



QPlayer 사용자 매뉴얼

Version 2.3-Cougar, Jan '24



ETRI 한국전자통신연구원
Electronics and Telecommunications
Research Institute

Lightweight, Scalable, and Fast Quantum Emulator

CONTENTS

CHAPTER 1	QPlayer 소개	1
개념		2
사용자 인터페이스		2
버전 히스토리		3
용어		4
라이선스		5
연락처		5
CHAPTER 2	소프트웨어 설치	6
운영환경		7
운영체제		7
하드웨어 제약사항		7
소프트웨어 설치		7
단계1: QPlayer 소스코드 다운로드		7
단계2: QPlayer 소스코드 컴파일		8
설치 후 검증		8
CHAPTER 3	Command-Line Interface (CLI)	9
명령어 규격		10
사용 시나리오		10
유스케이스1: QPlayer 버전 검사		10
유스케이스2: OpenQASM 2.0 양자 프로그램 실행		10
유스케이스3: OpenQASM 2.0 양자 프로그램 반복 실행		11
유스케이스4: 시뮬레이션 실행 정보를 JSON 파일로 출력		11
유스케이스5: 시뮬레이션 실행 정보를 콘솔 화면에 출력		12
예외 처리		14
메모리 오버플로우		14
OpenQASM 코드 문법 에러		14
CHAPTER 4	Qiskit-compatible python Interface	15
python 인터페이스 설치		16
python 기반 양자 프로그램 개발		16
python 기반 코드 작성		16

python 기반 코드 실행.....	17
Jupyter Notebook 활용	17
conda-forge 설치.....	18
conda 가상환경 생성 및 활성화	18
Jupyter notebook 설치.....	18
conda 가상환경의 필요 패키지 설치.....	18
Jupyter notebook 실행.....	18
Jupyter notebook 기반 python 코드 작성.....	19
Jupyter notebook 기반 python 코드 실행.....	20
CHAPTER 4 Native C-like Interface	21
클래스	22
QRegister Class	22
QTimer Class	22
게이트 함수.....	23
initZ(QRegister *QReg, int qubit).....	23
initX(QRegister *QReg, int qubit).....	23
X(QRegister *QReg, int qubit)	23
Z(QRegister *QReg, int qubit)	23
Y(QRegister *QReg, int qubit)	24
H(QRegister *QReg, int qubit).....	24
S(QRegister *QReg, int qubit).....	24
T(QRegister *QReg, int qubit).....	24
SDG(QRegister *QReg, int qubit)	24
TDG(QRegister *QReg, int qubit).....	24
P(QRegister *QReg, int qubit, double angle)	25
U1(QRegister *QReg, int qubit, double lambda)	25
U2(QRegister *QReg, int qubit, double phi, double lambda)	25
U3(QRegister *QReg, int qubit, double theta, double phi, double lambda)	25
RX(QRegister *QReg, int qubit, double angle)	25
RY(QRegister *QReg, int qubit, double angle)	25
RZ(QRegister *QReg, int qubit, double angle)	26
CU1(QRegister *QReg, int control, int target, double lambda)	26
CU2(QRegister *QReg, int control, int target, double phi, double lambda).....	26

CU3(QRegister *QReg, int control, int target, double theta, phi, lambda).....	26
CH(QRegister *QReg, int control, int target)	26
CX(QRegister *QReg, int control, int target).....	26
CZ(QRegister *QReg, int control, int target).....	27
CY(QRegister *QReg, int control, int target).....	27
CRZ(QRegister *QReg, int control, int target, double angle).....	27
CCX(QRegister *QReg, int control1, int control2, int target).....	27
SWAP(QRegister *QReg, int qubit1, int qubit2)	27
CSWAP(QRegister *QReg, int control, int qubit1, int qubit2).....	27
SX(QRegister *QReg, int qubit)	28
iSWAP(QRegister *QReg, int qubit1, int qubit2)	28
측정 함수.....	28
int M(QRegister *QReg, int qubit).....	28
int MF(QRegister *QReg, int qubit, int state)	28
유틸리티 함수	29
void dump(QRegister *QReg).....	29
void getMemory(uint64_t *memTotal, uint64_t *memAvail, uint64_t *memUsed) ..	29
양자 프로그램 예제.....	29
Quantum hello world !!.....	29
QFT algorithm.....	30

CHAPTER 1 QPlayer 소개

이 장에서는...

- 개념
- 사용자 인터페이스
- 버전 히스토리
- 용어
- 라이선스
- 연락처

QPlayer 는 ETRI (Electronics and Telecommunications Research Institute)에서 자체적으로 개발한 양자컴퓨팅 에뮬레이션 SW 이다. QPlayer 는 고전 컴퓨터를 활용해서 양자 회로를 시뮬레이션 하기 위한 새로운 방법을 제공한다. QPlayer 는 기존 양자 에뮬레이터들과 다른 고유의 양자 상태 관리를 통해 적은 메모리 자원만으로도 더 많은 큐비트들의 양자 회로에 대한 빠른 시뮬레이션을 가능하게 한다.

이 장에서는 QPlayer 의 주요 개념과 인터페이스, 버전 별 개발 히스토리, 라이선스 정책 등을 소개한다.

개념

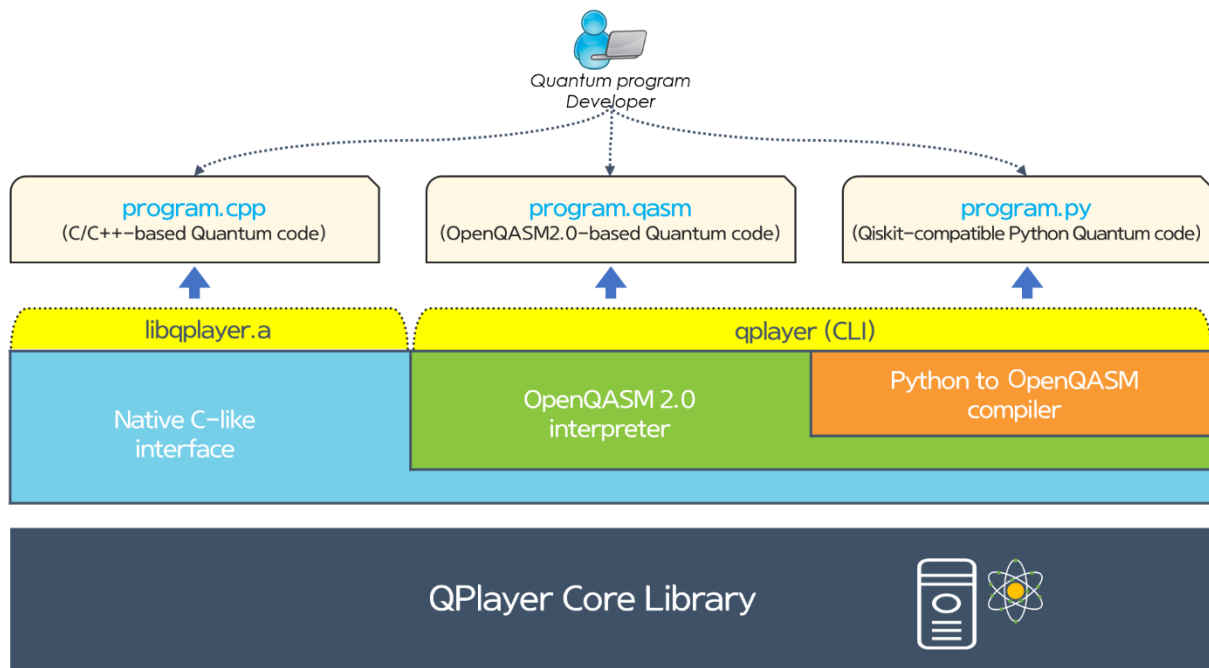
양자 컴퓨팅의 급속한 발전에 따라 디지털 양자 시뮬레이션은 양자 알고리즘 검증, 양자 오류 분석 및 새로운 양자 응용에 필수적인 것으로 간주되고 있다. 그러나 메모리 오버헤드와 양자 연산 시간의 기하 급수적인 증가는 수년 동안 해결되지 않은 어려운 문제로 남아있다. QPlayer는 이전보다 더 작은 메모리로 더 많은 큐비트와 더 빠른 양자 작업을 제공하는 새로운 접근 방식을 제공한다. QPlayer는 전체 양자 상태를 메모리에 저장하는 대신 축소된 양자 상태 표현 체계를 사용하여 물리적으로 의미를 가진 양자 상태들을 선택적으로 관리한다. 이러한 접근 방법은 전역 양자 상태를 손상시키지 않으면서도 적은 메모리 공간으로도 빠른 양자 계산을 보장하는 장점을 가진다. 또한, QPlayer를 이용한 실험 결과들은 다양한 양자 알고리즘들에 대해 기존 양자 에뮬레이터들의 한계를 극복하고 있음을 보여준다. Grover 알고리즘의 경우 최대 55 큐비트의 시뮬레이션을 지원하고, 표면 코드(Surface Code) 알고리즘은 512GB 메모리를 갖는 단일 컴퓨팅 노드에서 최대 85 큐비트로 시뮬레이션 될 수 있음을 실험적으로 검증하였다.

