (C) 2002-2017, and GNU
GPL'd, by Julian Seward et al.
==28591== Using Valgrind-3.13.0 and LibVEX;
rerun with -h for copyright info
==28591== Command: ./dict
==28591==
0
4
(...)
7
10
==28591== Invalid read of size 4
==28591==      at 0x109624:
Dict::operator[](unsigned long) (dict.cpp:83)
==28591==      by 0x10920B: main (main.cpp:31)
==28591==   Address 0x20 is not stack'd, malloc'd
or (recently) free'd
==28591==
==28591==
```

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)
- » Uruchamianie
- » Konfiguracja



# Przykład: błędy pamięci

```
valgrind ./dict
==28591== Memcheck, a memory error detector
==28591== Copyright (C) 2002-2017, and GNU
GPL'd, by Julian Seward et al.
==28591== Using Valgrind-3.13.0 and LibVEX;
rerun with -h for copyright info
==28591== Command: ./dict
==28591==
0
4
(...)
7
10
==28591== Invalid read of size 4
==28591==      at 0x109624:
Dict::operator[](unsigned long) (dict.cpp:83)
==28591==      by 0x10920B: main (main.cpp:31)
==28591== Address 0x20 is not stack'd, malloc'd
or (recently) free'd
==28591==
==28591==
```

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)
- » Uruchamianie
- » Konfiguracja
- » Wynik działania programu

# Przykład: błędy pamięci

```
valgrind ./dict
==28591== Memcheck, a memory error detector
==28591== Copyright (C) 2002-2017, and GNU
GPL'd, by Julian Seward et al.
==28591== Using Valgrind-3.13.0 and LibVEX;
rerun with -h for copyright info
==28591== Command: ./dict
==28591==
0
4
(...)
7
10
==28591== Invalid read of size 4
==28591==    at 0x109624:
Dict::operator[](unsigned long) (dict.cpp:83)
==28591==    by 0x10920B: main (main.cpp:31)
==28591== Address 0x20 is not stack'd, malloc'd
or (recently) free'd
==28591==
==28591==
```

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)
- » Uruchamianie
- » Konfiguracja
- » Wynik działania programu
- » Wynik analizy: co jest złe

# Przykład: błędy pamięci

```
valgrind ./dict
==28591== Memcheck, a memory error detector
==28591== Copyright (C) 2002-2017, and GNU
GPL'd, by Julian Seward et al.
==28591== Using Valgrind-3.13.0 and LibVEX;
rerun with -h for copyright info
==28591== Command: ./dict
==28591==
0
4
(...)
7
10
==28591== Invalid read of size 4
==28591==      at 0x109624:
Dict::operator[](unsigned long) (dict.cpp:83)
==28591==      by 0x10920B: main (main.cpp:31)
==28591==   Address 0x20 is not stack'd, malloc'd
or (recently) free'd
==28591==
==28591==
```

- » Varglind (dynamic analysis tools)
  - wycieki pamięci
  - buffer overflow error
- » Linux, Android, ~~Windows~~
- » Program działa 10x wolniej (nie nadaje się do analizy w real-time)
- » Uruchamianie
- » Konfiguracja
- » Wynik działania programu
- » Wynik analizy: co jest źle
- » Wynik analizy: gdzie jest źle

# IDE

- profiler
- debugger

# IDE

- generate definition
- rename
- setter/getter
- extract function
- autogen for, if, while (code snipped)

# ToDo

czy to wszystko?

# Dziękuję