SEE ALSO:

"QPlayer: Lightweight, scalable, and fast quantum simulator", ETRI Journal, 2022

"Multilayered logical qubits and synthesized quantum bits", Quantum Science and Technology, 2023

사용자 인터페이스



QPlayer는 리눅스 계열의 운영체제(CentOS 7.x 이상, Ubuntu 20.x 이상)를 탑재한 컴퓨터에서 동작하는 소프

트웨어이다. QPlayer는 양자 시뮬레이션을 위한 코어 라이브러리와, 사용자가 이를 활용하여 양자 응용 프로그램을 작성하기 위한 인터페이스로 구성되어 있다. QPlayer에서 제공하는 프로그래밍 인터페이스들의 종류와 그 특성들은 아래 표와 같다.

특성	Native C-like	OpenQASM 2.0	Python
주요 사용자	IT 개발자	비 IT 개발자	IT 개발자
프로그램 개발 난이도	높음	낮음	보통
양자 코드 컴파일 유무	필요	불필요 (직접 실행)	불필요 (직접 실행)
양자 프로그램 실행 형태	컴파일한 바이너리 실행	전용 CLI 활용	전용 CLI 활용
양자 코드 개발 용이성	높음 (모든 C/C++ 규칙 지원)	낮음 (제한적 함수와 if 조건문으로 한정)	높음 (Qiskit compatible)
적용 버전	v0.2-Cat 이후	v1.0-Leopard 이후	v2.3-Cougar 이후

버전 히스토리

v0.2-Cat

- 배포일자: 2021년 6월 24일
- 주요특징
 - 슈뢰딩거 스타일 상태 벡터 양자 시뮬레이션
 - 축소 힐버트 공간을 이용한 고속 양자 연산 처리
 - 사용자 친화적인 native-C 스타일 인터페이스
 - 21개 양자 게이트 연산 지원

v0.5-Jaguar

- 배포일자: 2023년 4월 30일
- 주요특징
 - 매트릭스 계산 성능 최적화
 - 롱런 시험시 메모리 누수 현상 개선
 - OpenQASM v2.0 지원 (기본적 문법 검사 처리)
 - 32개 양자 게이트 연산 지원

v1.0-Leopard

- 배포일자: 2023년 6월 23일
- 주요특징

- QPlayer 소스 코드 디렉토리 체계 재구성
- OpenQASM v2.0 지원 (문법 및 예외 처리 100% 지원)
- 동적 메모리의 효율적 처리를 위한 상태 벡터 메모리 풀 지원
- JSON 파일을 통한 시뮬레이션 실행의 통계 정보 출력

v2.0-Puma

- 배포일자: 2023년 9월 1일
- 주요특징
 - 게이트 특성에 기반한 고속 양자 연산 알고리즘 지원
 - 대각 게이트, 비대각 게이트 양자 연산 선별 동작

v2.1-Cheetah

- 배포일자: 2023년 10월 19일
- 주요특징
 - 병렬 쓰레드간 잠금 제거
 - boost library 의존성 제거
 - 양자 상태 표현을 위한 __uint128 bit 자료형 지원

v2.3-Cougar

- 배포일자: 2024년 1월 16일
- 주요특징
 - Qiskit-compatible python 인터페이스 제공

용어

본 문서에서 사용한 용어들은 아래 표와 같다.

큐비트	중첩과 얽힘의 양자 역학적 현상을 이용하는 양자 정보 처리의 기본 최소 단위. 큐비트는 특정 고유 상태(Eigen state)들이 확률적으로 동시에 공존함
중첩	실생활에서 파동의 특성을 갖는 중첩은 양자역학에서 보편적 원리로, 한 개의 큐비트가 고유 상태의 확률적 선형 결합으로 표현되는 상태
얽힘	임의 큐비트가 다른 큐비트와 독립적으로 존재하지 않고 상호 의존성 갖는 상태로 존재하는 현상. 즉, 두 개 이상의 큐비트가 상호 연관성을 가지고 있는 상태
측정	큐비트의 중첩 상태가 큐비트를 구성하고 있는 특정한 한 가지 고유 상태로 확정(붕괴)되는 행위. 측정 후 확정된 양자 상태는 이전 상태로 환원이 불가능한 양자역학적 비가역성(Irreversibility)이 적용됨

양자 에뮬레이터	양자 에뮬레이터는 양자 컴퓨터의 동작과 양자 시스템의 특성을 모사하고 시뮬레이션 하기 위한 소프트웨어 도구임. 양자 에뮬레이터는 양자 회로와 양자 게이트의 동작을 고전 컴퓨터로 시뮬레이션 하고, 초기 양자 상태와 양자 연산에 의한 양자 상태 진화를 추적함
양자 게이트	큐비트의 상태에 적용하는 양자적 연산을 의미하며, 게이트가 적용되는 큐비트 수에 따라 1-큐비트 게이트, 2-큐비트 게이트 등이 있음
양자 회로	양자 컴퓨터 및 양자 시스템에서 양자적 계산을 위해 설계된 일련의 양자 게이트 연산으로 구성된 계산 모델

라이선스

This source code is licensed (Dual License: GPL v3 & Commercial) under the license found in the LICENSE file in the root directory of this source tree.

GPL v3 License

When you copy and distribute the library in object code or executable form, you should accompany it with the complete corresponding machine-readable source code under the GPL v3 license at

<https://www.gnu.org/licenses/gpl-3.0.html>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Commercial License

If you don't want to open your source code in the course of copying and/or distribution, please consult <gicha@etri.re.kr> to get a permission with applicable conditions.

연락처

QPlayer 개발팀은 양자 연구원들의 다양한 요구사항을 수용하기 위해 항상 최선의 노력을 기울이고 있습니다. 기능 확장, 버그 보고서, 공동 연구 또는 기타 문제와 관련하여 궁금한 사항이 있으면 다음 채널을 통해 언제든지 문의해 주시기 바랍니다.

- Github: <https://github.com/eQuantumOS/QPlayer.git>
- Email: ksjin@etri.re.kr or gicha@etri.re.kr
- Phone: +82-042-860-1535 (1259)

CHAPTER 2 소프트웨어 설치

이 장에서는...

- 운영환경
- 소프트웨어 설치
- 설치 후 검증

QPlayer 는 리눅스 환경에서 동작하는 SW 이다. 공개 소프트웨어 라이선스인 GPL v3 정책을 준수하는 범위 내에서 누구나 자유롭게 소스 코드를 다운로드하여 설치 및 운용할 수 있다.

이 장에서는 QPlayer 의 소스코드 구조, 설치 방법 및 설치 후 검증 방법 등에 대해 설명한다.

운영환경

운영체제

QPlayer는 리눅스 계열 운영체제 중 CentOS 7.x 이상, Ubuntu 20.x 이상의 환경에서 동작한다.

하드웨어 제약사항

QPlayer는 단일 서버 환경에서 동작하도록 개발되었으며, 이를 운용하기 위한 하드웨어 제약사항은 아래 표와 같다.

구분	요구사항	비고
CPU	제약 없음	빠를수록 시뮬레이션 성능 향상
GPGPU	지원하지 않음	v3.0-Panther 이후 버전에서부터 GPGPU를 이용한 성능 가속을 지원할 예정
Memory	1 GB 이상	QPlayer는 양자 상태 진화에 따라 메모리가 동적으로 확장 또는 축소되는 특성이 있다. 메모리가 클수록 더 많은 큐비트들을 시뮬레이션 할 수 있으며, 30큐비트 이내의 양자 시뮬레이션을 위해서는 최소 1GB 이상의 메모리를 권장한다.
Storage, Network	제약 없음	

소프트웨어 설치

단계1: QPlayer 소스코드 다운로드

아래와 같이 QPlayer의 최신 소스 코드를 github 저장소에서 다운로드 한다.

```

>> git clone https://github.com/eQuantumOS/QPlayer.git
>> ls -l QPlayer
drwxr-xr-x 4 root root 4096 7월 3 10:41 core/
drwxr-xr-x 2 root root 4096 7월 3 10:39 docs/
drwxr-xr-x 5 root root 4096 7월 5 13:31 qasm/
drwxr-xr-x 5 root root 4096 6월 23 13:55 test/
-rw-r--r-- 1 root root 452 6월 20 09:22 AUTHOR
-rw-r--r-- 1 root root 795 6월 20 09:22 CONTRIBUTION
-rw-r--r-- 1 root root 35149 6월 20 09:22 LICENSE
-rw-r--r-- 1 root root 911 6월 23 13:55 Makefile
-rw-r--r-- 1 root root 2996 7월 5 13:31 README.md

```

단계2: QPlayer 소스코드 컴파일

아래와 같이 QPlayer의 소스 코드를 컴파일 한다.

```

>> cd QPlayer
>> make
>> ls -l
drwxr-xr-x 4 root root 4096 7월 3 10:41 core/
drwxr-xr-x 2 root root 4096 7월 3 10:39 docs/
drwxr-xr-x 5 root root 4096 7월 5 13:31 qasm/
drwxr-xr-x 5 root root 4096 6월 23 13:55 test/
drwxr-xr-x 5 root root 4096 6월 23 13:55 release/
-rw-r--r-- 1 root root 452 6월 20 09:22 AUTHOR
-rw-r--r-- 1 root root 795 6월 20 09:22 CONTRIBUTION
-rw-r--r-- 1 root root 35149 6월 20 09:22 LICENSE
-rw-r--r-- 1 root root 911 6월 23 13:55 Makefile
-rw-r--r-- 1 root root 2996 7월 5 13:31 README.md
>> ls -l release
drwxr-xr-x 5 root root 4096 7월 5 13:31 bin/
drwxr-xr-x 5 root root 4096 7월 5 13:31 include/
drwxr-xr-x 5 root root 4096 7월 5 13:31 lib/

```

컴파일이 완료되면 release 디렉토리가 새롭게 생성되며, 그 하위에 bin, include, lib 디렉토리를 포함한다. 하위 디렉토리들 중에서 release/bin 디렉토리는 OpenQASM 2.0 또는 python 코드를 실행하기 위한 CLI 파일을, release/include, release/lib에는 native-C를 이용하여 양자 프로그래밍을 할 수 있는 라이브러리 및 헤더 파일들이 저장되어 있다.

설치 후 검증

모든 설치 과정이 정상적으로 완료되었는지 확인하기 위해 아래 테스트를 실행한다. 아래와 같은 결과 메시지가 출력되면 설치된 SW의 기능이 정상적으로 동작함을 의미한다.

```

>> make TEST
>> cd test/verification
>> ./install_test
[P=0.250000] [+0.500000, +0.000000] |0000000000100000>
[P=0.250000] [+0.500000, +0.000000] |0000000000111110>
[P=0.250000] [-0.500000, +0.000000] |0000001111100000>
[P=0.250000] [-0.500000, +0.000000] |0000001111111110>
===== dump quantum states(4) =====

```

※ 각주: 상기에서 각 필드의 의미를 양자 상태 |0000000000100000>를 예로 들어 아래와 같이 설명한다.

[P=0.250000] → 양자 상태의 존재 확률

[+0.500000, +0.000000] → 양자 상태의 진폭 (+0.5: 실수부, +0.0: 허수부)

CHAPTER 3 Command-Line Interface (CLI)

이 장에서는...

- 명령어 규격
- 사용 시나리오
- 예외 처리

QPlayer 는 OpenQASM 2.0 기반으로 작성된 양자 프로그램 동작을 지원하기 위해 전용의 실행 바이너리인 'QPlayer' 명령어를 제공한다. 해당 명령어는 리눅스 콘솔에서 직접 수행하거나 또는 웹기반의 GUI 도구와 연동을 위해 활용할 수 있다.

이 장에서는 QPlayer 명령어의 상세 규격 및 사용 시나리오, 예외 처리 규칙 등을 설명한다.

명령어 규격

OpenQASM 2.0으로 작성된 양자 코드의 실행을 위해서는 전용의 CLI(QPlayer)를 사용해야 한다. 실행 파일은 소스 코드 컴파일이 완료된 후 ~release/bin 디렉토리에서 찾을 수 있으며 사용 규격은 아래 표와 같다.

옵션	인자	설명	속성
-i	QASM 입력 파일 경로	OpenQASM 2.0으로 작성된 양자 코드	필수
-o	결과 출력 파일 경로	시뮬레이션 결과 출력 파일	필수
-j	json 출력 파일 경로	시뮬레이션 실행 정보를 json 파일로 출력한다.	옵션
-s	실행 횟수	시뮬레이션 실행 횟수 (기본값: 1)	옵션
-h	N/A	실행 파일의 도움말을 화면에 출력한다	옵션
--version	N/A	QPlayer의 버전을 화면에 출력한다.	옵션
--verbose	N/A	시뮬레이션 실행 정보를 화면에 출력한다.	옵션

사용 시나리오

유스케이스1: QPlayer 버전 검사

배포된 QPlayer의 버전 정보는 아래와 같이 확인할 수 있다.

```

>> cd ~/release/bin
>> ./QPlayer --version

QPlayer v2.3-Cougar

```

유스케이스2: OpenQASM 2.0 양자 프로그램 실행

다음은 OpenQASM 2.0으로 작성된 양자 회로를 실행하는 예제이다. QPlayer를 컴파일한 후 생성된 release/bin 디렉토리에서 아래 내용과 동일하게 실행해 볼 수 있다.

```

>> cd ~/release/bin
>> ./QPlayer -i examples/test.qasm -o log/simulation.res
>> cat log/simulation.res

Total Shots: 1
Measured States: 1

100%, 1/1, |1011>

```

유스케이스3: OpenQASM 2.0 양자 프로그램 반복 실행

유스케이스#2는 주어진 코드를 기본적으로 1회만 실행하는 것이다. 만약 동일한 코드를 반복적으로 실행하고 싶을 경우 아래와 같이 사용할 수 있다. 이 경우 개별 실행시마다 측정 결과가 달라질 수 있으며, 이에 대한 확률 분포를 결과 로그 파일을 통해 확인할 수 있다.

```
» ./QPlayer -i examples/test.qasm -o ./log/simulation.res -s 10
» cat log/simulation.res
```

```
Total Shots: 10
Measured States: 8
```

```
10%, 1/10, |0010>
10%, 1/10, |0011>
20%, 2/10, |0110>
10%, 1/10, |0111>
10%, 1/10, |1010>
10%, 1/10, |1011>
20%, 2/10, |1110>
10%, 1/10, |1111>
```

유스케이스4: 시뮬레이션 실행 정보를 JSON 파일로 출력

QPlayer는 측정된 양자 상태와 별도로 시뮬레이션 실행 과정에서의 상태 정보들에 대해 별도의 JSON 파일로 생성하는 기능을 제공한다. 출력하는 정보의 범주는 1)실행 양자 회로 정보, 2)실행시간, 3)실행 작업, 4)시스템 정보 등으로 구성된다. 아래는 JSON 파일 생성을 위한 옵션의 실행 예제이다.

```
» ./QPlayer -i examples/test.qasm -o ./log/simulation.res -j ./log/task.json
```

SEE ALSO: JSON Specification

```
{
  "circuit": {
    "used qubits": Number,
    "used gates": Number,
    "gate calls": {
      "Gate": Number,
      "Gate": Number,
      "Gate": Number,
    }
  },
  "runtime": {
    "total simulation time": Number,
    "individual gate time": {
      "Gate": [total(Number), max(Number), min(Number), avg(Number)],
      "Gate": [total(Number), max(Number), min(Number), avg(Number)],
    }
  }
}
```

```

        "Gate" : [total(Number), max(Number), min(Number), avg(Number)]
    },
    "simulation jobs" : {
        "max states" : Number,
        "final states" : Number,
        "used memory" : String
    },
    "system" : {
        "OS" : {
            "name" : String,
            "version" : String
        },
        "CPU" : {
            "model" : String,
            "cores" : Number,
            "herz" : String
        },
        "Memory" : {
            "total" : String,
            "avail" : String
        }
    }
}

```

※ runtime에서 시간 단위는 ms임

※ runtime에서 total, max, min, avg는 해당 게이트가 실행된 총시간, 최대시간, 최소시간, 평균시간임

유스케이스5: 시뮬레이션 실행 정보를 콘솔 화면에 출력

유스케이스#4에서 생성된 JSON 파일은 별도의 뷰어를 통해서 확인이 가능하다. 그러나 작업 콘솔에서 vi 등으로 열람하기에 직관성이 부족하기 때문에 동일한 내용을 작업 콘솔에서 규격화된 형태로 출력하는 기능을 별도로 제공한다. 다음 화면은 --verbose 옵션을 통해 실행한 스크린 화면을 캡처한 예제이다.

1. Circuit Information

1. used qubits	4
2. used gates	9
3. total gate calls	27
4. indivisual gate calls	
H	4 (14 %)
S	4 (14 %)
SDT	4 (14 %)
T	4 (14 %)
TDG	4 (14 %)
RX	1 (3 %)
RY	1 (3 %)
RZ	1 (3 %)
MEASURE	4 (14 %)

2. Runtime (micro seconds)

1. total simulation time	658			
2. each gate execution time	total	max	min	avg
H	257	190	18	64
S	75	19	18	19
SDT	75	19	18	19
T	74	19	18	19
TDG	74	19	18	18
RX	21	21	21	21
RY	19	19	19	19
RZ	20	20	20	20
MEASURE	44	15	9	11

3. Simulation Jobs

1. max number of quantum states	16
2. final number of quantum states	1
3. used memory	4.8 MB

4. System Information

OS	name	Ubuntu
	version	20.04.3 LTS (Focal Fossa)
CPU	model	11th Gen Intel(R) Core(TM) i7-11700KF @ 3.60GHz
	cores	2
	herz	3600.002
MEM	total	3.8 GB
	avail	1.8 GB

예외 처리

메모리 오버플로우

대다수의 고전적 양자 에뮬레이터들은 큐비트 수에 따라 필요한 메모리 용량이 지수적으로 증가하는 문제점을 안고 있다. 반면 QPlayer의 경우 물리적 실체를 가지는 양자 상태만을 선별적으로 메모리에 보관하기 때문에 양자 회로의 전개에 따라 메모리가 동적으로 변동하는 고유의 특성을 가진다. 따라서, 시스템이 메모리 과부하로 인한 비정상 상태에 빠지는 상황을 방지하기 위해 양자 상태의 수와 메모리 잔여량을 상시 비교하여 메모리 공간이 부족할 것으로 예측되는 경우 메모리 부족 경고 메시지와 함께 프로세스를 강제 종료한다.

OpenQASM 코드 문법 에러

사용자가 작성한 QASM code에서 문법 오류가 발생한 경우 실행 콘솔에 아래 표와 같은 로그를 출력함과 동시에 실행 프로세스를 종료한다.

순번	에러 메시지
1	ERROR: undefined OPENQASM 2.0
2	ERROR: undefined include
3	ERROR: expected □□ in line-○
4	ERROR: argument is not a qreg in line-○
5	ERROR: argument is not a creg in line-○
6	ERROR: index of qreg is out of bound in line-○
7	ERROR: index of creg is out of bound in line-○
8	ERROR: unexpected numerical argument in line-○
9	ERROR: undefined gate or code string in line-○
10	ERROR: invalid gate in line-○

CHAPTER 4 Qiskit-compatible python Interface

이 장에서는...

- python 인터페이스 설치
- python 기반 양자 프로그램
- Jupyter Notebook 활용

이장에서는 Qiskit-compatible python 기반 양자 프로그램의 작성 및 실행 인터페이스의 사용법을 제공한다.

python 으로 작성된 코드는 Linux 환경의 콘솔 터미널에서 실행할 수 있으며, 별도의 Jupyter notebook 을 통해서도 사용할 수 있다.

python 인터페이스 설치

QPlayer에서 Qiskit 호환 python 인터페이스를 사용하기 위해서는 연관된 패키지들의 설치가 필요하다. python 운영과 관련된 다양한 패키지들이 필요하나, 설치의 편리함을 위해 아래와 같이 install.sh 명령어 실행으로 모든 설치 과정을 완료할 수 있도록 지원한다.

```

>> cd ~qasm/qiskit
>> ./install.sh

```

※ 각주: 시스템 환경에 따라 install.sh 파일에 명시한 python 패키지들을 개별적으로 설치할 수도 있다.

정상 설치 이후 python 인터페이스가 정상 설치되었는지 검증은 아래와 같은 pip 명령으로 확인할 수 있다.

```

>> pip show qplayer
Name: QPlayer
Version: 2.3
Summary: Qiskit-compatible python interface for QPlayer
Home-page: https://github.com/eQuantumOS/QPlayer
Location: /usr/local/lib/python3.10/dist-packages
Requires: cmake, matplotlib, pybind11, qiskit, scikit-build, setuptools, virtualenv

```

python 기반 양자 프로그램 개발

python 기반 코드 작성

콘솔 기반 python 양자 프로그램은 vi 또는 기타 텍스트 편집기를 통해 작성할 수 있다. 아래는 python을 이용한 양자 프로그램의 예시이다.

```

01  from qiskit import QuantumCircuit
02  from qiskit.visualization import plot_histogram
03  from qplayer import executeQASM
04  import matplotlib.pyplot as plt
05
06  qc = QuantumCircuit(2)
07  qc.h(0)
08  qc.h(1)
09  qc.measure_all()
10
11  qasm = qc.qasm()
12  result = executeQASM(qasm, 1080)
13  figure = plot_histogram(result.get_counts(), figsize=(10, 6))
14
15  plt.show();

```

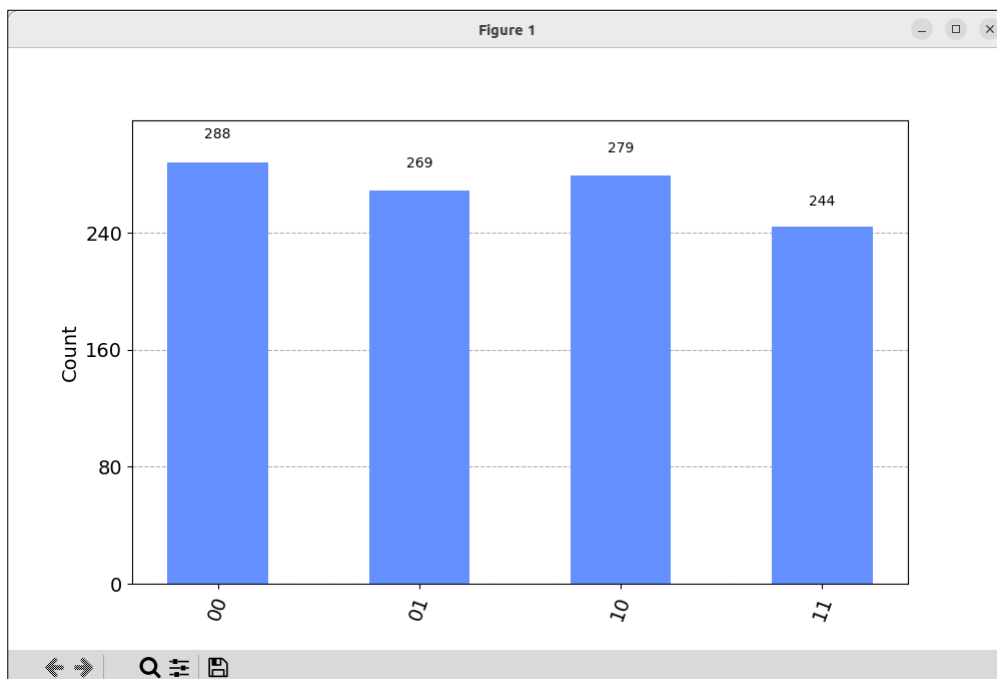
예시 프로그램에 대한 간단한 설명은 아래와 같다. 예시 코드에서 주황색은 Qiskit 인터페이스, 파랑색은 QPlayer를 위한 인터페이스를 의미한다.

- 01-02 라인은, Qiskit 라이브러리 사용을 위한 선언부이다.
- 03 라인은, QPlayer 실행을 위한 선언부이다.
- 04 라인은, python 표준 시각화 라이브러리 사용을 위한 선언부이다.
- 06-09 라인은, Qiskit 인터페이스를 활용한 양자 회로를 의미한다.
- 11-12 라인은, 작성된 양자 프로그램을 OpenQASM 2.0 언어로 변환 후 QPlayer를 실행한다.
- 13 라인은, QPlayer의 실행 결과를 Qiskit 인터페이스를 사용하여 시각화한다.
- 15 라인은, python 표준 라이브러리를 이용하여 결과 이미지를 콘솔의 새로운 창에 출력한다.

python 기반 코드 실행

작성된 코드는 아래와 같이 python 명령어를 이용하여 실행할 수 있다. 실행이 완료되면 matplotlib을 이용한 시각화 독립 결과 창이 화면에 보여진다.

» python3 test.py



Jupyter Notebook 활용

표준 CLI 이외에도 Conda와 같은 가상환경에서 Jupyter Notebook으로도 손쉽게 양자 프로그램의 작성 및 실행을 할 수 있다. 본 절에서는 Jupyter Notebook 사용을 위한 필요 패키지 설치 및 사용 방법에 대해 설명한다.

conda-forge 설치¹

1) Miniconda 설치

- wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
- sh Miniconda3-latest-Linux-x86_64.sh

2) conda 채널 변경

- cd ~/miniconda3/bin
- ./conda config --add channels conda-forge
- ./conda config --set channel_priority strict

3) 콘다 초기화

- cd ~/miniconda3/bin
- ./conda init

4) 환경 변수 재설정

- source ~/.bashrc

conda 가상환경 생성 및 활성화

- » conda create -n qplayer -c conda-forge
- » conda activate qplayer

Jupyter notebook 설치

- » conda install jupyter

conda 가상환경의 필요 패키지 설치

- » cd ~qasm/qiskit
- » ./install.sh

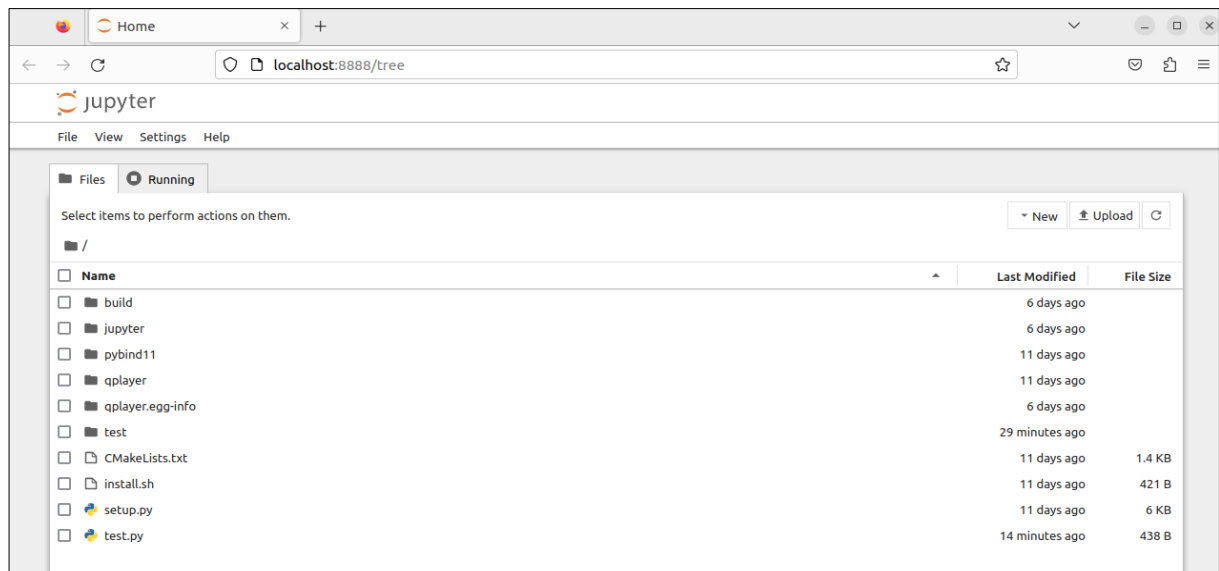
Jupyter notebook 실행

- » jupyter notebook (일반 유저)

¹ Jupyter Notebook 사용을 위한 가상 환경으로 무료 버전인 conda-forge 의 사용을 권장한다. Anaconda 의 경우 라이선스 유료화 정책에 따라 사용 비용이 발생할 수 있다.

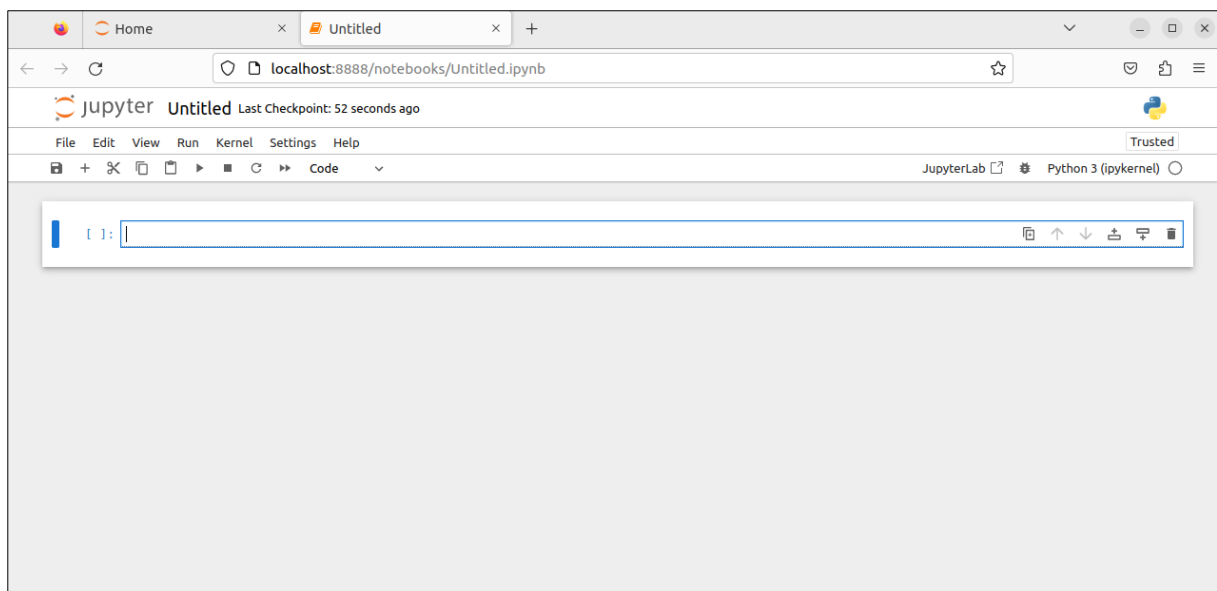
» `jupyter notebook --allow-root` (루트 유저)

상기 명령을 실행하면 아래와 같은 Jupyter notebook 화면이 기동된다.

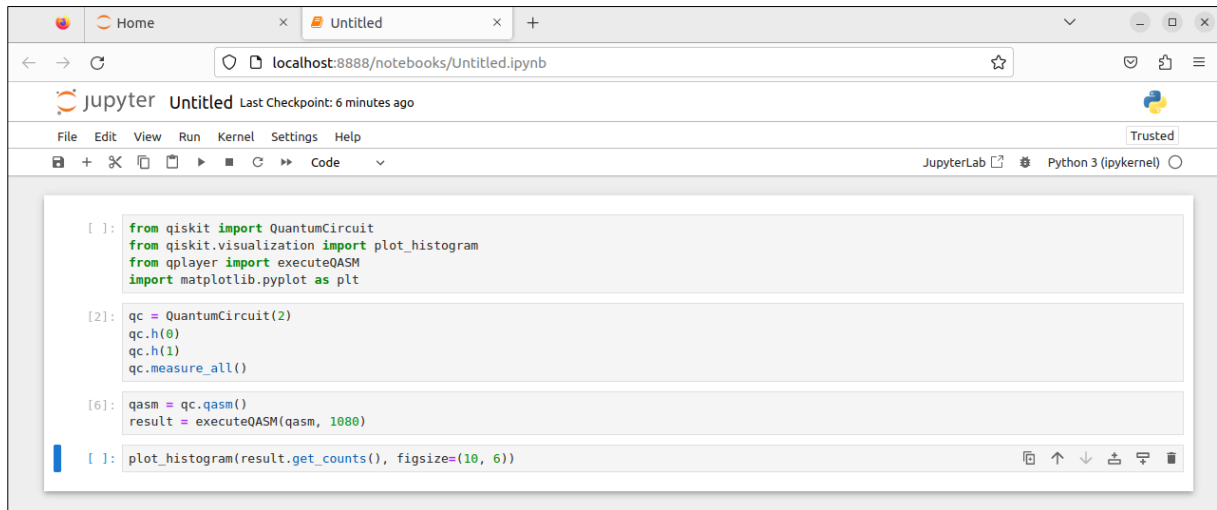


Jupyter notebook 기반 python 코드 작성

메뉴에서 File - New - Notebook을 선택하면 새로운 프로그램 작성 화면이 나타나며, 커널 선택 메뉴에서 "Python 3 (ipykernel)"을 클릭하면 아래와 같은 빈 프로그램 화면을 볼 수 있다.



이후 새로운 줄을 추가하면서 Qiskit 인터페이스를 이용한 양자 프로그램을 작성할 수 있다. 다음은 앞선 CLI 기반 양자 프로그램을 동일하게 작성한 예제이다.



The image shows a Jupyter Notebook interface in a web browser. The browser address bar shows 'localhost:8888/notebooks/Untitled.ipynb'. The Jupyter Notebook title is 'Untitled' and it shows 'Last Checkpoint: 6 minutes ago'. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The code cell contains the following Python code:

```
[ 1]: from qiskit import QuantumCircuit
      from qiskit.visualization import plot_histogram
      from qplayer import executeQASM
      import matplotlib.pyplot as plt

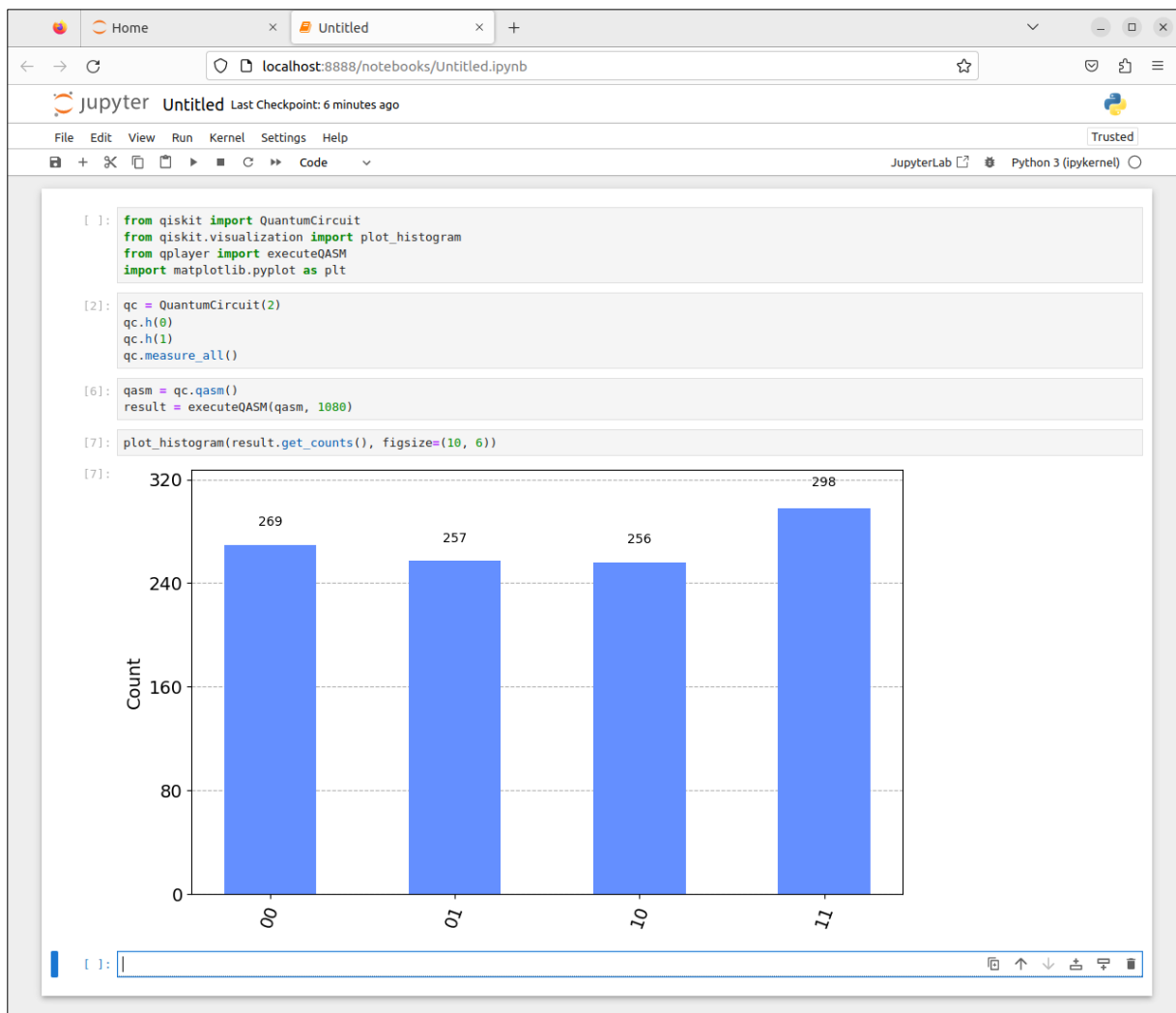
[2]: qc = QuantumCircuit(2)
      qc.h(0)
      qc.h(1)
      qc.measure_all()

[6]: qasm = qc.qasm()
      result = executeQASM(qasm, 1080)

[ ]: plot_histogram(result.get_counts(), figsize=(10, 6))
```

Jupyter notebook 기반 python 코드 실행

프로그램을 작성 후 상단 아이콘 메뉴의 실행 버튼(▶)을 클릭하면 아래와 같은 결과 화면을 볼 수 있다.



CHAPTER 5 Native C-like Interface

이 장에서는...

- 클래스
- 게이트 함수
- 측정 함수
- 유틸리티 함수
- 양자 프로그램 예제

IT 개발에 익숙한 연구자들은 QPlayer 에서 제공하는 C/C++ 라이브러리를 이용하여 복잡한 양자 프로그램들을 손쉽게 개발할 수 있다. 단순 함수 또는 제한적인 if 조건문만 허용하는 OpenQASM 2.0 과 달리 native-C like 스타일의 다양한 프로그래밍 기법들과 연계하여 양자 프로그램을 작성할 수 있다.

본 장에서는 QPlayer 에서 제공하는 주요 클래스, 게이트 함수들과 이들을 이용한 프로그래밍 예제등을 설명한다.

클래스

QRegister Class

QRegister 클래스는 양자 상태를 관리하는 저장소를 관리한다. 사용자가 큐비트에 양자 게이트를 적용하기 위해서는 사전에 QRegister 클래스의 인스턴트를 생성해야 한다.

클래스 규격:

멤버 함수	기능
QRegister (int qubits)	클래스 생성자 - 큐비트수에 해당하는 양자 레지스터 생성
~QRegister ()	클래스 소멸자 - 양자 레지스터 인스턴스 반환
void reset ()	양자 레지스터에 저장된 모든 상태 정보를 초기화
int getNumQubits ()	양자 레지스터에 정의된 큐비트의 수를 반환
qsize_t getNumStates ()	양자 레지스터에 저장된 양자 상태의 수를 반환

활용 예제:

```
void activate_qregister(int qubits) {
    QRegister *QReg = new QRegister(qubits)

    // execute quantum gates...

    delete QReg;
}
```

QTimer Class

QTimer 클래스는 양자 회로의 시뮬레이션 소요 시간을 측정하는데 활용하는 유틸리티 클래스이다.

클래스 규격:

멤버 함수	기능
QTimer ()	클래스 생성자
~QTimer ()	클래스 소멸자
void start ()	타이머 작동 시작

<code>void end ()</code>	타이머 작동 종료
<code>double getElapsedSec ()</code>	타이머에 의해 측정된 소요 시간을 반환 (초 단위)
<code>double getElapsedMSec ()</code>	타이머에 의해 측정된 소요 시간을 반환 (밀리초 단위)
<code>double getElapsedUSec ()</code>	타이머에 의해 측정된 소요 시간을 반환 (마이크로초 단위)
<code>double getElapsedNSec ()</code>	타이머에 의해 측정된 소요 시간을 반환 (나노초 단위)
<code>const char *getTime ()</code>	타이머에 의해 측정된 소요 시간을 문자열로 변환하여 반환

활용 예제:

```
void check_runtime(void) {
    QTimer timer;

    timer.start();

    // run your tasks...

    timer.end();

    printf("Elapsed Time: %.f nano seconds\n", timer.getElapsedNSec());
    printf("Elapsed Time: %s\n", timer.getTime());
}
```

게이트 함수

initZ(QRegister *QReg, int qubit)

주어진 큐비트의 상태를 $|0\rangle$ 으로 초기화 한다.

initX(QRegister *QReg, int qubit)

주어진 큐비트의 상태를 $|1\rangle$ 로 초기화 한다.

X(QRegister *QReg, int qubit)

X 게이트는 양자 상태의 비트를 플립 시킨다

$$|0\rangle \rightarrow |1\rangle$$

$$|1\rangle \rightarrow |0\rangle$$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Z(QRegister *QReg, int qubit)

Z 게이트는 양자 상태의 위상을 플립 시킨다

$$|0\rangle \rightarrow |0\rangle$$

$$|1\rangle \rightarrow -|1\rangle$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Y(QRegister *QReg, int qubit)

Y 게이트는 양자 상태의 비트와 위상을 플립 시킨다

$$|0\rangle \rightarrow -i|1\rangle$$

$$|1\rangle \rightarrow i|0\rangle$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

H(QRegister *QReg, int qubit)

H 게이트는 X+Z 축에 대해 양자 상태를 π 회전시킨다.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

S(QRegister *QReg, int qubit)

S 게이트는 Z 축에 대해 양자 상태를 $\pi/2$ 각도만큼 회전시킨다.

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$$

T(QRegister *QReg, int qubit)

T 게이트는 Z 축에 대해 양자 상태를 $\pi/4$ 각도만큼 회전시킨다.

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

SDG(QRegister *QReg, int qubit)

SDG 게이트는 Z 축에 대해 양자 상태를 $-\pi/2$ 각도만큼 회전시킨다.

$$SDG = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$$

TDG(QRegister *QReg, int qubit)

TDG 게이트는 Z 축에 대해 양자 상태를 $-\pi/4$ 각도만큼 회전시킨다.

$$TDG = \begin{pmatrix} 1 & 0 \\ 0 & -e^{i\pi/4} \end{pmatrix}$$

P(QRegister *QReg, int qubit, double angle)

P 게이트는 Z 축에 대해 양자 상태를 angle 만큼 회전시킨다.

$$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

U1(QRegister *QReg, int qubit, double lambda)

U1 게이트는 Z 축에 대해 양자 상태를 lambda 각도만큼 회전시킨다.

$$U1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$

U2(QRegister *QReg, int qubit, double phi, double lambda)

U2 게이트는 X 축에 대해 phi, Z 축에 대해 lambda만큼 양자 상태를 회전시킨다.

$$U2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix}$$

U3(QRegister *QReg, int qubit, double theta, double phi, double lambda)

U3 게이트는 3개의 오일러 각도에 대해 양자 상태를 회전시킨다.

$$U3(\theta, \phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda} \sin\left(\frac{\theta}{2}\right) \\ e^{i\phi} \sin\left(\frac{\theta}{2}\right) & e^{i(\phi+\lambda)} \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

RX(QRegister *QReg, int qubit, double angle)

RX 게이트는 X 축에 대해 양자 상태를 angle 만큼 회전시킨다.

$$RX(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i \sin\left(\frac{\theta}{2}\right) \\ -i \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

RY(QRegister *QReg, int qubit, double angle)

RY 게이트는 Y 축에 대해 양자 상태를 angle 만큼 회전시킨다.

$$RY(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}$$

RZ(QRegister *QReg, int qubit, double angle)

RZ 게이트는 Z 축에 대해 양자 상태를 angle 만큼 회전시킨다.

$$RZ(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

CU1(QRegister *QReg, int control, int target, double lambda)

CU1은 control의 비트가 1 인 경우 target에 U1 연산을 적용한다.

CU2(QRegister *QReg, int control, int target, double phi, double lambda)

CU2는 control의 비트가 1 인 경우 target에 U2 연산을 적용한다.

CU3(QRegister *QReg, int control, int target, double theta, phi, lambda)

CU3은 control의 비트가 1 인 경우 target에 U3 연산을 적용한다.

CH(QRegister *QReg, int control, int target)

CH는 control의 비트가 1 인 경우 target에 H 연산을 적용한다.

CX(QRegister *QReg, int control, int target)

CX는 control의 비트가 1 인 경우 target에 X 연산을 적용한다.

CZ(QRegister *QReg, int control, int target)

CZ는 control의 비트가 1 인 경우 target에 Z 연산을 적용한다.

CY(QRegister *QReg, int control, int target)

CY는 control의 비트가 1 인 경우 target에 Y 연산을 적용한다.

CRZ(QRegister *QReg, int control, int target, double angle)

CRZ는 control의 비트가 1 인 경우 target에 RZ 연산을 적용한다.

CCX(QRegister *QReg, int control1, int control2, int target)

CCX는 control1, control2의 비트가 모두 1 인 경우 target에 X 연산을 적용한다.

SWAP(QRegister *QReg, int qubit1, int qubit2)

SWAP 게이트는 입력된 두 큐비트의 상태를 서로 바꾼다

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

CSWAP(QRegister *QReg, int control, int qubit1, int qubit2)

CSWAP은 control의 비트가 1 인 경우 두개 큐비트에 SWAP 연산을 적용한다.

SX(QRegister *QReg, int qubit)

SX 게이트는 큐비트의 양자 상태에 Sqrt(X) 연산을 적용한다.

$$SX = \sqrt{X} = \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$$

iSWAP(QRegister *QReg, int qubit1, int qubit2)

iSWAP 게이트는 두개 큐비트의 허수(i)를 추가하여 $|01\rangle, |10\rangle$ 상태를 바꾼다.

$$|00\rangle \rightarrow |00\rangle$$

$$|01\rangle \rightarrow i|01\rangle$$

$$|10\rangle \rightarrow i|10\rangle$$

$$|11\rangle \rightarrow |11\rangle$$

$$iSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

측정 함수

int M(QRegister *QReg, int qubit)

큐비트를 진폭 확률에 기반하여 측정하고 그 결과를 반환한다. 측정된 양자 상태에 따라 반환되는 정수 값은 아래와 같다.

$$|0\rangle \rightarrow 0$$

$$|1\rangle \rightarrow 1$$

int MF(QRegister *QReg, int qubit, int state)

큐비트를 진폭 확률과 무관하게 주어진 state로 큐비트를 강제 붕괴시키고 그 결과를 반환한다.

※ 이 연산은 양자 회로의 디버깅 또는 알고리즘 검증 등의 용도로 사용될 수 있다.

유틸리티 함수

void dump(QRegister *QReg)

주어진 양자 레지스터에 저장된 양자 상태들을 화면에 출력한다.

void getMemory(uint64_t *memTotal, uint64_t *memAvail, uint64_t *memUsed)

시스템 또는 프로세스에서 사용하고 있는 메모리 사용 정보를 반환한다

- memTotal: 시스템에 장착된 최대 메모리 용량 (bytes)
- memAvail: 시스템에서 가용한 메모리 용량 (bytes)
- memUsed: 시뮬레이션 프로세스에서 사용중인 메모리 용량 (bytes)

양자 프로그램 예제

Quantum hello world !!

```
#include "QPlayer.h"

int main(int argc, char **argv)
{
    QRegister *QReg = new QRegister(3)           // 3개 큐비트를 가진 양자 레지스터 생성

    H(QReg, 0);                                  // 0번 큐비트에 Hadamard 연산 적용
    CX(QReg, 0, 1);                              // 0을 control 큐비트로 하여 1에 CNOT 연산 적용
    Z(QReg, 2);                                  // 2번 큐비트에 Z 연산 적용

    dump(QReg);                                  // 레지스터에 저장된 양자 상태들을 화면에 출력

    delete QReg;                                 // 양자 레지스터 메모리 해제
}
```

QFT algorithm

```

#include "QPlayer.h"

using namespace std;

void QFT(QRegister *QReg, int qubits) {
    X(QReg, qubits-1);

    for(int i=0; i<qubits; i++) {
        double angle = M_PI;

        H(QReg, i);
        for(int j=i+1; j<qubits; j++) {
            angle /= 2;
            CRZ(QReg, j, i, angle);
        }
    }

    for(int i=0; i<qubits/2; i++) {
        SWAP(QReg, i, qubits-i-1);
    }

    dump(QReg);
}

int main(int argc, char **argv)
{
    int qubits = 0;
    int c;

    while ((c = getopt_long(argc, argv, "q:", NULL, NULL)) != -1) {
        switch(c) {
            case 'q':
                qubits = atoi(optarg);
                break;
            default:
                break;
        }
    }

    if(qubits == 0) {
        printf("<USAGE> : %s -q <qubits>\n", argv[0]);
        exit(0);
    }

    QRegister *QReg = new QRegister(qubits);
    qft(QReg, qubits);
}

```